2014-12-01

Replication and Knowledge Production in Empirical Software Engineering Research

Jonathan L. Krein
Brigham Young University - Provo

Replication and Knowledge Production in Empirical Software Engineering Research

Jonathan L. Krein

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Charles D. Knutson, Chair
Kevin D. Seppi
Michael A. Goodrich
William A. Barrett
Parris K. Egbert
Natalia Juristo

Department of Computer Science

Brigham Young University

December 2014

ABSTRACT

Replication and Knowledge Production in Empirical Software Engineering Research

Jonathan L. Krein
Department of Computer Science, BYU
Doctor of Philosophy

Although replication is considered an indispensable part of the scientific method in software engineering, few replication studies are published each year. The rate of replication, however, is not surprising given that replication theory in software engineering is immature. Not only are replication taxonomies varied and difficult to reconcile, but opinions on the role of replication contradict. In general, we have no clear sense of how to build knowledge via replication, particularly given the practical realities of our research field. Consequently, most replications in software engineering yield little useful information. In particular, the vast majority of external replications (i.e., replications performed by researchers unaffiliated with the original study) not only fail to reproduce the original results, but defy explanation. The net effect is that, as a research field, we consistently fail to produce usable (i.e., transferable) knowledge, and thus, our research results have little if any impact on industry.

In this dissertation, we dissect the problem of replication into four primary concerns: 1) rate and explicitness of replication; 2) theoretical foundations of replication; 3) tractability of methods for context analysis; and 4) effectiveness of inter-study communication. We address each of the four concerns via a two-part research strategy involving both a theoretical and a practical component. The theoretical component consists of a grounded theory study in which we integrate and then apply external replication theory to problems of replication in empirical software engineering. The theoretical component makes three key contributions to the literature: first, it clarifies the role of replication with respect to the overall process of science; second, it presents a flexible framework for reconciling disparate replication terminology; and third, it informs a broad range of practical replication concerns.

The practical component involves a series of replication studies, through which we explore a variety of replication concepts and empirical methods, ultimately culminating in the development of a tractable method for context analysis (TCA). TCA enables the quantitative evaluation of context variables in greater detail, with greater statistical power, and via considerably smaller datasets than previously possible. As we show (via a complex, real-world example), the method ultimately enables the empirically and statistically-grounded reconciliation and generalization of otherwise contradictory results across dissimilar replications—which problem has previously remained unsolved in software engineering.

Keywords: replication, experimentation, generalization, context analysis, multi-site joint replication, post-hoc moderator analysis, Bayesian methods, theory of conceptual frameworks, design patterns, Conway's Law

ACKNOWLEDGMENTS

I am indebted to the following individuals for sharing ideas, giving valuable feedback, and being endlessly supportive:

- My wife, Sara Krein, and my parents, Ron and Teresa Krein, for their support and positive encouragement—including listening to my oft-repeated concerns with patience as I worked through the problems and difficulties of graduate school.
- Charles Knutson (Dept. of Computer Science, Brigham Young University) for advising the dissertation—including, for allowing me to do original things by working "outside the box," for teaching me how to think in fundamentally richer ways, for forcing me to tackle big problems on my own most of the time, for providing continuous positive encouragement, and for always making time when help was most needed.
- Lutz Prechelt (Institut für Informatik, Freie Universität Berlin) and Natalia Juristo (Facultad de Informática, Universidad Politécnica de Madrid) for significantly contributing to my development as a researcher and thinker, for endless feedback throughout the dissertation's development, and especially for support to persevere during discouraging times. Additionally, Lutz for suggesting and coordinating the execution of the first joint replication, Natalia for serving on my dissertation committee, and both for contributing numerous hours to help organize and run the RESER workshop. The importance of these contributions to myself personally, as well as to the content of the dissertation, cannot be overstated.
- Howard Bahr (Dept. of Sociology, Brigham Young University), David Grandy (Dept. of Philosophy, Brigham Young University), and Dag Sjøberg (Dept. of Informatics, University of Oslo) for taking time to review the theoretical work of the dissertation.
- Charlie Morgan (Dept. of Sociology, Brigham Young University) for advising me on qualitative methods, especially grounded theory, which is an integral part of the theoretical development of the dissertation.

- Kevin Seppi and Paul Felt (Dept. of Computer Science, Brigham Young University) for advising me on Bayesian methods and for always being available for questions and consultation.

- Dennis Eggett and Gilbert Fellingham (Dept. of Statistics, Brigham Young University) for providing expert advice on the frequentist and Bayesian statistics, respectively.

- Alexander MacLean and Landon Pratt (Dept. of Computer Science, Brigham Young University) for grading the subject's solutions in the joint replication, as well as for frequent support and feedback on the research in general.

- Sabrina Bailey, Kyle Blatter, and Scott Burton (Dept. of Computer Science, Brigham Young University) for taking the lead on many of the small-scale studies, thus allowing me to cover more ground in the dissertation than otherwise would have been possible.

- Marek Vokáč (Senior Software Developer, Architect, and Database Manager, SuperOffice ASA) for contributing data from a prior study to the joint replication analysis, for member checking my assessment of his work, and for providing general feedback on the joint replication report.

- Patrick Wagstrom and Clay Williams (Software Governance, IBM T. J. Watson Research Center) for mentoring me on industrial research, which experiences have significantly contributed to the development of my ideas, as well as the ways in which I think about research.

- Peri Tarr (Software Governance, IBM T. J. Watson Research Center) for advising the initial iteration of the RESER workshop, as well as for early (but critical) advice on the dissertation topic.

- Christian Bird (Research in Software Engineering (RiSE) group, Microsoft Research) for his considerable encouragement and support of the RESER workshop, including contributing numerous hours to help organize and run it.

- Sira Vegas (Facultad de Informática, Universidad Politécnica de Madrid) for pitching in to help with the RESER workshop in 2011 when Natalia unexpectedly could not attend.
- Fabio Q. B. da Silva (Centre for Informatics, Federal University of Pernambuco) for expressing genuine gratitude and support for the RESER workshop at a time when it was most needed; and, along with so many others, for making significant contributions to the workshop year after year.
- James Herbsleb (School of Computer Science, Carnegie Mellon University), Victor Basili (Dept. of Computer Science, University of Maryland), and Mel Conway (originator of *Conway's Law*) for participating in and supporting the RESER workshop as keynote speakers; and (in addition to Mel's wife, Ruth) for being great conversationalists, engaging dinner guests, and all-around gracious people, which made the experience fun and rewarding.
- Lionel Briand (Centre for ICT Security, Reliability, and Trust, University of Luxembourg), Joerg Doerr (Institute for Experimental Software Engineering, Fraunhofer), Audris Mockus (Dept. of Electrical Engineering and Computer Science, University of Tennessee, Knoxville), Dieter Rombach (Software Engineering: Processes and Measurement Research Group, University of Kaiserslautern), Per Runeson (Dept. of Computer Science, Lund University), Dag Sjøberg (Dept. of Informatics, University of Oslo), and Peri Tarr (Software Governance, IBM T. J. Watson Research Center) for their critical support of the RESER workshop in the capacity of advisory committee members.
- All other committee members and participants of the RESER workshop for their valuable advice, support, and feedback.

Table of Contents

List of Figures xvi

List of Tables xviii

1 Introduction 1

1.1 Research Area—Empirical Software Engineering . . . 2

1.1.1 Why Study Empirical Methods in Software Engineering? . . . 2

1.1.2 Empirical Beginnings (1960–1999) . . . 3

1.1.3 Refinement of Methods (1999-2013) . . . 6

1.2 Research Topic—Empirical Replication . . . 7

1.2.1 Problems of Replication and Knowledge Production . . . 8

1.2.2 Who is Affected by Replication Problems? . . . 9

1.3 Related Work . . . 10

1.3.1 The Rate of Replication . . . 10

1.3.2 Terminology/Taxonomy and the Role of Replication . . . 11

1.3.3 Methods for Synthesizing Results across Studies . . . 12

1.3.4 Methods for Inter-Study Communication . . . 19

1.3.5 Summary of Open Problems . . . 20

1.4 Thesis Statement . . . 22

1.5 Project Description . . . 22

1.5.1 Theoretical (Outer Study) Component . . . 23

1.5.2 Practical (Inner Study) Component . . . 28

1.6 Validation 30

1.6.1 Theoretical Component 31

1.6.2 Practical Component 32

1.7 Dissertation Format (Series of Papers) 33

1.8 Contributions 35

1.8.1 Exploratory/Impact Component (RESER Workshop) 35

1.8.2 Practical Component—Conway's Law 41

1.8.3 Practical Component—Design Patterns 44

1.8.4 Theoretical Component 48

2 Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University 50

2.1 Introduction 50

2.2 Original Experiment (PatMain) 52

2.3 Replication Setup 53

2.3.1 Joint Replication Framework 53

2.3.2 Brigham Young University Specifics 55

2.4 Analysis 59

2.4.1 Grading Solutions for Correctness 59

2.4.2 Data Preparation 60

2.4.3 Statical Evaluation Method 65

2.4.4 Model Tuning 65

2.5 Results 68

2.6 Additional Context Information and Threats to Validity 70

2.6.1 Framework Concerns 71

2.6.2 Solution Grading Concerns 72

2.7 Joint Replication as a Methodology—Lessons Learned 74

2.8 Conclusions 75

3 A Multi-Site Joint Replication of a Design Patterns Experiment using Moderator Variables to Generalize across Contexts 76

3.1 Introduction 77

3.2 Objectives and Contributions 80

3.3 The PatMain Experiment 81

3.3.1 The Original Study (E_orig) 81

3.3.2 The First Replication (E_repl) 83

3.3.3 The Joint Replication (E_joint) 84

3.4 Data and Metrics 88

3.4.1 Joint Dataset 89

3.4.2 Developer Experience (*devExp*) 90

3.4.3 Pattern Knowledge (*patKnow*) 90

3.4.4 Time 90

3.4.5 Correctness 91

3.5 Analysis Methods 92

3.5.1 Why Bayesian Models? 92

3.5.2 Frequentist Models 93

3.5.3 Bayesian Models 94

3.5.4 Results Interpretation—Frequentist vs. Bayesian 96

3.5.5 Moderator Categories (low/high) 97

3.6 Analysis Results 97

3.6.1 Assessment of Heterogeneity 97

3.6.2 Moderator Analysis 102

3.6.3 Original Hypotheses and Final Results 111

3.7 Threats to Validity 120

3.7.1 Construct Validity 120

3.7.2 Conclusion Validity 121

3.7.3 Internal Validity 122
3.7.4 External Validity 122
3.7.5 Other Threats to Validity 124
3.8 Conclusions 124
3.8.1 Design Patterns 124
3.8.2 Research Methods 125
3.8.3 Research Methodology 126
3.8.4 Future Work 127
3.9 Acknowledgments 127

4 A Method for Generalizing across Contexts in Software Engineering Experiments 128

4.1 Introduction 129
4.1.1 The Tractability of Context Analysis 130
4.1.2 Research Question 131
4.1.3 Contributions 132
4.1.4 Approach 133
4.1.5 Structure of this Article 133
4.2 Related Work 134
4.2.1 Frameworks for Relating Studies 134
4.2.2 Systematic Literature Reviews 135
4.2.3 Methods for Quantitative Aggregation 135
4.2.4 Context-Sensitive Generalization 137
4.2.5 Methods for Context Analysis 138
4.2.6 The State of Results Synthesis 139
4.3 Working Example (PatMain) 140
4.3.1 The Original Study (E_orig) 141
4.3.2 The First Replication (E_repl) 141

4.3.3 The Joint Replication (E_joint) 142

4.4 The TCA Method 143

4.4.1 Joint Replication 143

4.4.2 Post-hoc Moderator Analysis 148

4.4.3 Bayesian Models 156

4.5 Application of TCA Results 169

4.6 Limitations 170

4.7 Conclusions 172

4.8 Acknowledgments 173

5 The Humanity of Science versus the Complexity of Reality: A Theoretical Study of Replication and Knowledge Production 174

5.1 Introduction 175

5.1.1 Contributions 178

5.1.2 Scope 178

5.1.3 Structure of this Article 180

5.2 Conceptual Frameworks 180

5.3 Relationship to Truth and Knowledge 185

5.3.1 Knowledge is Discrete and Partial 187

5.3.2 Conceptual Frameworks Selectively Represent Reality 190

5.3.3 Conceptual Frameworks are Prerequisite to Knowledge 196

5.3.4 Conceptual Frameworks Influence Perception and Thought 200

5.3.5 Summary of Concepts 206

5.4 Simmel's Transcendency 207

5.5 Transcending Our Conceptual Frameworks 212

5.6 Building Knowledge through Replication 216

5.6.1 Implications of Expanding Replication 217

5.6.2 A Framework for Constructing Replication Types 218

5.6.3 Initial Exploration of Axis Quality . . . 220

5.7 Additional Ideas for Consideration . . . 236

5.8 Conclusions . . . 239

5.9 Acknowledgements . . . 242

6 Future Work 243

6.1 Conway's Law . . . 243

6.2 Highly Differentiated Replication . . . 244

6.3 Design Patterns . . . 244

6.4 The TCA Method . . . 245

6.5 Replication Theory . . . 246

A Replication Web Portal 247

B Information on Sites 249

B.1 Additional Protocols . . . 249

B.2 Design Pattern Education . . . 251

C Participant Demographics per Site 253

C.1 Developer Experience . . . 253

C.2 Pattern Knowledge . . . 256

D Summary Statistics for Key Variables 258

E Unusable Data 260

F Dataset Schema Definitions 262

F.1 Experiment Metadata . . . 262

F.2 Pre-Questionnaire 1—Developer Experience . . . 263

F.3 Pre-Questionnaire 2—Design Pattern Knowledge . . . 264

F.4 Task Responses . . . 266

F.5 Final Comments 268

F.6 Survey and Task Times 268

F.7 Task Correctness Scores 271

G Data Preparation Process 272

H Variables Excluded from Analysis 274

I Derived Metrics 277

I.1 Developer Experience 277

I.2 Java Familiarity 279

J Statistical Model Assumptions 280

J.1 Normality 280

J.2 Multicollinearity 281

J.3 Heteroscedasticity 282

K Tuning Frequentist Models 285

L Discretization of Bayesian Variables 286

M Bayesian Priors 287

N Notes on Observation Filtering 289

N.1 *correctness* × *variant* Interaction 289

N.2 *correctness-time* Relationship 291

N.3 Summary 292

O Notes on Participant Filtering 294

O.1 Filtering by Correctness 294

O.2 Filtering by Time 297

P Visualization of Moderator Variables 300

Q Moderator Variables Continued 303

Q.1 Task Difficulty 303

Q.2 *correctness* and *time* 304

Q.3 Program Order 306

Q.4 Perceived Time Limits 307

Q.5 Cultural (or Regional) Variation 310

Q.6 IDE Preferences 312

Q.7 Language Barriers 313

Q.8 Clarity of Task Instructions 315

Q.9 Compilation/Testing Expectations 316

R Detailed Program/Task Descriptions 319

R.1 Decorator: Communication Channels (CO) 320

R.1.1 CO—Work Task 1 320

R.1.2 CO—Work Task 2 321

R.2 Composite and Abstract Factory: Graphics Library (GR) 321

R.2.1 GR—Work Task 1 322

R.2.2 GR—Work Task 2 323

S Comparison of Statistical Methods 324

S.1 E_orig 324

S.2 E_repl 325

S.3 E_joint 326

T Additional Results Data 328

U Threats to Validity Continued 329

U.1 Construct Validity 329

U.2 Conclusion Validity 330

U.3 Internal Validity 331

U.4 External Validity 332

V Researcher Interactions **333**

V.1 E_repl 333

V.2 E_joint 333

V.3 Summary 334

W Future Work **335**

W.1 PatMain Replications 335

W.2 PatMain Meta-analysis 336

W.3 Historical and Case Study Investigations of Moderators 338

W.4 Taxonomy of Interfering Variables 338

W.5 Design Pattern Properties 339

W.6 Studies of Motivation 339

X Frequentist Statistical Results **341**

X.1 Results Layout 341

X.2 Results Interpretation 342

Y Bayesian Statistical Results **352**

Y.1 Results Layout 352

Y.2 Results Interpretation 353

Z Example Bayesian Model Specification **364**

Z.1 Likelihood Functions 364

Z.2 Prior Distributions 369

Z.3 Complete Conditionals 369

References **374**

List of Figures

3.1 Time data for CO task 1, showing ALT versus PAT displayed by site. 98

3.2 Frequentist results for the *patKnow* × *variant* interaction (CO_time model, filtered data). 107

3.3 Relative impact that developer experience and pattern knowledge have on the effect of Decorator and Abstract Factory, generalized across all three PatMain studies (E_orig, E_repl, and E_joint). 119

4.1 Excerpt of E_joint time data showing ALT versus PAT displayed by site. . . 145

4.2 Example Bayesian network. 158

4.3 Bayesian network depicting the proposed model applied to E_joint. 159

J.1 Density plots for the *time* response variable *before* log transformation. 281

J.2 Density plots for the *time* response variable *after* log transformation. 282

J.3 Scedasticity plot for *patKnow* (CO_time model). 284

O.1 E_joint participants plotted by average task time and correctness. Plot used by the four independent reviewers to select the filter threshold. 295

O.2 E_joint participants plotted by average task time and correctness. Same as Figure O.1, but with participants categorized by site. 295

O.3 E_joint participants plotted by average task time and correctness. Same as Figure O.2, but including participant IDs and filter threshold. 296

O.4 Post-hoc validation of the filter threshold, showing questionable data from E_joint, as well as professionals from E_orig. 296

O.5 Time data, showing ALT versus PAT displayed by site. 298

O.6 Correctness data, showing ALT versus PAT displayed by site. 299

Q.1 Relative impact that developer experience has on the effect of design patterns, taking into account the co-moderating influence of task difficulty. 304

Q.2 Participants plotted by average task time and correctness, categorized by site. Repeat of Figure O.2 from Appendix O. 308

List of Tables

1.1 Publications resulting from this dissertation. 34

1.2 RESER workshop stats. 37

2.1 Summary statistics for dependent variables *time* and *correctness*. 62

2.2 Statistical significance (p-values) of final model variables (fixed effects). . . . 68

2.3 Estimates for the *task * version* interaction. 70

3.1 Frequentist statistical models. 94

3.2 Bayesian model interactions. 96

3.3 Frequentist model p-values for *site* and *variant*. 99

3.4 Frequentist model p-values for *devExp* and *patKnow*. 101

3.5 Bayesian interaction results—moderator assessment. 104

3.6 Overview of the PatMain programs, tasks, and hypotheses. 112

3.7 Comparison of results across the three PatMain studies. 113

B.1 General information about the experiment and participants at each of the four sites. 250

C.1 Summary statistics for the developer experience pre-questionnaire. 254

C.2 Summary statistics for the design pattern knowledge pre-questionnaire. . . . 255

D.1 Summary statistics for key explanatory variables. 258

D.2 Summary statistics for response variables. 259

J.1 Results for the Shapiro-Wilk test of normality for the *time* response variable before/after log transformation. 283

M.1 Prior distributions for all Bayesian model parameters. 288

N.1 Unfiltered Bayesian results showing the *correctness* × *variant* interaction and the marginalized effect of *variant* for each of the four tasks. 290

N.2 Unfiltered frequentist and Bayesian results showing the significance of *correctness* as a covariate in the *time* models (and vice versa). 292

P.1 Bayesian interaction results—moderator assessment. Expanded version of Table 3.5 from Section 3.6.2. 301

Q.1 Frequentist model p-values for *correctness* and *time*. 305

Q.2 Bayesian interaction results—moderator assessment. Excerpt from Table 3.5 in Section 3.6.2, showing models T4 and C4. 305

Q.3 Frequentist and Bayesian results showing the significance and effect of program *order*. 306

Q.4 Average task time and correctness for each participant. 309

Q.5 Prevalence of participants at each site who complained that the task instructions were difficult to understand. 315

Q.6 Comparison of participants—those who complained that the task instructions were difficult to understand versus those who did not. 317

T.1 Comparison of results across the three PatMain studies. Supplement to Table 3.7 in Section 3.6.3. 328

X.1 CO_time, unfiltered. Type 3 tests of fixed effects. 344

X.2 CO_time, unfiltered. Solution for fixed effects. 344

X.3 CO_time, unfiltered. Marginal means. 344

X.4 CO_time, unfiltered. Differences for marginal means. 344

X.5 CO_correctness, unfiltered. Type 3 tests of fixed effects. 345

X.6 CO_correctness, unfiltered. Solution for fixed effects. 345

X.7 CO_correctness, unfiltered. Marginal means. 345

X.8 CO_correctness, unfiltered. Differences for marginal means. 345

X.9 GR_time, unfiltered. Type 3 tests of fixed effects. 346

X.10 GR_time, unfiltered. Solution for fixed effects. 346

X.11 GR_time, unfiltered. Marginal means. 346

X.12 GR_time, unfiltered. Differences for marginal means. 346

X.13 GR_correctness, unfiltered. Type 3 tests of fixed effects. 347

X.14 GR_correctness, unfiltered. Solution for fixed effects. 347

X.15 GR_correctness, unfiltered. Marginal means. 347

X.16 GR_correctness, unfiltered. Differences for marginal means. 347

X.17 CO_time, filtered. Type 3 tests of fixed effects. 348

X.18 CO_time, filtered. Solution for fixed effects. 348

X.19 CO_time, filtered. Slopes for *patKnow* × *variant*. 348

X.20 CO_time, filtered. Marginal means. 348

X.21 CO_time, filtered. Differences for marginal means. 348

X.22 CO_time, filtered. Differences for *patKnow* × *variant*. 348

X.23 CO_correctness, filtered. Type 3 tests of fixed effects. 349

X.24 CO_correctness, filtered. Solution for fixed effects. 349

X.25 CO_correctness, filtered. Marginal means. 349

X.26 CO_correctness, filtered. Differences for marginal means. 349

X.27 GR_time, filtered. Type 3 tests of fixed effects. 350

X.28 GR_time, filtered. Solution for fixed effects. 350

X.29 GR_time, filtered. Marginal means. 350

X.30 GR_time, filtered. Differences for marginal means. 350

X.31 GR_correctness, filtered. Type 3 tests of fixed effects. 351

X.32 GR_correctness, filtered. Solution for fixed effects. 351

X.33 GR_correctness, filtered. Marginal means. 351

X.34 GR_correctness, filtered. Differences for marginal means. 351

Y.1 Unfiltered Bayesian Results. 354

Y.2 Filtered Bayesian Results. 359

Z.1 Templates for the 12 Bayesian models. 368

Z.2 Prior distributions for all Bayesian model parameters. 370

Chapter 1

Introduction

> *The full potential of this powerful concept can only be used if it is named, recognized, and addressed systematically. So far, replication marks a blind spot in the social sciences' tool box. To change this situation we have to discuss replication extensively. We should reflect on replication aspects in our own research with every experiment we set up. Furthermore, we need to add this topic to our textbooks and to teach it to our students. Finally, we have to discuss the editorial policies of our journals and the handling of the replication issue in the review process.*
>
> —Schmidt (2009), on replication in psychology [186, p. 99]

The field of *software engineering* involves the study and application of "a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [93, p. 67]. Within that field, the area of *empirical software engineering* specializes in the formulation, refinement, and application of experience-based (i.e., empirical) research methods to the study of software—including both *experimental* methods, wherein the researcher controls the context of study, and *observational* methods, wherein the researcher does not exercise control. The aim of this dissertation is to address longstanding problems of *replication* and *knowledge production* in empirical software engineering research.

We begin with an overview of the research area (empirical software engineering) in Section 1.1, followed by an introduction to the research topic (empirical replication) in Section 1.2. In Sections 1.3 and 1.4, we discuss related work and present the thesis statement. In Sections 1.5 and 1.6, we describe the research project and validation methods. In Section 1.7, we describe the format of the dissertation (i.e., as a series of papers), and in Section 1.8, we summarize the role and contributions of each chapter/paper.

1.1 Research Area—Empirical Software Engineering

In this section, we present an overview of empirical software engineering—including its importance as a research area, its early development, and its primary contributions to date.

1.1.1 Why Study Empirical Methods in Software Engineering?

A study of empirical methods is needed in software engineering for several reasons:

First, software engineering presents a unique blend of research challenges, including: many of the technical and social complexities of modern software projects are difficult or impossible to experimentally simulate; context variables are numerous such that experimental results vary across replications; and proprietary concerns prevent access to most software organizations for observational studies [15, 19, 206]. While we may profit from studying similar challenges in other fields [206], we must still tailor our methods to our specific and evolving circumstances.

Second, modern software is as much a product of social dynamics as it is of technical concerns [188]. Consequently, software engineering requires the integration of empirical methods traditionally foreign to computer science, which must be adapted from other fields (e.g., the social sciences) [1, 188]. Qualitative methods, in particular, have only recently become well-recognized in software engineering, and the work of refining them for use in software contexts is still ongoing [1].

Third, it should be noted that any research discipline requires a continuous effort to monitor its progress so as to maintain scientific integrity. Even mature fields, such as medicine, have been found in recent years to suffer from pervasive "methodologic errors" [113].

1.1.2 Empirical Beginnings (1960–1999)

Software development as a process has long been notorious for high variability. In 1968, Sackman et al. [182] published one of the earliest known controlled studies of programmer behavior. The study tested debugging performance under conditions of online and offline access to a computer. Although the study found debugging to be significantly faster under online conditions, it also revealed striking variability in individual programmer performance—the legendary 28:1 ratio. Although the 28:1 ratio has since been shown to be overestimated [165], the Sackman study is important because it was a high-profile result that helped motivate the emergence of empirical software engineering.[1]

In response to problems of software variability, NASA (in cooperation with the University of Maryland) founded the empirically-based Software Engineering Laboratory (the SEL) in 1976 under the direction of Victor Basili [17]. The SEL's original focus was to study software process via case study methods using NASA projects as a data source. The SEL, and in particular Basili's work, helped form the foundation of empirical software engineering; many of the SEL's papers are still cited today.

One of Basili's earliest contributions (developed in the 1980s as part of David Weiss's PhD thesis) was the Goal/Question/Metric paradigm (GQM) [14]. GQM is essentially a framework for structuring the empirical investigation of software processes. The framework involves three broad phases: 1) define and map operational goals to software measurements, 2) implement data collection, and 3) define and implement feedback cycles for improvement.

Part of the purpose of GQM and other such paradigms proposed by Basili et al. was to develop process models that can be reused across a variety of projects [18]. Unlike manufacturing, Basili argued, software does not involve reproducing the same object; each product is different from the last; contexts, goals, and requirements vary from project to project [15]. In the late 1980s, Basili and Rombach published a paper about the TAME (Tailoring a

[1] Other high-profile examples include Frederick Brooks's 1975 book [33], *The Mythial Man Month*, a 1979 paper by Norman Augustine [9], *Augustine's Laws and Major System Development Programs*, and a 1981 book by Barry Boehm [26], *Software Engineering Economics.*

Measurement Environment) project model [18]. TAME was essentially an instantiation of the GQM paradigm for empirically evaluating software projects in a “tailorable” and “tractable” way. In addition to presenting the project model itself, the paper also described a prototype to automate the model.

Despite the work of Basili and others, empiricism was still not the norm in the 1990s. The general lack of emphasis on empiricism was partly due to a continued focus on technical problems. However, even in technical areas claims were being made about techniques and frameworks which were not being empirically evaluated. In response, Basili et al. commented, “Unfortunately, in computer science and, more specifically, in software engineering, the balance between evaluation of results and development of new models is still skewed in favor of unverified proposals” [19, p. 456].

Given the general lack of empirical evaluation, coupled with a rapid growth in the complexity and proliferation of software, it is not surprising that several significant software mishaps occurred in the 1980s and 1990s. A famous case is the Therac-25 medical electron accelerator, which over the course of two years (1985–1987) administered massive overdoses of radiation to at least six patients, resulting in serious injury and death [132]. The accidents, it turned out, were the result of software errors, which surfaced only in response to complex interactions between technical, organizational, and managerial factors [132]. In response to the Therac-25 accidents and other such incidents, Leveson commented that “we may be straining at the limits of what we can do effectively without better inventions based on known scientific and engineering principles” [131, p. 7]. In the early 1990s, Leveson and Turner conducted root-cause analysis on the Therac-25 accidents [132]. Leveson’s efforts to analyze and document software failures, particularly in light of historical technology mishaps (e.g., high-pressure steam engines [131]), helped to motivate a culture of empiricism in software engineering.[2]

[2] Another famous case is the 1996 catastrophic failure of the ARIANE 5 satellite launch, the direct cost of which was approximately $370 million [58]. The initial inquiry board reported that software faults were the cause of the failure [136], but it was later shown that the software specification and design processes were primarily to blame [58].

In the mid 1990s, Tichy, Basili, and others began publishing articles advocating empirical methods (especially experimentation) in software engineering. Tichy et al. [206] surveyed the literature (400 articles) and showed that computer scientists were, at that time, publishing few papers with experimentally validated results. They compared the software engineering literature with that of other engineering fields. For other fields, the percentage of papers that needed empirical validation, but lacked it, was only 12–15%. In software engineering, however, the percentage was much higher, on the order of 50%. In 1996, Basili published an article advocating the need for empirical methods and describing experimentation as one such component [15]. The article was one of the first to analyze the role of experimentation specifically within software engineering.

Work on software metrics continued in the 1990s. However, many researchers turned their focus toward validation. In particular, Kitchenham et al. developed a framework for software measurement validation [112]. The framework addressed several key questions, such as how to validate a metric, how to assess the validation work of others, and how to determine when it is appropriate to apply a given metric. Basili et al. also conducted one of the first systematic, empirical evaluations of software engineering metrics [16]. Their study examined six design metrics (including coupling and cohesion) to determine whether those metrics could be used as early quality indicators. Other significant contributions include Briand's efforts to define and validate metrics such as coupling and cohesion [29] and to develop unified measurement frameworks [27, 28].

In 1995, Brooks et al. [31, 32] published the first systematic discussion of replication in software engineering. Their arguments focused mostly on experimental studies, with validation as the primary goal for replication. In 1999, Basili et al. [19] extended the discussion by introducing the concept of *families of experiments.* A *family* is a framework for organizing sets of related studies, where "related studies" can include more than replications. The idea is that by synthesizing results across a broader set of studies, important insights can surface that would not otherwise be apparent. The systematic juxtaposition of related studies can also guide future work.

1.1.3 Refinement of Methods (1999-2013)

Over the last fifteen years, the focus of empirical software engineering has shifted from promoting empiricism to articulating and refining specific research methods:

Experimentation. In 2002, Kitchenham et al. [113] published a set of guidelines for empirical research, which they adapted from the medical field. The guidelines, which concern all levels of the research process (including researchers, reviewers, and meta-analysts), mention observational studies, but focus primarily on experimentation. The guidelines address six areas of experimentation: context, design, conduct and data collection, analysis, results reporting, and results interpretation.

Evidence-based software engineering. Evidence-based medicine (EBM) is an approach to identifying and aggregating best research across studies, integrating that research with clinical expertise, and then transferring the resulting knowledge into practice. In 2004, Kitchenham et al. [110] proposed an adaptation of the organizational and technical infrastructure supporting EBM to the context of software engineering (EBSE). In their analysis, they found two complicating factors inherent in software development—the skill factor and the lifecycle factor—which would need to be addressed in order to make EBSE successful.

Systematic literature reviews. Kitchenham has published several papers on methods for systematic literature reviews (SLRs). Her work is currently the authoritative standard for SLR research in software engineering. Kitchenham's most cited paper on the topic is a 2004 technical report derived from three existing guidelines used by medical researchers, which she adapted to the context of software engineering [107]. The adapted guidelines cover three phases of the SLR process: planning the review, conducting the review, and reporting the review.

Reporting guidelines. In 2005, Jedlitschka and Pfahl proposed reporting guidelines for controlled experiments [96]. These guidelines are a distillation of prior work in software engineering, as well as a synthesis of guidelines from other fields (medicine and psychology). Although they were presented as a proposal, they currently constitute a de facto standard.

Case studies. Case study methods have been used to study software engineering since at least the early 1970s [17]. However, Kitchenham et al. [114] were the first (in 1995) to publish guidelines on the topic. Later, in 2009, Runeson and Höst [179] developed a more comprehensive set of guidelines based in part on guidelines from other fields (the social sciences and information systems). While both papers are well cited, Runeson and Höst's work is the contemporary authority, providing guidelines for both researchers and readers of case studies.

Qualitative methods. In 1999, Carolyn Seaman published one of the first papers on qualitative methods [188]. Her work came in response to a growing interest among both researchers and practitioners to investigate human factors. Seaman discusses an array of data collection and analysis methods, including participant observation, interviewing, coding, constant comparison, and cross-case analysis. Over the last decade, qualitative methods have gained in popularity. Grounded theory, in particular, is now well-recognized. In 2011, Adolph et al. published one of the first systematic explanations of grounded theory specifically tailored to the context of software engineering [1].

1.2 Research Topic—Empirical Replication

The software engineering literature is replete with statements advocating replication—such as, "replication is a key feature of experimentation in any scientific or technological field" [104, p. 295], "replication is a basic component of the scientific method, so it hardly needs to be justified" [108, p. 219], or more simply, "The value of experimental replications is evident..." [40, p. 1]. As a research community, we believe that replication is essential to knowledge production.

In its most basic form, replication can be defined simply as “repeating a study” [135, p. 217]. However, a number of practical variables influence replication outcomes, such as researchers, measurement instruments, measures, population, research location, study design, analysis methods, etc. [84]; thus, many types of replication can be imagined, each with strengths and weaknesses [83, 84]. Replication has two basic purposes [11]:

1. To validate findings (i.e., the reliability test).
2. To investigate the sources of variability that influence a given result (i.e., the generality test).

Most results in software engineering research suffer from threats to validity that can be addressed by replication. These threats include [31, 32, 189]:

- Lack of independent validation for empirical results.
- Contextual shifts in software engineering practices or environments since the time of the original studies.
- Limited data sets at the time of the original research studies.

1.2.1 Problems of Replication and Knowledge Production

Despite wide acceptance of its importance, replication is still uncommon in practice. For instance, da Silva et al. [49] found only 133 replications of 72 original studies in a survey of more than 16,000 articles (from 1994–2010), including all major software engineering publications. Several factors inhibit and/or discourage replication [31, 32, 108, 189, 190]:

- A perception exists that replication studies are less valuable than original studies.
- Original datasets are frequently unavailable.
- Published reports are often insufficiently detailed to allow replication [40, 96].
- Research tools are often unavailable and/or inoperable, rendering precise replication impractical.

Additionally, innovation in our field, though not necessarily more rapid than elsewhere [194], complicates replication. For instance, studies relying on evolving programming languages or development frameworks are both more difficult to replicate and in greater need of replication.

When effectively practiced, replication presumably has the power to build knowledge [32]. As Juristo and Vegas note, "After several replications have increased the credibility of the results, the small fragment of knowledge that the experiment was trying to ascertain is more mature" [102, p. 356]. However, there is little discussion in the literature on the actual mechanisms by which replication *matures* knowledge (especially in the face of contradictory results), and the obvious function of replication as a guardian against experimental mistakes and fabrication does not fully satisfy that question. Moreover, "exact" (or strict) replication, for the purpose of validating results, is currently infeasible for all but the most small-scale studies, and even in those cases we often cannot reproduce results [49, 102, 104] (e.g., [103, 126, 155, 167]). For instance, only 65% of published replications from 1994–2010 [49] were able to successfully reproduce the original results. Worse still, only 26% of external replications—i.e., replications performed by researchers other than those who conducted the original study [4, 32]—were able to successfully do so (as compared to 82% of internal replications) [49]. In other words, as a research field, we are largely failing to produce *usable* (i.e., transferable) knowledge; key details are either not being sufficiently communicated across studies or our conclusions are too brittle to generalize across contexts. Thus, as Juristo and Vegas have indicated, "we might be dealing with the issue of [software engineering] experiment replication from too naive a perspective" [102, p. 356].

1.2.2 Who is Affected by Replication Problems?

Replication issues affect many areas of software engineering research. For example, Kapfhammer explains [106, 158] that industry adoption of regression testing techniques is currently inhibited by a lack of empirical evidence, primarily due to problems replicating experiments. Similarly, Runeson et al. recently assessed inspection versus unit testing via a series of experimental replications [181]; not only did the results differ across the replications, but "the differences in the instrumentation and the between-experiment participants themselves were larger than the differences between inspection versus unit testing" [181, p. 35]. Besides software testing [76], replication issues also affect research in software engineering education [137], software requirements [70, 176, 216], software process [151], software architecture (e.g., design patterns [212, 222]), mining software repositories [86, 140, 177], effort/cost estimation [97, 111, 141, 146, 207], defect/fault prediction [170, 214], safety analysis for embedded systems [99], programming languages [117], etc.

1.3 Related Work

In this section, we review related work. Topics include: the rate of replication, terminology/taxonomy and the role of replication, methods for synthesizing results across replications, and methods for inter-study communication. We conclude with a summary of open problems in replication research.

1.3.1 The Rate of Replication

Several researchers have surveyed the literature on replication studies—including Sjøberg et al. [197], Almqvist [4], and da Silva et al. [49]. The surveys all show a low rate of replication. For example, of the 16,000+ articles surveyed by da Silva et al., only 96 (or <0.6%) include a replication. Further, from 1994–2003, an average of only 4 replications were published per year (2 internal, 2 external). From 2004–2010 the average increased to 13 (10 internal, 3 external)—but that increase also corresponds to an increase in internal replications being published in the same articles as the original studies. Thus, the increase may simply be due to a shift in publication strategy.

Noting similar problems in psychology, Schmidt [186] points out that more replications probably occur in practice than show up in the surveys. To a degree, this is good news, since the problem is likely not as bad as the numbers indicate. However, Schmidt also argues that to be a science, we must be systematic about things. In his words, “With this lack of explicitness comes along a lack of a systematic approach” [186, p. 96].

1.3.2 Terminology/Taxonomy and the Role of Replication

A number of terms are currently used in the software engineering literature to refer to a variety of replication types, including: strict, close, similar, exact, non-exact, differentiated, conceptual, literal, theoretical, independent, dependent, internal, external, etc. [4, 19, 32, 102, 104, 120, 137, 190]. Many of these terms appear to denote similar concepts, but often a deeper reading uncovers subtle divergences in the authors’ intended meanings. More problematic, some terms are reused by different authors to convey completely different meanings. Many authors also describe replication taxonomies (e.g., [4, 19, 32, 190]), almost all of which conceptually overlap. However, most of the taxonomies also diverge in such a way that we cannot reconcile them by simply mapping terminology [49, 52].

Addressing issues of replication in psychology, Schmidt stated in 2009 [186] that no established taxonomy of replications exists in any scientific field (including definitions, meaning, usage, etc.). Similarly, in 2010, Gómez et al. [83, 84] surveyed other scientific fields and found 18 taxonomies covering 79 (somewhat-overlapping) replication types. In response to the conceptually disparate state of replication, Schmidt [186] further argued that we need to be more systematic, especially considering the critical role that replication plays in the scientific method. In response to the many replication terms, de Magalhães and da Silva [52] have recently proposed (in 2013) to develop a comprehensive taxonomy based on empirically-derived replication needs and concerns.

Given the state of replication terminology, it is not surprising that opinions on the role of replication also conflict. For instance, Brooks et al. [32] focus mostly on replication as a validation process—the idea that we can only have confidence in our findings if they are repeatable. Conversely, Shull et al. [190] focus mostly on generalizability—i.e., using replication to understand the sources of variability that influence a given result. Shull et al. actually recognize both goals (reliability and generality testing), but simply place much greater emphasis on the latter.

Interestingly, Shull et al. also discourage highly differentiated (or independent) replications, arguing that the costs are too high in most cases, given the likelihood of contradictory results. In direct rebuttal, Kitchenham [108] argues that sharing materials between studies can also be costly; doing so not only propagates experimental errors, but it violates a key assumption of meta-analysis, which is study independence. Moreover, as Kitchenham argues, generalization to industry is our primary goal, and independent replications provide much more confidence in that regard. In addition to these viewpoints, Mäntylä et al. [142] argue in favor of broadening our view of replication to admit more than controlled experiments; in particular, they argue the value of replicating case studies.

Given the many disparate opinions on replication in the literature, Natalia Juristo concluded in her 2013 ESEM keynote address [100]:

> There is no agreement yet on terminology, typology, purposes, operation and other replication issues. There is not even agreement on what a replication is! Different authors consider different types of changes to the baseline experiment as admissible.[3] [101, slide 18]

1.3.3 Methods for Synthesizing Results across Studies

In this section, we review methods for synthesizing results across studies. We include this topic in the related work because synthesis is necessary in order to build knowledge. Contrary to common belief, replication does not by itself produce knowledge (nor validation, nor generalization). Rather, replication simply produces an additional set of results. To say

[3] Since Juristo's keynote in 2013, and since the drafting of this dissertation in 2014, Gómez, Juristo, and Vegas have published a comprehensive taxonomy of replication types [85]. The Gómez taxonomy was developed by surveying replication taxonomies from other fields (from which the authors classified the experimental elements that can vary between a replication and its baseline experiment, as well as the various functions that a replication can serve in the research process). Gómez et al. conclude by showing how their taxonomy maps onto, and thus represents a superset of, the existing software engineering taxonomies. Although the Gómez taxonomy shows promise as a unifying classification scheme, insufficient time has passed to determine whether it is functionally viable and/or will become a commonly accepted standard.

something of validation and generalization, and ultimately to produce knowledge, requires some type of *synthesis across studies*, taking into account contextual and execution-related differences between those studies. Thus, when we talk of replication as a mechanism for building knowledge, we are necessarily referring to synthesis as well.

Frameworks for Relating Studies

A number of papers describe frameworks for relating studies in order to build knowledge:

Daly et al. [51, 148, 219] describe a multi-method approach that calls for a series of studies, each of which relies on a different empirical method (e.g., structured interviews, followed by a survey, followed by a laboratory experiment). By varying the research method, Daly et al. argue, the results can be shown to be more robust—at least inasmuch as they agree across studies.

Basili et al. [19] propose a framework for organizing *families* of experiments. The idea is to group studies that address similar concepts for the purpose of forming meta-level conclusions and identifying experimental gaps in the underlying theory. In a sense, Basili's concept of *family* is simply a broadening of the traditional concept of replication.

Lastly, Krein and Knutson present a framework for relating replications [120]. The framework is essentially a conceptual diagram showing how various replication types fit into the overall knowledge-building process of science.

Ultimately, each of these three frameworks adds insight to the issue of replication, but none provides a clear-cut practical method for synthesizing results.

Systematic Literature Reviews

Systematic literature reviews (SLRs; e.g., [61, 111]) represent another possible solution to the problem of synthesis. According to Kitchenham, SLRs are "a means of identifying, evaluating and interpreting all available research relevant to a particular research question" [107, p. 1]. Unfortunately, Cruzes and Dybå [48] report finding *synthesis* to be the single most challenging (and neglected) component of SLR research—almost half of the studies surveyed did not contain any synthesis, and of those that did, two thirds performed only basic thematic and narrative synthesis. Further, of the synthesis methods identified by Cruzes and Dybå, most are heavily or entirely qualitative. Only one, meta-analysis, was found to be fully statistical. Below we summarize meta-analysis, as well as two other approaches not mentioned by Cruzes and Dybå.

Methods for Quantitative Aggregation

Meta-analysis is the current standard for aggregating quantitative results across studies [108]. Meta-analysis is a process of pooling data to increase the number of observations, thereby reducing statistical error [56, 160]. Meta-analysis can be used to combine data even in cases where studies report contradictory results, as long as the overall variance is not too extreme [160]. However, experiment variables must match and, to some degree, the studies must be independent [108].

Bayesian methods [54, 205] are an alternative to traditional meta-analysis which allow data to be accumulated over time from a series of experiments by incorporating past results as prior knowledge into the analysis of future replications. Bayesian methods can also be used to combine results such that all data are treated as current observations, in which case additional parameters can be used to account for differences between studies. Either way, Bayesian methods yield posterior probabilities, which can be preferable over p-values in a variety of circumstances (e.g., when statistical power is low, as we discuss in Section 4.4.2). Bayesian methods also naturally handle missing data [54].

A third option is to simply analyze all data together using traditional frequentist methods (as is done by Runeson et al. [180])—e.g., analysis of variance (ANOVA), multiple regression, or mixed models. Like the Bayesian approach, additional variables can be incorporated to account for differences between studies.

The difficulty of the latter two options (i.e., the Bayesian and frequentist options) is that they are both a form of *raw-data aggregation*, and so both require raw data to be available from all studies involved, which is often not the case in software engineering. Further, even when the necessary data are available, those data may be poorly documented or inconsistently measured [23], thus making the analysis fraught with assumptions and error prone. Consequently, meta-analysis is currently the de facto standard.

However, meta-analysis is not without limitations. In 2000, Miller [147] applied it to a set of software engineering experiments, but found the results to be highly unstable. Miller concluded that the root cause was cross-study contextual variation. Other software engineering researchers have also reported finding high levels of contextual variation [19, 74, 137, 188]—e.g., related to programmer experience, motivation, languages, tools, etc. Such variation affects most studies to some degree [19, 74, 137], but more particularly those involving human subjects [188]. Contextual variation inhibits not only the reproduction of results within individual replications, but also the synthesis of results across replications [196].

Despite the problem of contextual variability, Dieste et al. [56] argue that meta-analysis can still be effective given sufficiently large datasets. Accordingly, they propose that publishers should accept small-scale replications, the rationale being that such replications are easier to perform, thus leading to more published studies and larger aggregate datasets.

Another option, proposed by Miller [147] (as well as others [130]), is to transform a heterogeneous dataset into a group of homogeneous datasets by accounting for moderators (a type of context variable [13, 183]). This latter option, however, requires quantitative methods for identifying and evaluating context variables.

Context-Sensitive Generalization

Ultimately, as Lindsay and Ehrenberg argue [135], meta-analysis is no silver bullet. It only works when variance is sufficiently small relative to the size of the dataset [56]. The problem for software engineering is that experimental outcomes notoriously vary from context to context, and as da Silva et al. have shown [49], we cannot expect a dramatic increase in the number of replications for specific studies any time soon. Further, for most research questions, it is not yet clear how large the dataset would need to be in order to obtain stable results since the extent of the contextual variance cannot be known without additional data.

In light of such challenges, psychologists Lindsay and Ehrenberg [135] argue in favor of studying context variables in order to synthesize across replications. In their view, replication is precisely a process of context-sensitive generalization. For a replication to be useful, it must demonstrate that a result holds in some other context, even if only slightly different. We must be able to vary a few things (e.g., the time of day, the phase of the moon, or more importantly, the experimenters) and still reproduce the results, or we have not produced usable (i.e., transferable) knowledge. For instance, verification (or the reliability test) is really just a narrow form of generalization. Conversely, a truly identical replication (i.e., one identical in every imaginable detail) would be worthless because it would, without doubt, reproduce the prior results [135, 186]. As Lindsay and Ehrenberg explain, "we need to cash in on such differences in the conditions of observations as do occur. . . rather than to try to sweep them under the carpet" [135, p. 220].

Traditional meta-analysis does not solve the context-sensitive generalization problem. Rather than establishing the generalizability of the results under the various conditions of observation, it simply seeks to reduce sampling errors by pooling data [135]. Moreover, it "does not tell us in what way its scope (that is, its generalizability) has increased with a successful replication, or how it has been circumscribed or negated with an unsuccessful one, and what, in either case, one might therefore want to do next" [135, p. 219]. Thus, studying context variables to facilitate synthesis provides a clear advantage over using meta-analysis alone; the former approach not only enables meta-analysis in cases where it would not otherwise be possible, but also provides groundwork for theory construction [98]. Ultimately, as Lindsay and Ehrenberg point out, replication "is needed not merely to validate one's findings, but more importantly, to establish the increasing range of radically different conditions under which the findings hold, and the predictable exceptions” [135, p. 217].

Methods for Context Analysis

In keeping with Lindsay and Ehrenberg’s perspective, Juristo and Vegas [102, 104] propose a process for identifying context variables in non-exact replications. The process concerns all four phases of a replication: definition and planning, operation and analysis, results interpretation, and contribution evaluation. In a related paper, Juristo et al. [105, 211] outline a method for leveraging communication between researchers to identify context variables. This process—which involves structured meetings, as well as unstructured phone calls and emails—can be nested within and used as part of the broader process mentioned above.

Both processes provide a helpful framework for structuring the investigation of context variables. However, both also suffer from three key deficiencies:

1. They are specified at fairly high levels, thus leaving researchers the burden of figuring out how to apply them in practice.
2. They rely on subjective qualitative methods to pick out candidate variables.
3. They say little about what to do once a candidate variable is identified, other than to conduct additional replications.

Alternatively, context variables can be quantitatively investigated by aggregating raw data from multiple studies into a single statistical analysis, as described previously. However, context data are even less likely than experiment data to be available, consistently measured, and adequately documented (even in the case of close replications; e.g., see [180]; also compare [168], [212]).

A third option, fUSE [42], is built on top of meta-analysis, and so does not require raw data. fUSE adds two tests to meta-analysis, one for assessing whether moderators need to be investigated and one for evaluating the explanatory potential of candidate moderators.

fUSE is modestly robust to both missing data and to the problem of metric standardization. However, being based on meta-analysis, fUSE inherits several weaknesses:

1. It requires independence between studies for the results to be accurate.
2. It does not specify a process for discovering moderators.
3. It can only be used to test a moderator for which each individual study represents one cohesive level or subgroup of that moderator.
4. Since it uses experimental runs as the unit of measure, instead of individual observations, it requires a large number of replications.

The last limitation is especially problematic for the analysis of moderators because such analysis requires subdividing the data (e.g., see [42], in which a set of 21 experimental runs was still insufficient to achieve satisfactory results).

The State of Results Synthesis

In summary, current replication frameworks are abstract and do not define concrete methods for results synthesis. Systematic literature reviews are by definition intended to handle results synthesis, but thus far deal more with the collection of relevant articles than with synthesis itself. Synthesis methods from other fields are mostly qualitative and/or involve subjective components, with the exception of two general options: 1) meta-analysis, and 2) raw-data aggregation.

The primary problem with meta-analysis (at least for software engineering) is that its results are unstable in highly variable contexts. fUSE represents a possible solution to that problem, but being based on meta-analysis, requires a large number of replications to achieve satisfactory results. Thus, in software engineering, meta-analysis is confronted by a major roadblock—the cost to obtain a dataset large enough to address most questions of interest is impractically high. Unfortunately, raw-data aggregation is also inhibited by problems of data availability—though in that case, the problems are due to the difficulty of curating and reconciling data across past experiments.

1.3.4 Methods for Inter-Study Communication

In the biological sciences, Bissell [24] argues that failures in the transfer of technical/tacit knowledge between researchers is a primary cause for failed replications. So too, in software engineering, communication (or knowledge-sharing) issues are one of the most frequently cited explanations for replication failure [40, 105, 189, 191]. Most noticeably, communication issues affect the fidelity with which experimental conditions are reproduced (e.g., getting the study design and measurements right). However, communication issues also impact the discovery/awareness of context variables [105, 211], and hence affect the quality of the knowledge resulting from a replication. After all, the same dataset can be interpreted in a variety of ways depending on the perceived context of its derivation.

To combat problems of communication, researchers have made efforts on two broad fronts:

1. *Archival communication*—improving the communication of original and replication studies for use in future replications.
2. *Real-time communication*—improving/structuring communication between the replicating researchers and prior investigators at the time of a replication.

Concerning the first category, in 2005 Jedlitschka and Pfahl [96] published a set of reporting guidelines for controlled experiments in software engineering. The guidelines are currently the accepted standard for original experiments and serve as a checklist of details that typically need to be included in order to facilitate future interpretation and replication of an experiment. In 2008, Kitchenham et al. [109] evaluated the 2005 guidelines and identified 44 areas for amendment and 8 defects. More recently, in 2010, Carver [40] argued that replication studies involve additional concerns beyond those of original studies, and so proposed an additional set of guidelines for reporting replications. Ultimately, research reporting is a hard problem, primarily because researchers cannot know up front which of the many context variables will ultimately prove important in future studies.

Another area of work to improve knowledge sharing is the development of lab packages. Lab packages are used to document experiment details that do not fit in a published report. Lab packages can contain protocol documents, datasets, data documentation, additional analyses, appendices, etc. In 2002, Shull et al. [189, 191] explored methods for eliciting and better communicating tacit knowledge via a variety of mechanisms, including lab packages. The methods were primarily qualitative. In 2006, Solari and Vegas [199] developed a framework for improving the evaluation and use of lab packages in replications. More recently, in 2013, Solari [198] studied critical incidents in replications with the long-term goal to develop improved lab package guidelines.

Less work has been done in the area of *real-time communication*. Vegas et al. [211] tested three researcher communication models and identified one, involving both structured and unstructured communication, as being the most effective for achieving a successful replication. The winning model has also been shown to help in identifying potentially relevant context variables [105].

Finally, related to both types of communication (*archival* and *real-time*), Mendonça et al. [145] developed a framework to guide communication within and across studies. Although the authors provide several concrete suggestions in discussing their framework, the framework itself is mostly conceptual.

1.3.5 Summary of Open Problems

Based on the literature discussed above, we find four open problems related to replication:

1. *The rate and explicitness of replication.* Few replications are conducted each year. Although researchers may be publishing some replications in disguise, most software engineering claims are still based on only a handful of largely disconnected studies.

Thus, we not only need more replications, but we need to make the replication process more explicit and systematic.

2. *The theoretical foundations of replication.* We have not yet established a precise definition for replication, nor an authoritative taxonomy of replication types, nor a comprehensive framework to show how the various types can be used to build knowledge. We have expressed many ideas about these topics in the literature, but the discussion is complex. A number of fundamental questions remain unanswered, including: To be more systematic about replication, should we authoritatively establish replication definitions, terminology, and taxonomy? Or, will doing so ultimately cripple the effectiveness of replication by limiting its methodological development? Alternatively, can we devise a sufficiently flexible framework to allow the reconciliation of current and future replication ideas? To begin answering such questions, we need to better understand the philosophical relationship between replication and knowledge production.

3. *The tractability of methods for context analysis.* Due to the highly variable nature of software contexts, producing usable knowledge requires developing context-sensitive generalizations. Thus, all methods for synthesizing results across replications (including both meta-analysis and any form of raw-data aggregation) must provide mechanisms for the identification, evaluation, and theoretical integration of context variables. Unfortunately, all current methods for context analysis are subject to critical limitations. In the case of meta-analysis, an impractically large number of studies is needed in order to confidently identify context variables. Conversely, raw-data aggregation is inhibited by problems of curating and reconciling data across past studies. Thus, we need a tractable method for context analysis—ideally a method for obtaining sufficient, current, and clean *raw* data from the right selection of participants, so as to minimize the number of samples/studies necessary to produce generalized conclusions.

4. *The effectiveness of inter-study communication.* A significant difficulty common to all methods of inter-study communication is that researchers cannot know at the time of

the communication which of the many context variables are most salient (i.e., which will ultimately prove necessary for synthesizing results across studies). Thus, we need data-driven methods to identify likely-relevant context variables earlier in the research process, so that researchers can be more effective in their communications (especially with respect to the data preserved in reports and lab packages).

Note that problem 2 primarily concerns the role of replication, whereas problems 1, 3, and 4 represent practical issues. In this dissertation, we address both types of concerns; we do so via two research components, theoretical and practical.

1.4 Thesis Statement

Synthesizing and integrating *external theory* with the software engineering literature clarifies the role of replication and informs a broad range of longstanding, practical replication concerns. Furthermore, *post-hoc moderator analysis*—based on *Bayesian models* and executed within the context of *joint replication*—enables the quantitative evaluation of context variables in greater detail, with greater statistical power, and via considerably smaller datasets than previously possible; thereby, the method enables reconciliation and generalization of otherwise contradictory results across replications—which problem has previously prevented the production of transferable knowledge in software engineering.

1.5 Project Description

Discussions of replication in the literature have primarily addressed practical questions (e.g., the development of lab package [189, 191]). Certainly, answering practical questions is necessary in order to make progress. However, the foundational nature of the concerns identified above suggests the need for a deeper, more systematic interrogation of our empirical assumptions. Therefore, as part of this dissertation, we explore and synthesize external replication theory; as we show, integrating such theory with the software engineering literature generates novel insights pertaining to many of the current, and especially longstanding, problems of replication.

Theory alone, however, is inadequate; ultimately, we need concrete, actionable results. Thus, a good solution should provide not only a theoretical component, but also a practical component. Accordingly, we address replication problems via a two-part research strategy, consisting of: 1) the theoretical work mentioned above, and 2) a series of practical replication studies. As we show, these two components, which are complementary, enable broader coverage of the problem space than either could achieve alone. Also, inasmuch as they rely on different research methods, the two components serve as validation for one another (as per Daly's multi-method approach [51]).

Of the four open problems identified in Section 1.3.5, the theoretical component primarily targets problem 2 (the theoretical foundations of replication), and the practical component primarily targets problem 3 (the tractability of methods for context analysis). However, solutions to problem 3 also address problem 4 (the effectiveness of inter-study communication). Additionally, solutions to problems 2 and 3, by clarifying the role and improving the success of replication, help address problem 1 (the rate and explicitness of replication).

1.5.1 Theoretical (Outer Study) Component

The purpose of the theoretical component is to *explore* and *synthesize* theory on replication from other fields, as well as to *integrate* that theory with the software engineering literature. Complete integration involves four tasks:

1. Distill a simplified explanation of the theory.
2. Identify positive and negative examples of the theory in the software engineering literature.
3. Formulate software-engineering-specific guidelines based on the theory.

4. Show how the theory relates to existing replication ideas.

To accomplish these objectives, we conduct a two-stage analysis, in which stage 1 occurs before the practical studies component and stage 2 occurs after. The benefit of the two-stage approach is that the theoretical (or outer study) component is able to guide and validate the practical (or inner study) component, and vice versa. In stage 1, we tackle all three objectives (*exploration*, *synthesis*, and *integration*). However, for the *integration* objective, we do not develop actionable guidelines yet; rather, we only identify elements of the theory which are corroborated in the software engineering literature. In stage 2, we refine the theory from stage 1 based on our experiences conducting the practical studies and complete the *integration* objective.

To accomplish the first two objectives of the theoretical component (*exploration* and *synthesis*), we use qualitative methods. We choose qualitative methods primarily because quantitative methods do not fit the conceptual nature of the problems we address; additionally, the problems are complex in, as yet, undefined ways, and quantitative frameworks are ill-suited to capturing amorphous complexity [47, 62]. A number of potentially applicable qualitative methods are available, including narrative synthesis, cross-case analysis, thematic analysis, and grounded theory [47, 48]. Of these, we select *grounded theory* because:

1. It is designed for generating a cohesive theory from complex, disparate data [47].
2. It is ideal for answering open-ended questions, such as “What is going on here?” [1, p. 491].
3. It lets the data drive the theoretical output of the research, as opposed to using theory to drive the interpretation of data (which makes it an excellent method for generating new perspectives on old problems [1]).
4. It can be applied to a study of articles and ideas (as opposed to people and cultures) with only minor adaptations.

Three variants of grounded theory are standard in the general scientific literature: Glaser's approach [82], Strauss and Corbin's approach [45], and Charmaz's approach [41]. We follow a process most similar to Strauss and Corbin, involving coding, memoing, the forming of categories, etc. However, we also accept Glaser's admittance of all forms of data (including quantitative data if such should become relevant to the investigation). Additionally, we incorporate two elements of Charmaz's approach. First, since replication is tightly coupled with individual and social learning processes, and since value judgments are inescapably connected with such processes [55, 80, 187], we recognize that constructivism necessarily plays a role in our theoretical analysis at some level. Accordingly, we assert that the resulting theory is not the final, nor the only "correct" or appropriate perspective, but merely a lens through which to view a set of problems. Second, we accept Charmaz's preference for relaxing the traditional formulaic process of grounded theory to suit the needs of the particular application. In our case, adjustments are necessary because we are applying grounded theory, which was originally developed to study people via interviews and observation [45, 82], to a study of concepts and ideas spread across an effectively limitless collection of documents.

The steps in our process are as follows:

1. Identify data sources.
2. Extract and catalog concepts from the data sources (*open coding*).
3. Consolidate the extracted concepts into general categories (*axial coding*).
4. Search for a common theme (*constant comparison*).
5. Arrange the data and categories under the common theme to form a theory.
6. Refine the categories and theory, adjusting them until all relevant concepts are accommodated (*selective coding*).
7. Repeat steps 1–6 until conceptual saturation is achieved (*theoretical sampling*)—i.e., no new relevant concepts emerge; also, the data are sufficient to construct a coherent explanatory story.

8. Consolidate the categories and simplify theory as much as possible without losing key concepts.

Note that in such investigations, it is important not to force convergence on the data too early in the process, but rather, to let the categories emerge from the data as concepts overlap to reveal unifying themes. We intend the output of the process to be an overarching theory, sufficiently general to inform a broad range of replication concerns in software engineering—i.e., a paradigmatic theory, or theoretical lens.

Since the domain of non-software-engineering fields is vast, we do not attempt to perform an all-encompassing systematic literature review. Instead, we use a semi-snowball [87, 133, 153] approach for identifying candidate data sources. To form the kernel of the snowball we begin with the social sciences, particularly sociology. Sociology is a good place to start because many of the replication problems we encounter in software engineering appear related to human factors. To identify initial data sources in sociology, we consult online search engines, as well as sociology professors and researchers. We also sample classical theorists, including Durkheim, Machiavelli, Marx, Nietzsche, Simmel, Tocqueville, and Weber [21, 43, 65, 138, 157, 162, 163, 193, 220].

By *snowball*, we mean that we follow references within identified sources in order to locate additional relevant literature. By *semi*-snowball, we mean that we continually seek to project our search into disparate regions of the scientific literature—e.g., by searching Internet databases, obtaining references from other researchers, following hunches, etc. The idea of the process is not to be rigorously systematic in our coverage of the literature, which would limit us to one or two fields; rather, we seek to incorporate as many perspectives as possible, spanning the breadth of the literature. We prefer the latter approach in order to maximize creativity and insight in the resulting analysis—which creativity is particularly needed given the longstanding nature of the problems we address. Based on this search process, the final analysis incorporates articles from a wide array of fields, including: sociology, psychology, linguistics, architecture, philosophy, physics, and biology.

Note that semi-snowball sampling is often used when seeking to obtain data from a broad range of observational units, but where statistically random sampling is not possible (e.g., [87, 133, 153]). The *semi* aspect of semi-snowball sampling also typically involves some type of opportunistic process, which is justified by the fact that no other sampling approaches are possible (or at least feasible).

Concerning this dissertation, an opportunistic search is reasonable because textbooks are simply not being written on the theory of replication, in any field, even in "hard" sciences such as physics [186] (or if they are, they are too rare and too far between to be efficiently discovered via systematic search); rather, discussions of replication are spread out across the literature in piecemeal fashion. Opportunistic sampling does not damage the theoretical results because our purpose is not to provide a systematic report on what other fields think about replication; rather, our purpose is to generate new insights pertaining to longstanding replication problems. Inasmuch as we accomplish the latter objective, the theoretical component can be said to be valid and successful (see Section 1.6 for further discussion of validity criteria).

When conducting grounded theory, it is often helpful to begin with a few broad questions [45]. However, the goal of the study per se is not to answer the questions; rather, the questions are merely a starting point for thinking about the data. In searching, selecting, and analyzing replication theory, we use the following questions as an initial guide:

- What is the *idealized* role of replication in science?
- What is the *practical* role of replication in science?
- How is replication related to truth and knowledge?
- What is the value of replication without reproduction of results?
- How does replication relate to various types of research, such as experimentation, case studies, systematic literature reviews, etc.?
- How far can we push the boundaries of replication and still call it replication?

1.5.2 Practical (Inner Study) Component

The purpose of the practical component is two-fold: 1) to provide data for further developing and validating the theoretical component; and 2) to make a substantial contribution in terms of methods for addressing problems 3 and 4 listed in Section 1.3.5—i.e., methods for context analysis. To address these objectives we incorporate both *breadth* and *depth* elements, respectively, into the practical studies. For breadth, we conduct a series of small-scale studies, covering a range of replication concepts and empirical methods. For depth, we focus on a specific replication study, in which we develop/test a new method for identifying and evaluating context variables.

For the small-scale studies we tackle two research topics: *Conway's Law* and *design patterns*. We select Conway's Law because it represents a classic example of a problem that commonly occurs in software engineering—that of widely-accepted but poorly explicated pseudo results.[4] In our study of Conway's Law, we evaluate the extent to which it has been empirically substantiated, as well as the extent to which variations in the law have been catalogued and explained. Additionally, via differentiated replications, we gauge the general applicability of replication for identifying meaningful variations in such phenomena. In particular, Conway's Law provides an ideal opportunity to test the value of *highly differentiated replication*—i.e., replications in which the only element maintained across studies is the underlying theory. As we show, the *phenomenon* of Conway's Law is rich in diversity, such that, without replication and subsequent contextualization, it cannot be trusted to apply in any specific context—that is, in most cases, broad abstractions of the law are likely to be inadequate.

We select design patterns because many studies (including replications) have investigated patterns, and to date their results are largely irreconcilable [6, 222] (likely due to the

[4] For instance, Endres and Rombach list nearly 50 "laws" in their *Handbook of Software and Systems Engineering* [67], almost all of which are based on only a handful of empirical studies. The authors even comment in the introduction that the term *rule* would probably be more appropriate; however, to keep with convention they use the term *law* instead.

influence of context variables). We replicate a 2001 study by Prechelt et al. [168] testing the impact of design patterns on software maintenance (abbreviated as "PatMain"). PatMain is one of the earliest studies on patterns and has been replicated once previously [212] with divergent results. We conduct our small-scale replication of PatMain as a strict or close replication using computer science students.

For the depth element of the practical component we conduct an extended analysis of the PatMain series of studies, the purpose of which is to reconcile divergences in the results across the three iterations of the study (the original, the first replication, and our study). The process involves four steps:

1. Assess the heterogeneity of the results.
2. Investigate context variables which may be inhibiting generalizability.
3. Address the original hypotheses.
4. Formulate generalized conclusions.

Via this process we test and refine a new method for context analysis, involving joint replication, post-hoc moderator analysis, and frequentist/Bayesian statistical models (see below for a description of each element). To conclude the practical component, we conduct a postmortem of the extended analysis, including an evaluation of our method for context analysis.

Joint replication. The idea of a joint replication is to organize a multi-site study, performed by separate research teams whose efforts are coordinated, yet the researchers at each site act independently in performing their own replication. The concept is most similar to multi-site randomized controlled trials in social work research [200]. Joint replication has not previously been done before in software engineering, but as we show, it enables evaluation of context variables in greater detail than historically possible. In particular, if the experiment protocol is tightly controlled across sites, the method can facilitate identification of industry-relevant context variables.

Post-hoc moderator analysis. Post-hoc analysis is commonly used in other fields (e.g., medicine [209]) for uncovering relationships (e.g., context variables) that would otherwise remain undetected given only a priori methods. In statistical terms, post-hoc analysis increases the likelihood of type 1 errors. Thus, the output of the process is not a set of final conclusions, but a list of likely-meaningful variables for use in the next round of testing. The concept of moderators comes from sociology and psychology [13, 183]. A moderator is any explanatory variable that interacts with another explanatory variable in predicting a response variable [183]—i.e., moderators are a type of context variable. In our analysis, we explore the concept of moderators to see if it can be used to encapsulate and formalize contextual information, sufficient to standardize such information as part of the experimental framework.

Frequentist/Bayesian statistical models. Frequentist models are commonly used in software engineering to evaluate experiment data. We are interested to see whether frequentist models (in particular, mixed models [174]) are sufficiently powerful to be used for the post-hoc identification and evaluation of context variables. We also test Bayesian models. In software engineering, Bayesian models are not commonly used to analyze experiment data. However, they are useful in cases where statistical power is limited due to small sample sizes and high variance [54]—which problems are not only key barriers to context analysis, but also common occurrences in software engineering. In particular, we test a specific type of additive-effects model described by Felt [69], which avoids linearity assumptions.

1.6 Validation

In this section, we describe the methods and criteria by which we validate the outcomes of this dissertation. For each component (theoretical and practical), we describe the validation process in terms of 1) a *mechanism* for promoting/ensuring validity during the course of the research, 2) a *test* for assessing final validity at the conclusion of the research, and 3) a *gold standard* against which to compare results in the final test of validity.

1.6.1 Theoretical Component

In keeping with our use of qualitative methods, we assess the theoretical component's validity based on accepted practices for qualitative research.

Mechanism

Triangulation of evidence is a standard technique used in qualitative research to promote validity [47]. Triangulation makes two requirements of the analysis—first, that it include multiple disparate data sources, and second, that it reconcile all relevant information from those data sources under the framework of the results. If both of these criteria are met, then the results should be realistic and applicable.

To facilitate triangulation, we use a method (grounded theory) that relies on the principle of *constant comparison* [45, 82, 188], which principle we follow rigorously. We also incorporate several sources of diversity into our data: 1) we include both theoretical and practical data sources; 2) we perform a cross-disciplinary literature search; and 3) via the breadth element of the practical component, we incorporate a range of replication concepts and empirical methods. Additionally, we implement a two-stage analysis—i.e., develop theory, perform practical studies, readdress theory—thus allowing the practical studies to inform and refine the theoretical analysis.

Test

To assess final validity, we compare the resultant theory against the experiences/outcomes of the practical studies, as well as against the existing software engineering literature.

Gold Standard

We consider success to have been achieved inasmuch as the following two criteria are met:

- The resultant theory clearly defines the role of replication.

- The resultant theory yields actionable insights relevant to the problems of replication and knowledge production in software engineering.

1.6.2 Practical Component

For the practical component, we divide each of the three categories (*mechanism*, *test*, and *gold standard*) into two subcategories, representing the *breadth* and *depth* elements of the component.

Mechanism

1. *Breadth element* (small-scale studies). We follow accepted practices for conducting empirical studies, including the guidelines outlined by Kitchenham et al. [113], Jedlitschka and Pfahl [96], and Carver [40]. Additionally, we publish a report for each study, thereby taking advantage of the peer review process to help ensure validity.
2. *Depth element* (test of a new method for context analysis). The best way to ensure the validity of a method is to apply the method to a real-world problem. We apply our method for context analysis to the PatMain series of studies.

Test

1. *Breadth element.* To assess final validity, we follow a standard validity framework [44, 217] consisting of four parts: construct, conclusion, internal, and external validity. Based on this framework, we catalogue threats to validity for each study and qualify the results in reference to those threats [113].
2. *Depth element.* To assess final validity, we examine (in reference to the PatMain series of studies) the degree to which the resultant method achieves the gold standard described below.

Gold Standard

1. *Breadth element.* We consider success to have been achieved inasmuch as the individual studies make a useful contribution to our understanding of replication. (In selecting this particular criterion, we recognize that all empirical studies have flaws, and it is only in the intersection of a set of variously flawed studies that we can obtain a more complete understanding—i.e., the principle of parsimony [19].)

2. *Depth element.* We consider success to have been achieved inasmuch as the resultant method achieves the following four objectives:

   - The method allows for the identification of context variables earlier in the research process than currently possible (so as to facilitate improved research reporting).

   - The method allows for the evaluation of context variables via fewer samples/studies than current methods.

   - The method enables the reconciliation of otherwise disparate results across a series of replicated experiments.

   - The method facilitates practically-useful conclusions that generalize to a broad set of contexts.

1.7 Dissertation Format (Series of Papers)

This dissertation is written as a collection of papers, where each chapter of the dissertation (with the exception of Chapter 1) represents a single, self-contained paper. Each paper has either been published or is at some stage in the publication process (see Table 1.1). Note that the dissertation as a whole necessarily contains some redundancies, since each paper must provide its own introduction and background discussion.

Additionally, note in Table 1.1 that the first eight papers are not included as chapters in the dissertation, but are published externally only. These "External Papers" represent work completed as part of the dissertation, and each addresses specific aspects of the project,

Table 1.1: Publications resulting from this dissertation.

External Paper 1	Charles D. Knutson, Jonathan L. Krein, Lutz Prechelt, and Natalia Juristo. 1st International Workshop on Replication in Empirical Software Engineering Research (RESER). In *International Conference on Software Engineering*, pages 461–462, 2010. [115]
External Paper 2	Charles D. Knutson, Jonathan L. Krein, Lutz Prechelt, and Natalia Juristo. Report from the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER 2010). *SIGSOFT Software Engineering Notes*, 35(5):42–44, 2010. [116]
External Paper 3	Jonathan L. Krein, Charles D. Knutson, Lutz Prechelt, and Natalia Juristo. Report from the 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER 2011). *SIGSOFT Software Engineering Notes*, 37(1):27–30, 2012. [124]
External Paper 4	Jonathan L. Krein, Charles D. Knutson, Lutz Prechelt, and Christian Bird. Message from the RESER 2013 workshop chairs. In *International Symposium on Empirical Software Engineering and Measurement*, page 395, 2013. [123]
External Paper 5	Jonathan L. Krein, Charles D. Knutson, and Christian Bird. Report from the 3rd International Workshop on Replication in Empirical Software Engineering Research (RESER 2013). *SIGSOFT Software Engineering Notes*, 39(1):31–35, 2014. [122]
External Paper 6	Sabrina E. Bailey, Sneha S. Godbole, Charles D. Knutson, and Jonathan L. Krein. A decade of Conway's Law: A literature review from 2003–2012. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–14, 2013. [12]
External Paper 7	Scott H. Burton, Paul M. Bodily, Richard G. Morris, Charles D. Knutson, and Jonathan L. Krein. Design team perception of development team composition: Implications for Conway's Law. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 52–60, 2011. [36]
External Paper 8	Kyle L. Blatter, T.J. Gedhill, Jonathan L. Krein, and Charles D. Knutson. Impact of communication structure on system design: Towards a controlled test of Conway's Law. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 25–33, 2013. [25]
Chapter 2	Jonathan L. Krein, Landon J. Pratt, Alan B. Swenson, Alexander C. MacLean, Charles D. Knutson, and Dennis L. Eggett. Design patterns in software maintenance: An experiment replication at Brigham Young University. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 25–34, 2011. [126]
Chapter 3	Jonathan L. Krein, Lutz Prechelt, Natalia Juristo, Aziz Nanthaamornphong, Jeffrey C. Carver, Sira Vegas, Charles D. Knutson, Kevin D. Seppi, and Dennis L. Eggett. A multi-site joint replication of a design patterns experiment using moderator variables to generalize across contexts. 2014, under review. [127]
Chapter 4	Jonathan L. Krein, Lutz Prechelt, Natalia Juristo, Kevin D. Seppi, Aziz Nanthaamornphong, Jeffrey C. Carver, Sira Vegas, and Charles D. Knutson. A method for generalizing across contexts in software engineering experiments. 2014, submission pending. [128]
Chapter 5	Jonathan L. Krein and Charles D. Knutson. The humanity of science versus the complexity of reality: A theoretical study of replication and knowledge production. 2014, submission pending. [121]

as outlined above (e.g., External Papers 6–8 represent the small-scale studies of Conway's Law). However, as the dissertation has evolved, the external papers have became less central, and thus we exclude them from the dissertation document in preference for other material.

For the interested reader, the external papers are all available online. Additionally, we summarize the contributions of each chapter/paper in Section 1.8, including the contributions of the external papers (in Sections 1.8.1 and 1.8.2). Lastly, note that the summaries we provide for the external papers are comprehensive with respect to the topic of replication. Thus, unless the reader is interested in non-replication topics (e.g., Conway's Law), the summaries should be sufficient to understand the contributions of the papers with respect to this dissertation.

1.8 Contributions

In this section, we summarize the role and contributions of each chapter/paper. We arrange the discussion into subsections, representing the key components of the dissertation. Note that the first two subsections describe "External Papers" (i.e., papers published exclusively outside of the dissertation). As described in Section 1.7, these papers represent work that was completed as part of the dissertation, but which ultimately became less central to the final results. For further explanation of these papers and why they are summarized here, see Section 1.7.

1.8.1 Exploratory/Impact Component (RESER Workshop)

External Papers 1–5 are reports describing the various iterations of the *International Workshop on Replication in Empirical Software Engineering Research* (or RESER workshop), which was held in 2010, 2011, and 2013. At the inception of this dissertation (in 2009), we founded the RESER workshop as a way to explore and, hopefully, positively impact replication concerns. The external (or publicly-advertised) reasons for organizing the workshop included: 1) to raise the quality and amount of replication work performed in software engineering; 2) to collect and package advice, tools, and experience regarding replication; and 3) to provide a venue for publishing small-scale (or so-called "useless" [56]) replications, as well as a venue for publishing preliminary studies testing new replication ideas.

Internally (i.e., relative to the goals of this dissertation), our primary reason for founding the workshop was to help us develop research questions, understand and articulate replication problems, and explore a breadth of solution ideas. In particular, via the workshop we learned how other researchers perceive and experience replication, which exposure enabled us to more clearly characterize replication issues. Additionally, the workshop helped us to build a network of collaborations around specific problems, without which the joint replication (i.e., Chapters 3 and 4 of this dissertation) would not have been possible. Finally, the workshop allowed us to realize greater impact from our research—specifically, by raising awareness about the importance of replication, by engaging more researchers in addressing replication problems, and by positively impacting the rate of replication.

Concerning the rate of replication, de Magalhães and da Silva [52] report finding 44 new replications published in 2011 and 2012. Thus, the rate of replication has increased from an average of 13 per year (from 2004–2010) [49] to an average of 22 per year (2011–2012). Of this, de Magalhães and da Silva state, "The preliminary results of this update showed that there was a substantial increase in the numbers of the replications *mainly due to the RESER's effect*" [52, p. 52, emphasis added]. Moreover, notice (in Table 1.2) that the total number of papers reporting a replication published in RESER for the years 2011 and 2012 is only 6 (since the workshop was not held in 2012). Thus, RESER papers account for only 1/3 of the increase reported by de Magalhães and da Silva; or in other words, the replication increase is not simply an artifact of establishing an new venue, but represents a broader cultural shift among researchers and reviewers, presumably due (at least in part) to the influence of the RESER workshop.

External Paper 1 is a pre-workshop summary describing the purpose and topics of the 2010 workshop; the summary was written for the proceedings of the *International Conference*

Table 1.2: RESER workshop stats.

Year	Submitted Papers	Authors of Submitted	Accepted Papers	Authors of Accepted	Papers Reporting a Replication	Methodology & Other Papers	Registrants	Attendees
2010	12	32	10	28	1	9	17	20
2011	9	34	9	34	6	3	21	21
2013	12	42	11	38	7	4	21	24

on Software Engineering (ICSE), with which RESER co-located in 2010. External Paper 2 is a post-workshop summary describing the keynote, papers, discussion, and major takeaways of RESER 2010. Relative to this dissertation, the 2010 workshop primarily served as a resource for initial topic exploration. Lessons learned include:

- Despite its straightforward appearance, determining confirmation (i.e., whether a replication reproduced a prior study's results) is complex. In practice, determining confirmation is more of a synthesis and judgment task, than it is a declaration task—that is, confirmation is not something to simply be read off, as a number from a table. For example, in describing his experience conducting a replication, Jim Herbsleb (the keynote speaker) enumerated the similarities and differences between the results of his replication and those of the original study. As he concluded, it was clear that no one in the room was sure how to call the outcome; some parts seemed (or "felt") confirmed, others were close, and some parts appeared to contradict.
- Replications can be more difficult to publish than original studies. For example, referring back to Jim Herbsleb's replication, clearly the conflicted results are important for future readers of the original study to be aware of; nevertheless, the replication was rejected as unpublishable.
- More knowledge can often be gained from contradictory, or even inconclusive results, than from clean confirmation. For example, the conflicted results of Jim Herbsleb's replication necessitated further study, prompted new questions, and ultimately led to deeper insights.

- The problem of replication is not isolated to any particular context or easily solved under any specific circumstances. For example, even in the mining software repositories community, where results are generally thought to be more objective than in human subjects research, replication is both needed and fraught with strikingly similar challenges.
- Some (or possibly many) replications are being published, but not labeled as such, either because the authors fear their work will be perceived as less valuable, or because the studies do not fit traditional notions of replication (e.g., case studies).
- As a community, our understanding of the role of replication in the knowledge-discovery process is fragmented. Most of our notions of replication come form other disciplines and are not well adapted to the context of software engineering. Furthermore, many of our definitions of replication are either too vague and imprecise, or too narrowly focused. For example, the workshop was fraught with misunderstandings due to variations in individual definitions of replication.
- As with terminology, so too with methodology, researchers disagree on key issues concerning the conduct and reporting of replications. For example, at the workshop, a somewhat heated discussion ensued about reporting standards, with everyone recognizing a need for improved reporting, but disagreeing on how that improvement should be effected.

In 2011, RESER co-located with the *International Symposium on Empirical Software Engineering and Measurement* (ESEM). External Paper 3 is a post-workshop summary describing the keynote, papers, discussion, and major takeaways of RESER 2011. Relative to this dissertation, the 2011 workshop primarily served as a platform for testing a new replication method, strict joint replication. Lessons learned include:

- We cannot effectively define the concept of replication (at least if we are to operationalize it) in isolation from the type of knowledge we intend to build. Accordingly, an individual replication is fairly meaningless if not constructed within the broader context of a research strategy. Further, the notion of an individual replication is far less meaningful or analytically powerful (with respect to building knowledge) than that of replication as an extended process.

- Joint replication has the potential to reproduce the broader problem of replicability, but within the context of a single controlled experiment. For example, the participants in the 2011 joint replication appeared demographically similar, yet they performed considerably different across sites (which is consistent with what we typically observe across a set of replications).
- The study of context variables is paramount to building knowledge from replication. For example, the performance variations in the joint replication were significantly greater across sites than within. As such, the variations were likely not due to random noise, but to key moderating factors which differed across the four sites. Without an understanding of such factors, results cannot be synthesized across sites/studies, and therefore, we cannot produce usable (i.e., transferable) knowledge. This conclusion was evident in more than one study presented at the workshop.
- We need better (in particular, tractable) methods for context analysis. For example, the experiment framework of the joint replication collected data on nearly as many explanatory variables as we had observations for response variables—i.e., too many to effectively estimate coefficients for all of them in the same statistical model. It was not clear how to determine which explanatory variables should be modeled, as opposed to discarded. Moreover, the actual explanatory variables responsible for the divergent results may not have even been measured. Thus, numerous possible unmeasured variables also needed to be considered.
- The rate of replication is inadequately low in software engineering. We had a sense of this prior to the 2011 workshop (based on cultural perception, as well as on two prior,

but less comprehensive and potentially outdated, surveys [4, 197]). However, the da Silva et al. survey (presented at RESER 2011 [50], and later published in the EMSE journal [49]) provided much needed confirmation.[5]

- As a research community, we need to strike a balance between standardization and creativity—meaning, if the boundaries we draw around replication are too rigid, we may close ourselves off to the better part of knowledge that could be built via replication. For example, at RESER 2011, we saw repeated (from 2010) the problem of diverse and inconsistent terminology. However, this time around, we came to a somewhat different conclusion. As the attendees debated the relative merits of strict versus differentiated replication (in connection with a proposal to perform a differentiated joint replication for RESER 2013), we recognized that the disparate state of terminology is not entirely without benefit. On the one hand, maintaining disparate terminology degrades communication and collaboration between researchers; on the other hand, it fosters diversity and creativity in the scientific process. Ultimately, terminology, as a basic building block of science, influences the researcher's awareness of various learning approaches, as well as the knowledge potential of those approaches.

External Paper 4 is a pre-workshop summary describing the purpose and topics of the 2013 workshop; the summary was written for the proceedings of ESEM, with which RESER co-located in 2013. External Paper 5 is a post-workshop summary describing the keynote, papers, discussion, and major takeaways of RESER 2013. Relative to this dissertation, the 2013 workshop primarily served as a platform for experimenting with differentiated joint replication. Lessons learned include:

- Highly differentiated replications (i.e., replications in which the only thing maintained between the original study and the replication is the underlying theory) are helpful

[5] Incidentally, the da Silva et al. survey is another example of the impact that RESER has had on the software engineering research community. The survey was carried out in direct response to the 2010 workshop, which the lead author, Fabio da Silva, attended. The survey is worth special mention because of the foundational role it played in the development of this dissertation, for which contribution I am indebted to Fabio and his colleagues.

for conceptually exploring the domain of a phenomenon. As such, they appear to be most useful for working out theoretical concerns in an emerging research space. However, the process is also useful for identifying likely meaningful context variables, and thus can be valuable even in the case of mature research topics. In general, highly differentiated replication (especially when executed as a joint replication) can be an effective mechanism for generating new ideas and getting unstuck (as evidenced by the 2013 joint replication, a differentiated joint replication of Conway's Law).

- Even replications with a sample size of one can be valuable—in particular because they allow the researcher to explore the relationship between a phenomenon and its context in greater detail, and thus to uncover important context variables which may otherwise be missed. This conclusion is based primarily on Andrew Brooks's experience conducting a replication with only one developer, which he presented at the 2013 workshop [30]. However, the conclusion also echoes two papers from RESER 2010, one concerning the unrecognized value of case study replications [142], and another concerning the value of small-scale replications [56].

1.8.2 Practical Component—Conway's Law

External Papers 6–8 are small-scale studies investigating Conway's Law. For RESER 2013, we conducted a differentiated joint replication of Conway's Law, to which External Papers 6 and 8 were contributions. External Paper 7, which was presented at RESER 2011, was a precursor to (and motivator of) the Conway joint replication.

As previously discussed (in Section 1.5.2), Conway's Law is a classic example of a problem that commonly occurs in software engineering—that of widely-accepted but poorly explicated pseudo results. Relative to this dissertation, we had two primary purposes in conducting the Conway studies. First, we wanted to test a general hypothesis that many classic software engineering laws are, in fact, poorly understood—and thus in need of replication—despite being almost universally accepted. Second, we wanted to explore the value of highly differentiated replication (i.e., replications in which the only element maintained across studies is the underlying theory). Concerning this latter purpose, Kitchenham [108] argues strongly in favor of independent replications (i.e., replications that duplicate the purpose of a reference study, but via new researchers, procedures, and materials), of which highly differentiated replication is essentially an extreme example. Thus, in executing the Conway studies we aimed to test both the general importance of replication (for a particular class of research results, that of software engineering "laws"), as well as the utility of replication at the extreme end of the differentiation scale.

External Paper 6 is a systematic review of papers and books that cite Conway's Law spanning the years 2003–2012. The purpose of the survey was to determine how well Conway's Law is substantiated. For instance, do recent papers citing Conway's Law investigate the law directly, in an effort to substantiate it, or do they simply apply the law in pursuit of some other question? And in the latter case, do the citing studies reference a body of substantiating literature, or do they reference Conway's original paper, thus taking substantiation for granted? Furthermore, do papers citing Conway's Law share a common definition of the law, or is "Conway's Law" just a hodgepodge term representing a myriad of personal interpretations and assumptions?

Interestingly, we found: 1) that most (75–100%)[6] of the 259 papers citing Conway's Law (from 2003–2012) use the law in the context of some other study;[7] 2) most of the citing papers (90%) reference Conway's original paper *exclusively*; 3) most of the citing papers (93%) are either observational studies or are not empirical; and 4) only one of the citing papers (<0.5%) is a replication. Thus, as of 2012, Conway's Law appears to be unanimously considered true, but with little or no substantiation. Furthermore, definitions and interpretations of Conway's Law vary substantially from paper to paper, with most authors showing no recognition that such differences in perspective could and do exist. Thus, rather than explore the nuances of

[6] Exact numbers are no longer available in this case, but the percentage is known to be well over half, likely near 100.

[7] Meaning that the authors do not directly test or otherwise assess the law, but rather apply it as a foregone conclusion in an analysis of some other related topic.

Conway's Law (in order to develop transferable knowledge), authors tend to use the law as a generic vehicle for incorporating into a research study, in a seemingly substantiated fashion, their own subconsciously-personalized Conwayesque ideas.

That said, the literature does contain a plethora of both anecdotal and observational accounts documenting Conway-like phenomena. Thus, Conway's Law is probably true in some form. The problem is simply that we know so little about how the phenomenon actually operates in practice that any specific characterization of it is likely to be inaccurate in almost all real-world contexts. In other words, our knowledge of Conway's Law is too general (and thus too brittle) to be used effectively in industry as an engineering principle.

External Paper 7 is a controlled test of a particular nuance of Conway's Law. The primary purpose of the study was to validate that highly differentiated replication is capable of producing valuable insight, even in the case of well-established (and presumably settled) research topics. Generally speaking, Conway's Law predicts that the structure of a software system will reflect the structure of the organization that built it. In turn, we hypothesized that the designers' *perception* of the ultimate composition of the development team would affect the resultant system architecture more so than would the actual composition of the design team.[8] Interestingly, the results of the study confirmed this hypothesis. Thus, we conclude: 1) many (if not most) phenomena in software engineering are likely understudied and, thus, in need of more replication; 2) highly differentiated replication has significant potential for uncovering new perspectives on old problems and, as such, ought to be further developed as a research method in software engineering; and 3) Conway's Law, in particular, is far from fully understood and, thus, in need of many more replications of various types.

External Paper 8 is also a controlled test of Conway's Law, but in this case, we attempted to replicate the phenomenon described by Conway as precisely as possible (i.e., we were looking to confirm Conway's observation, rather than to elucidate a particular nuance

[8] More specifically, we tested the hypothesis that a design team led to believe that the future development team would be comprised of multiple developers would, per Conway, yield a more modular architecture than would a design team with no expectations of the cardinality of the future development team.

of it). The study consisted of an experiment, in which small teams with strictly defined communication channels were tasked with designing a system. After giving the teams a pre-determined amount of time in which to work, we compared their various communication structures with their resulting system architectures. Interestingly, the communication barriers we set up in the experiment did inhibit communication in the resultant system architectures (as predicted by Conway's Law). However, participants also overcame many of the barriers via ad hoc mechanisms, such as providing software interfaces. Moreover, the degree of the effect of Conway's Law appeared to be largely a function of the individual personalities of the participants—which suggests that social dynamics play a role in the outcome of Conway's Law.

Based on this study, we conclude: 1) some form of Conway's Law is likely true; 2) due to a general lack of empirical evaluation, many nuances of Conway's Law have been overlooked (e.g., the interplay of personalities and social dynamics); and 3) Conway's Law is not an all-or-nothing phenomenon, but rather, a gradient, parameterized by a collection of contextual factors. Thus, Conway's Law is not simply an assertion to be validated or invalidated with a simple "yes" or "no"—instead, it is a complex phenomenon that requires in-depth empirical evaluation. Accordingly, Conway's Law is in need of replication. Furthermore, all replications of Conway's Law (regardless of type) should seek to identify relevant context variables, as well as to develop explanatory and predictive theories, in an effort to account for results across a variety of contexts.

1.8.3 Practical Component—Design Patterns

Chapters 2–4 represent contributions to and results from the RESER 2011 strict joint replication. The 2011 joint replication involved four separate research teams,[9] each of which closely replicated the same experiment on design patterns. The teams were coordinated via a web portal, which handled (and thus standardized) administration of the experiment. The

[9] Brigham Young University, USA [126], Freie Universität Berlin, Germany [167], University of Alabama, USA [155], and Universidad Politécnica de Madrid, Spain [103].

experiment, known as PatMain, tested the impact of design patterns on software maintenance. PatMain had been conducted twice previously (including the original study and one prior replication). We selected design patterns because, as previously explained, many studies (including replications) have investigated patterns, and to date their results are largely irreconcilable [6, 222] (likely due to the influence of context variables). Thus, the purpose of the 2011 joint replication was two-fold: 1) to test a new research method (strict joint replication), and 2) to make headway on the question of whether design patterns really have a measurable impact on software maintenance.

Chapter 2 represents our individual contribution to the 2011 strict joint replication. With respect to this dissertation, Chapter 2 accomplishes three objectives. First, it documents the unique details of the sub-replication we conducted at Brigham Young University (BYU). Second, it makes a first attempt at statistically modeling the results of the PatMain experiment, which experience helped guide the combined analysis (described in Chapter 3). Third, it documents BYU-specific insights on context variables and threats to validity for consideration in the combined analysis. Note that, at this stage of the process, we were not concerned with synthesizing the results across past studies or forming final conclusions.

Chapter 3 (including Appendices A–Y) documents the combined analysis of the 2011 strict joint replication. The primary objective of Chapter 3 was to synthesize results across all three iterations of the experiment (i.e., the original study, the first replication, and the joint replication). Note in this regard that the results diverged not only across the four sites of the joint replication, but also between the original study and the first replication.

To tackle the synthesis objective, we broke the problem down into four sub-tasks: 1) assess the heterogeneity of the results; 2) investigate context variables; 3) address the original hypotheses; and 4) generalize the results across all three studies. Interestingly, in assessing the heterogeneity of the results in the joint replication, we found much greater differences across the four sites (i.e., sub-replications) than within. Thus, the joint replication evoked—within the context of a single, controlled experiment—the broader problem of generalizability that confronts the PatMain series of studies. Consequently, we were able to attribute much of the variance observed across sites to meaningful variables rather than to experimental artifacts.

We ended up having to perform two analyses: 1) a post-hoc analysis of moderator variables, based on frequentist and Bayesian statistics; and 2) an a priori analysis of the original hypotheses, based on frequentist statistics. Via the post-hoc analysis, we identified both *developer experience* and *pattern knowledge* as moderating the effect of design patterns. In both cases, greater experience/knowledge tended to enhance the benefits of patterns (or reduce their harm) during maintenance. We also found indirect evidence for *motivation* as a moderator. At the very least, lack of motivation appeared to increase statistical variance, which in turn confounded the comparison of results. However, whether and to what degree motivation impacts the effect of patterns outside the experimental setting is still unclear.

The analysis of moderators resolved contradictions in the results across the three iterations of the experiment sufficient to allow for generalized conclusions. Consequently, the final conclusions, which represent 126 participants from five universities and twelve software companies, generalize across a broader set of contexts than has previously been achieved in the study of design patterns. The final conclusions can be summarized as follows:

1. The Decorator pattern is preferable to a simpler solution during maintenance, as long as the developer has at least some prior knowledge of the pattern.
2. For Abstract Factory, the simpler solution is mostly equivalent to the pattern solution.
3. Abstract Factory requires a higher level of pattern knowledge and/or developer experience than Decorator for the pattern to be beneficial.

Chapter 4 (including Appendix Z) documents the methodological results of the 2011 strict joint replication. Ultimately, solving the problem of synthesis with respect to the PatMain experiment required developing a new method for identifying/evaluating context variables—which method we refer to as TCA (*a tractable approach to context analysis*). TCA involves three components (described previously in Section 1.5.2): joint replication, post-hoc moderator analysis, and a specific type of Bayesian model. In addition to the descriptions given previously, note the following:

- *Joint replication* is a sampling process for obtaining sufficient, current, and clean raw data on context variables without requiring an excessive number of participants or studies.
- *Post-hoc moderator analysis* defines pre- and post-experiment processes for identifying salient context variables.
- *Bayesian models* represents a specific, quantitative process for evaluating context variables that supports conclusions even when statistical power is low.

The overall strategy of the method is to use Bayesian models to investigate moderators within the context of a joint replication. However, being defined in terms of interchangeable components, TCA can be adapted fairly easily to a variety of other circumstances (e.g., if suitable data are available from past studies, then joint replication may not be necessary).

Each of the three components provides multiple benefits that make it particularly suited to context analysis. For instance, joint replication increases real-world contextual variation while, at the same time, limiting artificial experimental variation; post-hoc moderator analysis is particularly suited to uncovering relationships that would otherwise remain undetected given only a priori methods; and Bayesian models allow researchers to form conclusions even when statistical power is low (for further details, see Chapter 4). Collectively, the three components enable the quantitative investigation of context variables in greater detail, with greater statistical power, and via considerably smaller datasets than previously possible. Thus, TCA can be used to reconcile contradictory experimental results, to improve the conduct of future replications, to resolve heterogeneity in meta-analysis studies, and to guide researchers in selecting information for reports and lab packages. With sufficient development, we anticipate the method could be used to produce generalized conclusions that are broadly and demonstrably applicable to industry.

Additionally, by attempting to *explain* divergent results, rather than to eliminate them, TCA naturally leads to explanatory theory. The benefit of such theory is that it allows researchers to shift the goal of replication from results reproducibility, which often fails, to experiment predictability, which can more easily cross contextual boundaries. Ultimately, *effective generalization requires testable theory about context by which to tie studies together into a knowledge framework*. TCA facilitates the development of such theory by making practically feasible the study of context variables.[10]

At this point, it is important to note that TCA represents one of the most significant contributions of this dissertation. Prior to TCA, software engineering had no way to compare replications in order to derive useful information—at least not a method that was both practically feasible and empirically/statistically grounded. Most replications (especially external replications) could do little more than report the inexplicability of their results; researchers simply had no way to confidently attribute variations in replication outcomes to causal factors.[11] In contrast, TCA provides a systematic (and configurable) method for learning context variables, which not only allows contradictory results to be reconciled across replications (i.e., solves the synthesis problem), but also promotes theory development.

1.8.4 Theoretical Component

Chapter 5 represents the final results of the theoretical component of the dissertation. As described in Section 1.5.1, Chapter 5 is the culmination of a grounded theory study of replication-related ideas from other disciplines (including sociology, psychology, linguistics, architecture, philosophy, and the natural sciences). Chapter 5 consists of two parts:

[10] Incidentally, software engineering researchers have been struggling for years to raise theory development to a level of standard practice [49, 98, 108, 195, 196]. If adopted, TCA may help to effect this change.

[11] Given such circumstances, it is no wonder that researchers have consistently avoided replication, especially external replication (as da Silva et al.'s survey shows [49]), or that cross-study synthesis is so often ignored, even among systematic literature reviews [48].

1. A distillation of external replication theory into a consolidated theory of knowledge production, which we refer to as the *Theory of Conceptual Frameworks*.
2. A preliminary application of the theory of conceptual frameworks to problems of replication and knowledge production in empirical software engineering.

The primary contribution of Chapter 5 is to define the role of replication as it relates to the broader process of science. In particular, via the theory of conceptual frameworks, we were able to show that replication is far more than a simple process of validation; rather, it is a central mechanism for learning in science and, as such, is always practiced in some form, whether recognized or not. Accordingly, a key objective for science is not just to define methods for executing isolated replications, but to systematize replication as a governing process for cross-study synthesis. In this way, the theoretical component motivates and reinforces the work of Chapters 2–4 (i.e., the PatMain strict joint replication).

In addition to clarifying the role of replication, Chapter 5 also presents a framework (based on the theory of conceptual frameworks) for developing a unified taxonomy of replication types. As part of that framework, Chapter 5 outlines a working model of replication, which resolves many of the ambiguities encountered when tyring to classify replications. Additionally, Chapter 5 demonstrates the inadvisability of drawing fixed lines around replication methodologies in such a way as to discredit novel experimentation.

Ultimately, the value of the theoretical component is three-fold. First, it represents a significant standalone contribution in that it provides a multi-disciplinary, theoretically-grounded framework in which to reason about replication problems. Second, it develops several practical ideas about replication that address key problems in empirical software engineering. Third, in an indirect way it facilitated the work of Chapters 3 and 4. Specifically, it equipped us to dissect and articulate replication problems more clearly and at a deeper level than previously possible, which in turn enabled us to conceive of the TCA method. In fact, many (if not most) of the critical insights behind the TCA method came about in direct response to the theoretical analysis.

Chapter 2

Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University

In 2001 Prechelt et al. published the results of a controlled experiment in software maintenance comparing design patterns to simpler solutions. Since that time, only one replication of the experiment has been performed (published in 2004). The replication found remarkably (though not surprisingly) different results. In this paper we present the results of another replication of Prechelt's experiment, conducted at Brigham Young University (BYU) in 2010. This replication was performed as part of a joint replication project hosted by the 2011 Workshop on Replication in Empirical Software Engineering Research (RESER). The data and results from this experiment are meant to be considered in connection with the results of other contributions to the joint replication project.

2.1 Introduction

Software design patterns reportedly provide significant benefits to developers, including increased system flexibility, reduced development time, and reduced software maintenance costs [20, 35, 37, 73, 77]. However, empirical support for design patterns is still largely anecdotal [168, 169] and, "as software engineers have discovered before. . . words such as 'clearly' and 'obviously' do not constitute confirmation" [168, p. 1134]. Although most design pattern claims seem "obvious," an experiment by Prechelt, Unger, Tichy, and Votta [168], which tested the impact of design pattern use on the maintenance of software code, reports *non-obvious* results, indicating either 1) that design pattern claims must be further qualified by additional contextual factors or 2) that Prechelt's experiment was mistaken in some of its conclusions.

To date, Prechelt's findings have been re-examined only once [212], from which the authors report *significantly* inconsistent findings from the original experiment.[1] In this paper, we describe the results of another replication of Prechelt's experiment, designed to further explore context factors and to validate the original observations and experimental framework. We conducted this replication at Brigham Young University (BYU) as part of a *strict joint replication*[2] project [75] associated with the 2011 Workshop on Replication in Empirical Software Engineering Research (RESER) [115, 116, 175]. The RESER 2011 joint replication project is aimed at two general goals: 1) to better understand the impact of design patterns on software construction, and 2) to test a new research methodology for conducting large-scale distributed replications (i.e., joint replications) in Software Engineering.

In the next section, we review the experiment by Prechelt et al., after which we discuss the replication setup in Section 2.3, including deviations we make from the original. In Section 2.4 we describe our treatment of the data and our choice of statistical analysis. Section 2.5 presents results and Section 2.6 reviews threats to validity, including our peer review of the experimental framework. We conclude by discussing lessons learned about the joint replication process in Section 2.7, followed by general conclusions in Section 2.8.

[1] Considering that the phenomenon under study involves human cognition, inconsistency of results in early replications is not surprising. Behavioral theorists, for instance, required nearly twenty years and approximately four hundred replications to uncover the seventy context variables necessary to control flatworms in an experimental setting. Apparently, "flatworms are very sensitive creatures" [55, p. 337].

[2] The *joint replication* process involves independently replicating the same study at multiple research sites, but coordinated by a central, sponsoring organization (e.g., a research lab, workshop, or committee). In a *strict* joint replication, participating labs share a common definition of the experiment design, but in all cases participants gather subjects and collect, clean, and analyze data independently. A combined, meta-analysis may be performed either from the published reports or by aggregating raw data. In the context of the RESER workshop, the joint replication has been coordinated by workshop organizers, including Prechelt, a principal investigator from the original experiment under consideration.

2.2 Original Experiment (PatMain)

The original experiment by Prechelt et al. dealt with design patterns in software maintenance (abbreviated as "PatMain"). The PatMain study, performed in 1997 and published in 2001, was the second controlled experiment (after "PatDoc" [169]) ever conducted to examine software design patterns [168]. The PatMain experiment addressed the following question: *For a given problem, if using a design pattern is "overkill" (i.e., the pattern provides more functionality or flexibility than necessary) will the resulting solution be more difficult to maintain than if a simplified solution were implemented instead?*

The experiment design involved four different, rather small, C++ programs and six software design patterns:

- CO: Communication Channels (Decorator)
- GR: Graphics Library (Composite & Abstract Factory)
- ST: Stock Ticker (Observer)
- BO: Boolean Formulas (Composite & Visitor)

Each of these programs existed in two versions: one based on design patterns (PAT version), and another implementing a simplified design without patterns (ALT). The experiment was performed with *professionals working on paper*, and included two types of tasks for each program—comprehension-oriented tasks and code-modification tasks. The experiment began with pre-tasks, in which each subject worked on one PAT and one ALT version (of two of the four programs), followed by a training course on design patterns, followed by a set of post-tasks, in which subjects worked on the remaining two programs (again one PAT and one ALT version). Version ordering was alternated within and across subjects.

This design resulted in three key independent variables: 1) *program* (with levels CO, GR, ST, BO); 2) *version* (with levels PAT, ALT); and 3) *pattern knowledge* (with levels PRE, POST). The experiment measured two dependent variables: 1) *time*—how long it took subjects to complete each task, and 2) *correctness*—measuring completeness and correctness of the solution. Hypotheses were structured around the impact that *version* and *pattern knowledge* were expected to have on *time* and *correctness*. For additional details, we refer readers to the original publication [168] and to the first replication [212].

2.3 Replication Setup

In this section we describe the framework provided as part of the RESER 2011 joint replication project, including key differences between this framework and the design of the original PatMain experiment. We also review details specific to our instance of the joint replication, including the lab package for this experiment, subject recruitment procedures, and an overview of subject demographics.

2.3.1 Joint Replication Framework

Rather than pre-test, train, then post-test, the joint replication framework administers a single test (without training), involving only two of the four programs: CO (Communication Channels) and GR (Graphics Library). The framework excludes two programs for practical reasons—to reduce complexity, so participating research labs can more strictly replicate the experiment. Of the four programs, ST (Stock Ticker) is excluded because it is considered to be "relatively uninteresting," and BO (Boolean Formulas) is excluded because the Visitor pattern is considered to be "overly difficult" [75]. However, the experiment framework still provides an option to use ST and BO, though the general consensus among participants is to test on only CO and GR.

Another change from the original experiment (but consistent with the first replication [212]) is that subjects implement solutions on a computer, rather than on paper. The joint replication framework is deployed via a web portal. Subjects log in to the application using a randomly assigned ID (no personal identifiers). The application presents a questionnaire to assess the subject's experience with programming and design patterns (11 questions). This questionnaire provides control data for use in the statistical analysis (in lieu of the pre-test, training, post-test protocol). Subjects are then asked to perform minor maintenance tasks on two programs—one PAT version and one ALT version—and program versions and version ordering are alternated across subjects. All exercises are timed. For each program there is one programming exercise (subjects download, modify, and upload source code) and one exercise testing understanding of the program (1–2 short answer questions). The source code download screens recommend Eclipse as the IDE to use for code modifications and provide basic instructions on importing projects. At the completion of each program, subjects are asked to assess their performance (6 questions). At the end of the study, subjects are allowed to submit final comments. Total anticipated time commitment (based on the original experiment) is 2–3 hours. Also, because the experiment is administered online, direct interaction with subjects is limited to administrative tasks—i.e., all experiment data is collected through the application.

The experiment framework generates a list of random IDs (to be used by subjects to log in to the experiment). These IDs associate subjects with one of four groups ($group\# = ID\%4$). By assigning IDs in order, subjects are evenly bucketed into the following experimental groups:

- Group 0 receives GR-PAT followed by CO-ALT
- Group 1 receives CO-ALT followed by GR-PAT
- Group 2 receives GR-ALT followed by CO-PAT
- Group 3 receives CO-PAT followed by GR-ALT

At no time are subjects explicitly informed that the experiment includes multiple program versions, nor are they made aware of alternate program orderings or of the experimental groups. *For a given program*, all subjects see the same text regardless of the assigned treatment group.

In the original experiment, all programs were implemented in C++, but the joint replication includes Java and C# as well—translated from the original experiment. Also, although some questions have been modified and the language of the experiment has been translated from the original German, the intent of the instruments is unchanged.

2.3.2 Brigham Young University Specifics

The replication conducted at BYU involved only the CO and GR programs, and (according to the joint replication framework) subjects received *no* design pattern training as part of the experiment. We conducted our replication over the course of approximately three weeks. Subjects received login IDs on November 15, 2010 and completed the experiment at home on their own time by December 6, 2010. We instructed all subjects to complete the experiment in Java; all complied.

Lab Package

A lab package for this experiment is posted on the BYU SEQuOIA Lab website (see "Publications" page) [38]. The lab package includes the raw and cleaned data sets (with a data summary document), copies of the original experiment source code and the subjects' modified source code, SAS (Statistical Analysis Software) version 9.2 code with output, experiment screen shots documenting all questions and instructions given to subjects, and copies of the informed consent and subject instructions documents.

Subject Recruitment

The experiment protocol for this replication was approved by the BYU Institutional Review Board, and subjects were recruited according to standard practices for human subjects research (i.e., informed consent documents, voluntary participation, etc.). The BYU subject pool was drawn from a single, senior-level undergraduate software engineering course (offered by the BYU Department of Computer Science). The course is optional for students—one of many computer science courses that may be taken for senior degree credit.

The software engineering course was selected based on three criteria: 1) students in their senior year in the Computer Science program understand the programming concepts necessary to complete the experiment; 2) one of the course's prerequisites is a junior-level course involving training on design patterns (textbook by Gamma, Helm, Johnson, and Vlissides [77]), which is desirable for this experiment; and 3) students are the most available resource in a campus environment. Admittedly, students may differ from professionals in their response to design patterns either because they lack industry experience or because they have been trained more recently on design patterns (or for some other reason). With respect to the joint replication, we believe students (i.e., future developers) represent an interesting cross-section of the developer population for comparison.

The selected course already includes required hours that the students must spend each week engaged in software engineering activities (of their choosing). The hours are not graded, only checked for completion. In exchange for participation in this study, subjects were allowed to count participation time towards the required weekly course hours.

Subjects were also allowed to withdraw at any time without penalty. However, since the experiment requires completion in order for a subject's data to be used, we informed students that withdrawing prematurely for reasons reasonably preventable (e.g., loss of interest) would result in no course credit for participation time. Conversely, a withdrawal due to reasons beyond the subject's control (e.g., technical problems of the experiment website) would still merit course credit. These policies were explained in the informed consent document, as well as verbally at the time of solicitation. Note that of those subjects who began the experiment (i.e., accessed the online application) none withdrew prematurely for any reason.

Prof. Charles Knutson, a co-investigator on this research project, teaches the software engineering course in which we solicited subjects. To avoid placing undue pressure on students to participate, all interactions with students regarding the project, including the initial solicitation, were conducted solely by the principle investigator (and first author of this paper)—outside the presence of the course instructor.

The first author solicited subjects in class following a shortened lecture. After discussing the informed consent document, he issued login IDs with an instruction sheet to all students who wished to participate. In order to randomly assign students to experiment groups while still ensuring that groups were filled evenly (by handing out IDs in order), he made assignments according to a randomized class roll. Individuals who did not wish to participate were skipped. The instruction sheet issued with the login ID included the following instructions (quoted verbatim):

> This study is timed and must be completed in one sitting without interruption. Do not close the browser during the study. IDs can only be used once to login. Do not use your browser's back button. Only use links provided on the page. When asked to choose a programming language to work in, select Java. Please do not discuss the experiment with any other subjects (i.e., CS428 students). For technical or other difficulties contact [. . .].

Subject Demographics

At the time of solicitation, the course enrollment totaled 46 students, two of whom were female. Although we did not collect age statistics from subjects, all students in the solicited course were between the ages of 18 and 30—typical college-age undergraduates. Of the 46 students in the class, 36 accepted a login ID, and 22 completed the experiment. We informed the students that if they did not wish to publicly refuse participation it was acceptable to take a login ID, even knowing they would not complete the experiment. Other reasons (e.g., pressure from coursework) may also account for those who "signed up" but did not participate. Because some students failed to complete the experiment, group sizes were only roughly even. Of those who completed, 6 were assigned to group 0, 5 to group 1, 4 to group 2, and 7 to group 3.

The original experiment utilized a random, stratified block design. Stratification in that case was enabled by the pre-questionnaire, from which subjects were sorted based on software development experience. For our replication, however, we did not stratify the experimental blocks. First, the pre-questionnaire was administered through the web portal at the time of the experiment, which did not allow us to use the results to make block assignments. Second, based on age, college major, classes taken, locale, etc., we anticipated that the subject pool was homogeneous with respect to development experience and design pattern knowledge. As we show, this assumption is supported by the data.

Based on pre-questionnaire responses, subjects are predominantly Computer Science majors (21/22)—with only one subject studying Computer Engineering and one double majoring in both Computer Science and Animation. Almost all of the subjects (20/22) are undergraduate students, with the remaining two listing a graduate student status. As for "professional" development experience, most subjects (14/22) report zero years. Of the eight subjects who report some professional experience, the maximum listed was five years, with the other seven each listing fewer than four years. All subjects list Java as a frequently-used language, and most list C, C++, and C# as well. The average number of frequently-used languages listed is 5 (with a median of 4 and standard deviation of 2.5). With respect to software development experience, the most significant difference that we found between subjects showed up in their reports of lifetime lines of code (LOC) written. However, we distrust these reports due to the difficulty of estimating such a granular unit over such a long period of time. Case in point, 8/22 subjects listed 100,000 LOC or more, with one listing 8 million, which seems unlikely for the average undergraduate student.

The average number of patterns with which subjects report having ever worked is 9.6 (median of 8). The distribution is somewhat skewed by three outliers—two subjects rated themselves at 20 patterns, and one at 30 patterns. We also asked subjects to rate their individual knowledge of 18 specific patterns on a 7-point scale (where 1=never heard of it, 2=have only heard of it, 3=understand it roughly, 4=understand it well, 5=understand it well and have worked with it once, 6=understand it well and have worked with it two or three times, 7=understand it well and have worked with it many times). Averaging across all patterns, the collective score is 2.9 (standard deviation of 0.76). The individual averages for patterns specifically involved in the experiment—Decorator, Composite, and Abstract Factory—are 3.6, 3.5, and 3.5 (with standard deviations of 1.7, 2.0, and 0.9, respectively). In general, the data support our hypothesis that subjects are homogeneous with respect to pattern knowledge. They each have a general understanding of numerous design patterns, which is reflective of the fact that the subjects have been trained recently (within 1–2 years, based on a required prerequisite course). However, the data also indicate a general lack of *working* experience with design patterns, which is consistent with the fact that all of the subjects are students.

2.4 Analysis

In this section we discuss our treatment of the raw data—including our process for grading solution correctness, transformations we make to the data, and columns we discard prior to the statistical analysis.

2.4.1 Grading Solutions for Correctness

Two Computer Science graduate researchers graded solutions for correctness. One is a doctoral student who has worked professionally as a software developer. The other is a master's student who has worked professionally as a software tester. Both graders are qualified to assess software quality. Source code and final correctness scores for all tasks are included in the lab package (see Section 2.3.2).

For task 1 of each program, requiring modification of source code, the graders worked together in a pair-programming style arrangement. They initially reviewed several of the solutions in each program to determine how best to grade them. The correctness rubric they settled on is a five-point scale (similar to that used in the first PatMain replication by Vokáč et al. [212]):

1. *Requirements misunderstood*—the solution is completely wrong and it does not appear that the subject understood the requirements.
2. *Wrong answer*—it appears that the subject understood the requirements, but they did not produce the correct solution, even conceptually.
3. *Right idea*—the solution conceptually addresses the requirements, but is incomplete or contains an error; the solution also does not compile.
4. *Almost correct*—the solution conceptually addresses the requirements and compiles, but is incomplete or contains a functional error.
5. *Correct*—the solution is completely correct and fully meets the stated requirements (i.e., no functional or compiler errors).

The other three comprehension tasks (tasks 2–3 for CO and task 2 for GR) were graded by dividing the subjects in half—each grader assessed the remaining three tasks for 11/22 subjects. The graders agreed on a rubric for these tasks and compared results until they felt that the grading was consistent. For these comprehension-oriented tasks the rubric is binary (i.e., 0=incorrect, 1=correct). The graders report that "for the most part, it didn't seem like people were 'BS-ing'—they seemed to either know it, or not." The graders also report that they graded holistically, looking at everything a subject said, rather than simply searching for a specific answer.

2.4.2 Data Preparation

The following is an exhaustive list of procedures that we apply to the raw data prior to statistical analysis. We provide this information to enable other researchers to make more clear judgments about our conclusions, as well as to enable future replication. For those wishing to build on this work, *we provide copies of the raw and cleaned data sets in the lab package* (see Section 2.3.2). We also encourage the interested reader to review this section in connection with the data summary from the lab package.

Dependent Variables—Task Timings and Correctness Scores

The experiment framework records time spent by subjects on each page of the application. We sum timings for the download, task 1, and upload pages (of each program) into a single variable because of "back-button" capabilities. Several subjects, for instance, spent considerable time on the download page, which means they may have begun studying the source code before moving on to the task page or, quite possibly, they used the back button to return to the download page (e.g., to review the program description).[3]

Also note that timings for tasks 2–3 of the CO program are combined because the questions were presented on the same page of the online application. Having to combine them is unfortunate because task 2 is comprehension-oriented, whereas task 3 is code-oriented (similar to task 1). For this reason the first replication (by Vokáč et al.) excluded task 3 from the experiment. In our case, both are included and confounded. Thus we must also combine the correctness scores for these tasks by taking the average. From this point forward, we treat tasks 2–3 of the CO program as one task (which we refer to as "task 2").[4] For the statistical analysis we also convert task 2 correctness scores to a 5-point scale (i.e., 1=incorrect, 5=correct) to be consistent with task 1.

Summary statistics for both *time* and *correctness* are shown in Table 2.1. To prepare *time* for a parametric statistical model (which assumes a normal distribution), we apply a natural log transformation (similar to the first replication)—without which the timings are skewed with outliers. We do not transform the correctness scores because their range is small, thus precluding the possibility of gross outliers (a primary threat to model validity and a principle reason for concern with normality). We also note that meeting the normality assumption through data transformation is a distinctly different approach from the bootstrapping method used in the original experiment.

[3] Or perhaps they took a phone call, which we suspect of subject 59259, based on final comments (see lab package, Section 2.3.2).

[4] Accordingly, the raw data set in the lab package includes tasks 2 and 3, whereas the cleaned data set includes only a "Task 2".

Table 2.1: Summary statistics for dependent variables *time* (seconds) and *correctness* (5-point scale).

		CO		GR	
		Task 1	Task 2	Task 1	Task 2
Time	Mean	1,869	476	1,003	666
	Median	1,531	444	902	531
	Std. Dev.	943	205	561	565
Correctness	Mean	3.1	3.4	3.7	2.3
	Median	3.5	3.0	4.0	1.0
	Std. Dev.	1.5	1.7	1.5	1.9

Independent Variables—Software Development Experience

20/22 subjects are junior or senior undergraduate students, with the remaining two being graduate students, so we exclude student status from the analysis. Note in this regard that due to the small sample size (22 subjects), we are encouraged to limit the number of variables we include in the statistical analysis. Accordingly, we also ignore student major because all but one subject studies Computer Science.

Two questions in the software development experience pre-survey ask about the number of LOC subjects have written. One asks for an estimate of the total LOC ever written in any language and the other asks the same question with respect to Java. The averages across subjects for these two metrics are approximately 450K and 220K, respectively, which seems far too high for undergraduate students with less than one year (on average) of professional experience. Removing the most significant outlier (8 million lifetime LOC), the averages drop to approximately 80K and 40K, which seems more reasonable. However, several other subjects also report large numbers (e.g., 0.5 million), and in one case the LOC in Java estimate is dramatically higher than LOC in any language, which is impossible. In general, the ratio of suspicious responses to subjects is just too high to allow removal of outliers, so we are forced to exclude both variables entirely.

We also convert the lists of programming languages ever used to a numerical count and exclude from statistical analysis the lists of languages used often. In this regard, one subject (57033) stated in his or her final comments, “I couldn’t run the GraphicsP program because I'm so rusty in Java that. . . " This same subject listed Java as a language "I know well and have worked with several times." In general, we suspect that the languages-used-often question—those supposedly known well and used several times—may have elicited responses similar to the languages-ever-used question. The latter is intended to capture breadth of experience, the former depth. Because the depth question (languages-used-often) seems to have been interpreted similar to the breadth question, we ignore it in preference for the list of languages ever used.

We also ignore working hours/week and years of professional experience. First, a dependency between the two questions is implied by their being included in the same statement on the questionnaire. In fact, of subjects who list zero years experience, all correspondingly list zero working hours/week (and the reverse correspondence is also true). Second, we believe the term "professional" is ambiguous for students. Should they count part-time work, or does this question refer strictly to full-time jobs? 6/22 subjects report part-time hours (9–22 hours/week), whereas 2 subjects list 40 hours/week. Ultimately, most subjects report 0 years experience (and correspondingly, 0 working hours/week), at least some of whom are likely programming part-time at jobs they simply do not consider "professional."

We have some concern that self-assessments of programming skill may be overly biased (i.e., too noisy) for a statistical analysis with only 22 subjects. In particular, subjects are asked to rate themselves on a 7-point scale relative "to all other programmers in the world." Since the question (as worded) seems difficult to answer, we look for reason to discard the variable. To this end, we (temporarily) convert the languages-ever-used variable to a 7-point scale and compare it against self-proclaimed programming skill. In doing so, we find that subjects who report having used more languages also tend to rate their programming skills more highly (correlation of 0.458, p-value=0.03). Although correlations less than 0.7 are not considered problematic with regard to multicollinearity, the correlation is sufficient that we feel comfortable dropping the self-assessed programming-skill variable in favor of the count of languages ever used.

At this point, we have two remaining variables that assess software development experience—languages ever used and programming hours/week—both of which we include in the statistical analysis. The two metrics are not correlated (0.212, p-value=0.34), and we believe that together they address both breadth and depth of experience.

Independent Variables—Pattern Knowledge

First, we ignore the count of patterns ever used because many of the values look too high (e.g., one subject reported having used 30 patterns and several reported 20 patterns). We believe the question is problematic, similar to estimating total lifetime LOC written. Instead, we represent design pattern knowledge by averaging the subjects' estimates of pattern knowledge for the 18 individual patterns specifically listed on the questionnaire.[5] Individual design pattern knowledge questions are more focused, and we believe subjects are able to answer them more accurately. To further reduce the number of variables in the statistical model, we retain individual pattern knowledge estimates for only those patterns that are directly involved in the experiment. Thus we arrive at four pattern knowledge variables to include in the statistical analysis—our aggregate score, as well as individual scores for the Decorator, Composite, and Abstract Factory patterns.

Other Data

We ignore pre-survey timings as unrelated to the experiment hypotheses, and we ignore post-surveys (for the statistical analysis) because the original experiment did not report them. Similar to the first replication, we use the post-surveys to help us interpret the statistical results.

[5] Before averaging, we shift all scores down so that the scale begins at 0.

2.4.3 Statical Evaluation Method

We analyze the data with linear mixed models (containing both fixed and random effects). We include SAS code with output in the lab package (see Section 2.3.2) for this analysis. Because the CO and GR programs are significantly different, with different task-specific hypotheses, we do not analyze them in the same statistical model. We also run separate models for each dependent variable. When time is included as a dependent variable, correctness is treated as a co-variate (and vice versa). Thus we build a total of four statistical models: CO-Time, CO-Correctness, GR-Time, and GR-Correctness.

CO models initially include ten variables: *id*—the subject's login ID; *languages*—a count of the number of programming languages the subject reports having ever used; *hours programming*—the number of hours the subject reports having spent programming per week; *pattern knowledge*—an aggregate score of knowledge/experience self-assessed across 18 specific patterns; *decorator pattern*—the subject's self-assessed knowledge/experience score for the Decorator pattern; *order*—whether GR tasks were administered before or after CO tasks; *task*—1 or 2; *version*—PAT or ALT; *correctness*—a measure of the completeness and correctness of the subject's solution for each task; and *time*—a measure of the time spent by the subject solving each task.

GR models include the same initial variables as the CO models, except that *decorator pattern* (which applied specifically to the CO program) is replaced by the subject's scores for the Abstract Factory pattern (*abstract pattern*) and the Composite pattern (*composite pattern*), which are relevant to the GR program. Thus GR models initially include eleven variables. All variables except for *id* are treated as fixed effects; *id* is included as a random effect.

2.4.4 Model Tuning

For this discussion, it will be helpful to refer to the *time* and *correctness* variables as dependent variables, the *task* and *version* variables as main effects, and all others as co-variates. To optimize our statistical models, we first run each model without the main effects (*task* and *version*). The purpose of this procedure is to test the relationship between the co-variates and each dependent variable, without respect to the main effects. Any co-variates that are not statistically significant are dropped from the model as unnecessary noise (thus allowing more degrees of freedom and a more efficient analysis). Pruning of co-variates is done one variable at a time, dropping the least significant co-variate followed by recalculation of the model. This process continues until only statistically significant co-variates remain. Once the models are optimized, we add the main effects (with an interaction term *task * version*) and calculate the final results.

The Grungy Details

For the CO program, all co-variates are ultimately dropped from the models of time and correctness. Co-variates for CO-Time are dropped in the following order: *correctness* (p-value=0.9992), *languages* (p-value=0.48), *hours programming* (p-value=0.50), *decorator pattern* (p-value=0.50), *pattern knowledge* (p-value=0.37), and *order* (p-value=0.28). Co-variates for CO-Correctness are dropped in the following order: *time* (p-value=0.9992), *hours programming* (p-value=0.98), *order* (p-value=0.78), *decorator pattern* (p-value=0.57), *pattern knowledge* (p-value=0.45), and *languages* (p-value=0.22).

GR models retain some co-variates as significant. Co-variates for GR-Time are dropped in the following order: *abstract pattern* (p-value=0.98), *pattern knowledge* (p-value=0.97), *composite pattern* (p-value=0.88), and *languages* (p-value=0.34). The co-variates *correctness*, *order*, and *hours programming* are retained (p-values=0.006, 0.022, and 0.088, respectively). Although the p-value for *hours programming* is not statistically significant at the $\alpha_{.05}$ level, it is certainly close enough to be suggestive, so we keep the variable in the model. Co-variates for GR-Correctness are dropped in the following order: *hours programming* (p-value=0.96), *composite pattern* (p-value=0.83), *languages* (p-value=0.82), *pattern knowledge* (p-value=0.82), *order* (p-value=0.71), and *abstract pattern* (p-value=0.54). The co-variate *time* is retained (p-value=0.003).

Discussion

The non-significance of co-variates in the CO models may be the result of insufficient data (which a joint analysis could overcome), and/or it may reflect a relative lack of knowledge about context variables that affect programmer performance. The latter case is certainly at least partially true [125] and can only be overcome through additional replication on this topic.[6] However, the lack of significance of most co-variates is also not entirely unexpected. The result is consistent with our assertion that the sample is homogeneous with respect to programming and pattern knowledge.

Also consistent with expectations (and with the original experiment), program order was generally insignificant. However, in the case of GR-Time, order did turn out to be significant. As a partial explanation, we point out that the CO program tasks required approximately 11.25 more minutes to complete (on average), than did the GR program tasks. It is quite possible that when the CO program was tackled first, it caused subjects to become tired (or frustrated), thus they rushed more quickly through the GR tasks. However, because the GR tasks were "easier," assigning those tasks first may not have had the same effect on the CO tasks. This hypothesis is supported by the post-task questionnaires, in which numerous subjects indicated for the CO tasks that they had trouble understanding the requirements, whereas for the GR tasks, several subjects commented that "nothing was inherently difficult about the task." Nevertheless, we refer to this as a partial explanation because, ultimately, these results rest on a sample size of only 22 subjects.

We also note that time and correctness correlate for the GR program, but not for the CO program (i.e., time and correctness were ultimately dropped as co-variates from the CO models). We anticipated that time and correctness would correlate heavily for *both* programs.

[6] Again, we refer the reader to the empirical study of flatworms from behavioral science [55, pp. 335–339].

Table 2.2: Statistical significance (p-values) of final model variables (fixed effects).

CO-Time		CO-Correctness	
task	< 0.0001	*task*	0.5830
version	0.1199	*version*	0.1523
*task * version*	0.8106	*task * version*	1.0000
GR-Time		GR-Correctness	
order	0.0181	*time*	0.0225
correctness	0.0567	*task*	0.1472
hours programming	0.0719	*version*	0.9168
task	0.0285	*task * version*	0.3224
version	0.8610		
*task * version*	0.4080		

At this point, we have not been able to identify an obvious explanatory hypothesis; we fall back on the inadequate sample size as the most supportable hypothesis available.

2.5 Results

To optimize the statistical models (Section 2.4), we include program order, solution correctness, and programming hours/week as co-variates in the GR-Time model. We also include task completion time as co-variate in the GR-Correctness model. All of these co-variates remain at least marginally significant in the final models (see Table 2.2). According to the GR-Time model, assigning a subject to complete the GR tasks after the CO tasks correlates with a 38% *decrease* in GR completion time on average. Conversely, a one point increase on the 5-point correctness scale correlates with an *increase* in GR completion time of 11%. As for hours spent programming each week, an increase of 10 hours/week appears to reduce GR completion time by 14% on average.

Having summarized results for the co-variates, we now turn our attention to the main effects and the experiment hypotheses. All four models include the main effects *task* and *version*, as well as the interaction term *task * version*. Individually, *task* and *version* are meaningless (within the context of this study)—obviously, varying the assigned task will impact completion time, and possibly solution correctness (as the p-values in Table 2.2

indicate). Thus we focus our discussion on the *task * version* interaction, allowing us to inspect—within a specific task—the impact that *version* has on the dependent variables.

In all four models, the *task * version* interaction appears (not surprisingly) insignificant[7] (see Table 2.2). However, based on the original experiment and first replication, we believe the use of patterns does make a difference in some cases for completion time. If this hypothesis is true, then our statistical models simply lack the necessary power to identify those differences because we have too few data points. However, the purpose of this replication is not to stand alone, but to be included in a much larger joint analysis.

Nevertheless, for completeness we present estimates for the *task * version* interaction in Table 2.3. Based on our data (and being clear that we cannot rule out the null hypothesis), we find that *for all but one task, the patterns versions (PAT) took longer and resulted in more solution errors.* The only case in which the PAT version outperformed the ALT version was on Task 1 of the GR program. For this task, subjects who received the PAT version completed their solutions 10% *faster* and with 18% *fewer* errors on average—contrary to the original experiment. On task 2 of the GR program, however, subjects who received the PAT version required 19% *more* time to finish and committed 16% *more* errors. In this case the non-significance of the results is consistent with the original experiment. For task 1 of the CO program, subjects who received the PAT version took 30% *more* time to complete the task[8] and scored 21% *lower* on the correctness scale—contrary to the original experiment.

[7] It is worth noting that the interaction is dramatically more significant for both time and correctness in the case of the GR program. One possible explanation is that subjects had trouble understanding the CO program task requirements, and thus the data is noisier. In this regard, we note that numerous subjects commented in the post-task questionnaire that the most difficult aspect of the CO program was understanding the requirements.

[8] In his (or her) final comments, subject 57033 mentioned an IDE problem that consumed significant time on the first task (see lab package, Section 2.3.2). This time appears to be lumped into the subject's download time for task 1 (CO program), making it an outlier. We cannot subtract time because the subject stated a broad window (1–2 hours) for dealing with the problem, and we cannot discard the download time because the task and upload timings are very short—i.e., the subject may have worked on the task on the download page. Also, setting up the IDE was (apparently) part of the experiment, per the instructions, so we keep this subject's data as valid. However, since the subject received the ALT version, discarding his data would only increase the difference between the PAT and ALT versions on task 1 of the CO program (i.e., the conclusions are unaffected).

Table 2.3: Estimates for the *task * version* interaction. Time (seconds) was transformed (natural log) prior to statistical analysis to meet normality assumptions; the back-transformed results are included in this table. Correctness is on a 5-point scale (1=incorrect, 5=correct). Note also that differences between versions are *not* statistically significant.

*task*version*	Mean	Original Scale (e^{mean})	Mean
	CO-Time		CO-Correctness
1-ALT	7.2826	1,455 sec (24.3 min)	3.4545
1-PAT	7.5426	1,887 sec (31.5 min)	2.7273
2-ALT	5.9646	389 sec (6.5 min)	3.7273
2-PAT	6.1675	477 sec (8.0 min)	3.0000
	GR-Time		GR-Correctness
1-ALT	6.7917	890 sec (14.8 min)	3.1125
1-PAT	6.6838	799 sec (13.3 min)	3.6730
2-ALT	6.2215	503 sec (8.4 min)	2.7893
2-PAT	6.3961	600 sec (10.0 min)	2.3343

On task 2 of the CO program, subjects took 22% *longer* on the PAT version and scored 20% *lower* on the correctness scale—this time consistent with the original experiment.

Based solely on these results, we would have to conclude (firstly) that the use of design patterns looks suspiciously unhelpful for future software maintenance (at least for those who employ undergraduate Computer Science students). However, (secondly) the core results are not statistically significant, so increasing the sample size may lead to significantly different findings. Ultimately, our results both confirm and contradict the original experiment and the first replication—which in turn contradict one another—and so we are forced to conclude (thirdly) that the patterns question remains wide open. We (the community) have not yet aggregated enough data to form solid conclusions, nor do we sufficiently understand all of the contextual factors affecting the study of design patterns in software maintenance.

2.6 Additional Context Information and Threats to Validity

It is impossible to list all details of an experiment—especially since, as researchers, we are not even aware of all the conditions that impact subject performance. Nevertheless, in this section we attempt to list as many additional details as possible. We believe these details are relevant to our conclusions, to future interpretation of our data, to any meta-level analysis conducted across the individual joint replications, and to future replications on this topic. Some of these conditions may seem like minor nuances, but such "nuances" have been shown in other fields to be quite significant [55, pp. 335–339], [178, p. 258].

2.6.1 Framework Concerns

When subjects download program code from the online framework, the code is delivered in a .zip file. The file's name is appended with either an A or P (ALT or PAT version, respectively). Exposing this naming convention to the user may create a bias—subjects may realize, for instance, that the 'P' stands for "patterns." At the very least, they may become aware that the two programs differ by experimental design. Related to this issue, comments in the source code explicitly state the names of patterns being used (e.g., "Generator is an ABSTRACT FACTORY"). This might compromise some of the questions on the post-questionnaire (e.g., "Which design patterns have you noticed in the program?"). Further, the absence of pattern comments in ALT versions tells the subject still more information about the experiment setup.

The Java and C# programs have similar class-file structures. However, the C++ variant lumps all code into one file (two counting header). Though this issue does not affect our individual replication (which was conducted in Java), it should be considered as a potential confounding factor in any meta-analysis that compares across languages.

Instructions on the program download pages of the joint replication framework differ depending on the programming language selected. If the user selects Java, the instructions include a recommendation to use Eclipse. The C# and C++ download pages do not provide IDE instructions. The purpose of providing IDE guidance was to help inexperienced subjects, but giving no IDE guidance may lead to subjects using their normal development environment (be it Eclipse, Emacs, or whatever), which may produce results that more accurately represent real-world performance. In other words, suggesting a particular IDE biases the working environment of the subjects—of course, in this case only for Java developers.

A back link is included on the first task page of each program (to return to the download page). Consequently, task 1 page timings do not necessarily represent the total time spent working on the first tasks. To deal with this we include download/upload times as part of the total task completion time. Although this practice shifts the data, it down not affect the difference between PAT/ALT versions. A similar problem is the use of the *browser* back button. All changes in cases where pages are accessed by the browser back button are propagated to the database, and although page timings are accurately aggregated when the back button is used, the mere availability of a back button significantly multiplies the number of framework use cases. We are not confident that all use cases lead to accurate measurements. Though we cannot guarantee that all complied, we instructed subjects not to use the browser back button.

The language of the experiment was translated from the original German version. Unfortunately, several awkward sentence structures occurred which were not corrected. Based on tests of the framework (performed by two members of our own lab), we believe subjects understood the material despite errors in the text. Also, to prepare subjects we verbally informed them of the translation errors and invited them to note in their final comments any complications they encountered as a result of the errors. Only one final comment mentioned the errors (subject 64084): “If you go through the first page of questions there are a few instructions/questions that don’t make grammatical sense.”

2.6.2 Solution Grading Concerns

For our replication, we instructed all subjects to complete the experiment in Java. However, the Java code—having been translated from C++—is written in a strong C style, which the graders felt might have been confusing for subjects who were anticipating a Java paradigm. As an example, the Java class “CommLayer” (in the CO-PAT program) does not have a constructor that accepts a “Channel” object, which is a slight (and confusing) violation of the Decorator pattern.

The programming task for the CO program required inserting Hamming encode/decode functionality. Within the ALT version, subjects had to insert this functionality into a chain of “if” statements, thus requiring them to consider the correct ordering of steps (e.g., compress, encrypt, encode, log). Within the PAT version, however, subjects only had to create a Decorator object to manage the encoding/decoding functionality (but did not have to instantiate that object), so the subject never had to deal with the issue of ordering.

Using Eclipse utilities, the PAT version of GR task 1 could be completed with only two lines of manually written code. However, the ALT version required manually writing numerous lines of code. One might argue that the use of design patterns enables (in this case) the use of IDE code-generation tools, so using those tools should be part of the experiment. Nevertheless, when interpreting timings for task 1 of the GR program, the reader should note that the impact (on developer comprehension) of using design patterns is confounded with the use of IDE capabilities. We cannot say whether the use of design patterns specifically led to quicker program comprehension (and thus reduced development time), or whether the improvement occurred simply because the design patterns enabled the use of code-generation tools.

Some method bodies in the Java code contained only the comment, “/* Body doesn’t matter */”, with no return statement, causing the source code to not compile. In this regard, the graders note, “We encouraged developers to load the data into an IDE, but then they see a lot of red—it’s unsettling and distracting.” Also, providing non-compilable code may indicate to the subject that code compilation is not a requirement. One subject (74027) specifically mentioned compilation problems (see final comments in lab package, Section 2.3.2).

Similarly, the provided code did not support execution by default for either CO or GR. For CO there was simply no main method provided by which to run the code. For GR there was a “testrun” class, but it worked for only one of the two initial output modes of the program. Since in this case the task was to add a third output mode, the "testrun" class may have given subjects mixed signals on whether their code should be executable or not. The graders report that "without executable code grading correctness is more difficult—difficult to be objective because we can't use a test suite." Also, many developers solve problems by testing as they go. When the code is not executable by default, we force them to either spend time fixing it (which some subjects did, though it was not the assigned task), or to interact with the problem in an abnormal way. The graders summed up the non-compilation, non-execution issues by stating, "Sometimes they can compile, sometimes they can't. Sometimes they can partially execute, sometime they can't. Sends mixed messages, mixed expectations."

2.7 Joint Replication as a Methodology—Lessons Learned

Through participation in the RESER 2011 joint replication project, we have discovered several practical trade-offs of joint replications (as opposed to stand-alone experiments):

First, in the case of a *strict* joint replication, it is much easier to setup the experiment because participating labs do not have to create the experimental design and/or framework. However, analysis of the results is more difficult because experimental designs and frameworks are riddled with assumptions. To combat this problem, organizers of the PatMain joint replication setup a website, through which they published details about the framework (very helpful). Nevertheless—and despite receiving clarifications—we still had to extensively explore the PatMain framework to root out underlying assumptions.

The hours we spent exploring the framework, however, were not without reward. We have discovered multiple threats to validity, some of which likely relate to the original experiment. Thus the joint replication process encourages peer review of experimental instruments and procedures—something which almost never happens outside of the publication review process.

In general, the joint replication methodology not only encourages multiple replications on the same topic, but within the same window of time, thus allowing a joint analysis to be performed while details are fresh. Also, joint replications are likely to (collectively) incorporate many more subjects than a stand-alone experiment could, and as we discussed previously, joint replications help to overcome biases by distributing the investigation over multiple labs. Also (in the case of the *strict* variant), it is much easier to tear apart an experimental design created by someone else. Thus a significant benefit of joint replications is the fostering of transparency—and by extension, collaboration—in the research process.

2.8 Conclusions

This study is part of the first ever (to our knowledge) large-scale distributed experiment replication performed in Software Engineering. It is, therefore, of particular interest from both methodological and practical perspectives. From a methodological perspective, this study calls for new ways of coordinating the research community, to agree on fundamental concepts, to overcome biases, and to more adequately cover the research space for a particular line of inquiry—issues which the joint replication process is attempting to address. From a practical perspective this replication demonstrates (in conjunction with the original experiment and first replication) a need for deeper more meticulous studies on the topic of design patterns (as well as on any topic related to programmer performance)—studies designed to search out contextual information sufficient to obtain stable results. We also find that this search process requires additional replications and synthesis of data across replications, to which end we need publishing standards and tools that foster greater transparency in the research process. For example, guidelines for lab packages (as well as online tools to manage them) need to be established, and publishing standards need to become sufficiently flexible to permit the inclusion of supplemental appendices to facilitate replication, even when such appendices exceed traditionally established page limits.[9]

[9] In a modern era of digital libraries and electronic publication of academic papers, a fixed page limit feels like an anachronistic throwback to an age of printed proceedings with a fixed cost per page.

Chapter 3

A Multi-Site Joint Replication of a Design Patterns Experiment using Moderator Variables to Generalize across Contexts

Context. Several empirical studies have explored the benefits of software design patterns, but their collective results are highly inconsistent. Resolving the inconsistencies requires investigating moderators—i.e., variables that cause an effect to differ across contexts. **Objectives.** Replicate a design patterns experiment at multiple sites; assess the heterogeneity of the results; investigate potential moderators; address the original hypotheses; generalize the results across prior studies. **Methods.** We perform a close replication of an experiment investigating the impact (in terms of time and quality) of design patterns (Decorator and Abstract Factory) on software maintenance. The experiment was replicated once previously, with divergent results. We execute our replication using a new method for performing distributed replications based on closely coordinated, small-scale instances ("joint replication"). We perform two analyses: 1) a post-hoc analysis of moderators, based on frequentist and Bayesian statistics; 2) an a priori analysis of the original hypotheses, based on frequentist statistics. **Results.** The main effect differs across the previous instances of the experiment and across the sites in our distributed replication. Our analysis of moderators (including developer experience and pattern knowledge) resolves the differences sufficiently to allow for cross-context conclusions. The conclusions represent 126 participants from five universities and twelve software companies. **Conclusions.** The Decorator pattern is found to be preferable to a simpler solution during maintenance, as long as the developer has at least some prior knowledge of the pattern. For Abstract Factory, the simpler solution is found to be mostly

equivalent to the pattern solution. Abstract Factory is shown to require a higher level of knowledge and/or experience than Decorator for the pattern to be beneficial.

3.1 Introduction

Software practitioners have ascribed numerous benefits to the use of design patterns [20, 35, 37, 73, 77]. For instance, by reducing code coupling, many patterns enable developers to add functionality without modifying existing code. Patterns also simplify communication by providing standard terminology for complex concepts. Further, as standardized representations, patterns can facilitate architectural reuse, aid inexperienced developers, improve program comprehension, and encourage best practices. Ultimately, design patterns are thought to reduce development time, increase software quality, and reduce maintenance costs—but do they really? And if they do, then under what circumstances are the various benefits realized?

Since Gamma et al.'s 1995 book [77], numerous empirical studies have explored design pattern claims (e.g., [10, 78, 95, 156, 168, 213]). However, these studies do not provide a consistent answer. For example: Prechelt et al. [168] conclude that sometimes software may still be easier to maintain even if design patterns provide *un*necessary flexibility. Conversely, Wendorff reports that the "uncontrolled use of patterns" caused, in their case, "severe maintenance problems" [213, p. 77]. In general, we find support both for (e.g., [95, 156, 168]) and against (e.g., [10, 78, 213]) design patterns, for each of several related attributes—modifiability, maintainability, understandability, quality, and so forth. In response to these and other contradictory findings, the authors of a recent (2012) systematic literature review conclude, "We could not identify [aside from two marginal caveats] firm support for any of the claims made for patterns in general" [222, p. 1213], [6].

Apparently, in the case of design patterns, contextual variation drives complex tradeoffs. The study of design patterns is thus confronted by a *problem of generalizability*—i.e., the tendency of many effects to be unstable across varying contexts, so that considerable knowledge of *moderator variables* (*moderators* for short) is required before generalizable statements can be made. By *moderator*, we mean any explanatory variable that interacts with another explanatory variable in predicting a response variable [183, p. 624], [13].[1]

Referring to this problem in sociology, Diesing notes, "We cannot test one hypothesis about flatworms, or students, or therapy patients. . . until we have learned many things about the interacting factors affecting their behavior in an experimental situation" [55, p. 338]. Moreover, "it has been argued that the amount of progress in any discipline can be indexed by the degree to which its theory and research have considered the role of moderators" [183, p. 624]. Thus, identifying moderators is critical to the research process, and until we have done so, we cannot trust our conclusions. Also, as Diesing notes, "replication is a test; but it is also part of a larger search and discovery process" [55, pp. 337–338]. Thus, we should not be surprised to find disagreement in early replications, but instead should view the situation as an opportunity to explore moderators.

Given the contradictory results among design pattern studies, identifying relevant moderators is prerequisite to generalizing across contexts. Further, as Diesing explains [55, pp. 335–339], explication of moderators requires empirical replication. Thus, in this paper we present a *joint replication* on design patterns, conducted at four sites, involving three countries and two continents, the contextual breadth of which is well-suited to the exploration of moderators.

By *joint replication* we mean a multi-site study, performed by multiple research teams whose efforts are coordinated, yet the researchers at each site act independently in performing their own replication. Research teams explicitly communicate about important aspects of the experiment, including adopting a common definition of the experiment. However, each team gathers participants, collects data, and performs initial data analysis separately, after which the datasets are then merged and analyzed together.[2] A multi-site, joint replication of

[1] See Section 3.6.2 for further explanation of the term *moderator*. Also note, throughout the paper, we use the terms *explanatory* and *response* instead of *independent* and *dependent* to refer to variables because, as we show, the explanatory variables are not statistically independent. For this reason, we need to study moderators.

[2] Joint replication is similar to multi-site randomized controlled trials (RCTs) in social work research. For information on multi-site RCT methods, see work by Solomon et al. [200, pp. 173–176].

this sort has never been done before in software engineering—but, as we show, it enables evaluation of moderators in greater detail than previously possible. In fact, a key benefit of joint replication is the contemporaneous collection of a broad range of contextual data [200, p. 174].

For the joint replication, we closely replicate a seminal experiment investigating the impact of design patterns on software maintenance (abbreviated as "PatMain"). The original PatMain study, performed in 1997 by Prechelt et al. [168] and published in 2001, was the second controlled experiment ever undertaken to study design patterns (the first being "PatDoc" [169]). The experiment was replicated once previously, in 2004 by Vokáč et al. [212], with divergent results. In addition to being one of the first design pattern studies, we chose PatMain because it is sufficiently small scale in terms of participant time and it involves relatively simple metrics that can be mostly collected automatically.

We conducted the joint replication as part of the 2011 *Workshop on Replication in Empirical Software Engineering Research* (RESER) [75, 124, 175]. Initially eight research teams expressed interest, of which four submitted data:

- Brigham Young University (BYU), Utah, USA
- Freie Universität Berlin (FUB), Germany
- The University of Alabama (UA), Alabama, USA
- Universidad Politécnica de Madrid (UPM), Spain

The four teams submitted brief reports to the workshop describing their individual results [103, 126, 155, 167]. This paper, which builds on those reports, contributes a combined analysis of the four datasets, including the collection of contextual data, analysis of moderators, assessment of the original hypotheses, and synthesis of the results with the prior two PatMain studies.

From this point forward, we refer to the original PatMain study (by Prechelt et al.) as *E_orig*, the first replication (by Vokáč et al.) as *E_repl*, and the joint replication (our work) as *E_joint*. Also, to reference these experiments generically, we use the term *study*, as in, "the three PatMain studies." To generically reference sub-replications of E_joint (i.e., BYU, FUB, UA, and UPM), we use the term *site*, as in "the four E_joint sites."

3.2 Objectives and Contributions

We pursued five research objectives for this study:

1. Replicate the PatMain experiment at multiple sites.
2. Assess the heterogeneity of the results.
3. Investigate potential moderators which may be inhibiting generalizability of the results.
4. Address the original PatMain hypotheses.
5. Generalize the results across PatMain studies.

In pursing these objectives, we make several contributions to the literature.

Concerning design patterns: The moderator analysis resolves significant contradictions in the results across the three PatMain studies. Thus, the final conclusions generalize across a broader set of contexts than has previously been achieved in the study of design patterns. The moderator analysis also identifies two variables as influencing the effect of design patterns and documents other variables that likely influence cross-site variance, some of which may be important in future studies.

Concerning specific research methods: The moderator analysis demonstrates the use of Bayesian statistics and post-hoc methods—both of which are used in other fields (e.g., medicine [209])—for discovering relationships that would otherwise remain undetected given only frequentist statistics and a priori methods.

Concerning general research methodology: This paper provides a working example of a research approach that is underused in software engineering, but which is necessary to resolve contradictory experimental results. The approach is based on replication, but exceeds the basic notions of validation and generalization, which rely on results reproduction. As we show, the idea of the approach is to use replication as an *exploratory* tool, rather than simply as a method to confirm results.

In Section 3.3, we describe the setup and participants of the three PatMain studies. In Sections 3.4 and 3.5, we describe the data and analysis methods for E_joint. In Section 3.6, we present results. In Section 3.7, we discuss threats to validity, and in Section 3.8, we conclude.

Supplemental materials can be downloaded from the publisher's website—including: all appendices for this paper; a lab package for E_orig (also available at [166]); raw data for E_repl (obtained from Marek Vokáč and included with permission); and a lab package for E_joint. Since *replicability* and *generalizability* are key concerns of this study, we include a considerable amount of supplementary information in the appendices. The appendices are provided to assist future replicators and/or to facilitate complete transparency; none are necessary in order to understand the conclusions and general validity of the study.

3.3 The PatMain Experiment

In this section, we describe the three PatMain studies: E_orig, E_repl, and E_joint. To facilitate traceability, we reuse the terms, acronyms, and format of the prior two studies (E_orig and E_repl) as much as possible.

3.3.1 The Original Study (E_orig)

In this section, we describe E_orig (by Prechelt et al.).

Motivation and Research Questions

Motivation for E_orig was twofold. First, the authors sought to empirically validate design pattern claims, which at the time were only anecdotally grounded. Second, they noticed a complexity tradeoff: while design patterns provide flexibility, they can make changes more difficult by complicating potential solutions. To explore this tradeoff, the authors proposed the following research question [168, p. 1135, paraphrased]: *For a given problem, if using a* *design pattern is "overkill" (i.e., the pattern provides more functionality or flexibility than necessary), will the resulting solution be more difficult to maintain than if a simplified solution were implemented instead?*

Experiment Design

E_orig included four modestly-sized (300–700 LOC), well-documented, C++ programs, implementing five Gamma et al. [77] design patterns:

- Stock Ticker (ST): *Observer*
- Boolean Formulas (BO): *Visitor*, *Composite*
- Communication Channels (CO): *Decorator*
- Graphics Library (GR): *Abstract Factory*, *Composite*

Each program was implemented in two variants: one employing design patterns (PAT), the other using a simplified, alternative design (ALT). The simplified design discarded all patterns not required for the given program. Observer, Visitor, Decorator, and Abstract Factory were completely eliminated in the ALT variants; Composite was retained in part in both the BO and GR programs. Participants were not informed of the PAT/ALT distinction; they were only told, "the experiment tests the usefulness of patterns" [168, p. 1135]. The experiment was administered on *paper* and included 2–3 tasks for each program. Some tasks required modifying code (i.e., *coding tasks*); others tested comprehension of the code (i.e., *comprehension tasks*).

The experiment began with a pre-test (PRE) involving two of the four programs (one PAT, one ALT). A patterns training course was then administered, followed by a post-test (POST), involving the remaining two programs (again one PAT, one ALT). Since design patterns were new at the time of the experiment, the training course was a key incentive to participate. The experiment lasted two days—pre-test in the morning of day 1, training in the afternoon of day 1 and morning of day 2, post-test in the afternoon of day 2. All participants received all four programs. Program and variant orderings were alternated. Experience and pattern knowledge were pre-surveyed to facilitate stratification of the random assignment to groups.

This design resulted in six explanatory variables: *program* (with levels ST, BO, CO, GR); *task* (with levels 1, 2, 3); *variant* (with levels PAT, ALT); pattern knowledge (abbreviated as *patKnow*, with levels PRE, POST); program *order* (with levels first, second); and *subjectID*. The experiment assessed two response variables: *time* (measured in minutes) and *correctness* (i.e., quality of the solution, measured on a 4-point scale: no fault, minor problem, not-so-minor problem, major problem). Hypotheses addressed the impact of *variant* and pattern knowledge (*patknow*) on *time* and *correctness*.

Participants

The 29 participants were all volunteers, software professionals from the consultancy firm sd&m in Munich, Germany. The median industry experience was 3.5 years (mean 4.1), including 2 years (mean 2.4) with object-oriented programming. Fifteen (52%) of the participants had prior experience with patterns.

3.3.2 The First Replication (E_repl)

In this section, we describe E_repl (by Vokáč et al.).

Motivation and Research Questions

The goal of E_repl was to increase "the experimental realism and, thereby, the applicability of the results" [212, p. 150]. In particular, the participants in the replication worked on computers rather than on paper. Otherwise, the authors strove for a close replication.

Experiment Design

The design of E_repl involved five key changes: 1) participants worked on computers, in a standardized environment, rather than on paper; 2) correctness was assessed on a 5-point scale (requirements misunderstood, wrong answer, right idea, almost correct, correct); 3) the original programs were adjusted to facilitate compilation; 4) the participants were selected by their consultancy firms, rather than being volunteers; and 5) the participants were paid. Otherwise, E_repl closely duplicated E_orig's design; the patterns course was even taught by the same instructor (Walter Tichy) using the same materials.

Participants

The 44 participants included 39 software professionals from 11 consultancy firms and 5 graduate students. The median industry experience was 4 years (mean 6.6), including 2 years (mean 2.4) with object-oriented programming. Seventeen (39%) of the participants had prior experience with patterns.

3.3.3 The Joint Replication (E_joint)

In this section, we describe our replication, E_joint.

Motivation and Research Questions

Three motivations prompted E_joint: 1) a joint replication had never been done before in software engineering, so we anticipated significant learning with respect to methodology; 2) we were interested to see how homogeneous the results would be across sites; and 3) we wanted to address the issue of contradictory results among design pattern studies. In general, we wanted to test a new method for performing distributed replications based on closely coordinated, small-scale instances. Like E_repl, we maintain the original research question and hypotheses.

Experiment Design

We strove for a close replication. All changes (described below) were either improvements carried over from E_repl, or they were administrative changes designed to encourage participation and to facilitate consistency across sites.

Like E_repl, the participants worked on computers, rather than on paper. We administered the experiment via a web portal, through which the participants downloaded source code and uploaded solutions. The portal recorded work times by tracking the time spent on each page. The portal also managed experiment groups and administered the pre/post-questionnaires. For detailed information on the portal, see Appendix A.

Using a web portal provided four benefits: 1) it lowered the barrier to entry for replicators; 2) it facilitated uniformity across sites; 3) it allowed participants to take the experiment on their own time, thus reducing scheduling constraints; and 4) it allowed participants to work with their own tools in their own environments. The downside is that we had less control over what the participants did during the experiment (such as take phone calls). However, few participants reported interruptions or protocol deviations, and for those that did, we apply data corrections, as described in Section 3.4.

Another change involved reducing the scope of the experiment by eliminating the training course and the post-test, thus reducing the experiment from four programs to two. The purpose of the scope reduction was, as with the web portal, to lower the barrier to participation. Thus we traded breadth of protocol for increased coverage of contextual variables and greater diversity in the population sample. Of the four programs, we excluded ST because it is the least complex and BO because the prior two studies both found the Visitor pattern to be overly difficult.[3]

[3] For example, of E_repl, Vokáč et al. comment, "... Visitor was so difficult that even after a course that gave the instructor excellent feedback (grade better than 4 out 5), most subjects either ignored the pattern or were confused by it." [212, p. 172].

Additionally, note that pattern difficulty (or complexity) is not the same concept as that of "overkill" from the PatMain research question described in Section 3.3.1. "Overkill" refers to a pattern's complexity *relative* to the problem it aims to solve, rather than to its absolute complexity. Thus, although the Visitor pattern is structurally more complex than Decorator and Abstract Factory, it is not necessarily more suited to answering

In lieu of the training course, we assessed pattern knowledge via a pre-questionnaire. As with the other materials of our replication, we adopted the pattern knowledge questionnaire from the original experiment (where it had been used to facilitate stratification of the random assignment to groups). The questionnaire required participants to estimate their knowledge of 17 patterns on a 7-point ordinal scale:

- 1=*never heard of it.*
- 2=*have only heard of it.*
- 3=*understand it roughly.*
- 4=*understand it well.*
- 5=*understand it well and have worked with it once.*
- 6=*understand it well and have worked with it two or three times.*
- 7=*understand it well and have worked with it many times.*

Except for the Reactor pattern, which was originally defined by Schmidt et al. [185, pp. 179–214], the 17 patterns were all standard Gamma et al. [77] design patterns (for a complete list, see Appendix F). Admittedly, if participants are too homogeneous, a survey may fail to detect an effect even when one exists. On the other hand, if participants are initially too knowledgeable, a training course may also fail to detect an effect. Thus both approaches have limitations.

Lutz Prechelt translated the materials of the original experiment from German to English, which translation was used at all four sites. Lutz also modified the questionnaires to collect additional data. To test the translation, we administered the experiment to three native English-speaking computer science students not previously affiliated with the study; the test participants had no problem understanding the instructions and questionnaires. For

questions about overkill. By eliminating the visitor pattern we aimed to tackle the question of overkill with respect to Decorator and Abstract Factory first, after which we hoped to deal more easily with the Visitor pattern.

discussion of how the use of English may have impacted the sites differently, see Appendix Q (specifically the subsections on language barriers and clarity of task instructions).

Other changes included: 1) The web portal offered three language options: C++, Java, and C# (Martin Liesenberg and Christian Bird provided the Java and C# translations of the original C++ programs). Despite having three options, all participants worked in Java. At BYU and UA they were instructed to do so; at FUB and UPM they did so by preference. 2) We did not stratify the random assignment to groups; instead, we control for developer experience and pattern knowledge via covariates in the statistical analysis. Group assignments were made by the web portal based on randomly assigned IDs ($group\# = ID \% 4$):

- Group 0: GR-PAT, CO-ALT (14 participants)
- Group 1: CO-ALT, GR-PAT (12 participants)
- Group 2: GR-ALT, CO-PAT (12 participants)
- Group 3: CO-PAT, GR-ALT (15 participants)[4]

Our protocol preserves all variables from E_orig with only minor changes. Concerning explanatory variables: we limit *program* to two levels (CO, GR); we assess pattern knowledge (*patknow*) via a survey instead of a training course; and we add explanatory variables to represent developer experience (*devExp*) and *site* (BYU, FUB, UA, UPM). All other variables (*task*, *variant*, *order*, and *subjectID*) are as described previously for E_orig. Concerning response variables: *time* is measured in seconds instead of minutes, and *correctness* is measured on E_repl's 5-point scale (described in Section 3.4). Additionally, despite the standardized web portal, the four sub-replications still differed in a few minor respects—e.g., BYU's experiment spanned 3 weeks, whereas UA's was completed within 3 days. For a complete list of the differences, see Appendix B.

[4] Treatment groups are modestly imbalanced due to several no-shows and eight cases of unusable data, as discussed in Section 3.4. For group sizes listed by site, see the data file in the lab package.

Participants

The four sites independently solicited participants. Participants were expected to have a good working knowledge of C++, C#, or Java. Design pattern knowledge was not strictly required. All four teams enlisted students. The student demographic is useful in the case of PatMain because both prior studies employed professionals. As we discuss in Section 3.6, our use of students leads to several key insights.

The 53 participants were all solicited from software engineering courses. All were computer science majors or equivalent—including 27 undergraduate and 26 graduate (MS or PhD) students. The median industry experience was 0 years (mean 1.5). More than half of the participants reported understanding (at least roughly) most of the patterns surveyed. However, only one pattern (Observer) was reported by a majority of participants as having ever been used. For most of the participants, their implementation experience with patterns was the result of coursework. Thus, the participants had broad *exposure* to patterns, but little *practical* (and almost no industry) experience with them.

Concerning sites, we find three notable differences: 1) the BYU and FUB participants were mostly undergraduates, whereas the UA and UPM participants were entirely graduate students; 2) the BYU participants reported having used more programming languages than those at any of the other three sites; and 3) the UA and FUB participants reported greater *exposure* to patterns than the BYU and UPM participants. For further details on these demographics, see Appendix C.

3.4 Data and Metrics

In this section, we describe the joint dataset. We also provide additional information about explanatory and response variables. Variables not appearing in this section are described sufficiently above. For summary statistics on experiment variables, see Appendix D.

3.4.1 Joint Dataset

Of the 61 participants to take the experiment, we had to discard all data for 8. For a list of the affected participants, see Appendix E. Problems included: failure to complete any of the tasks and failure to adequately record breaks. Two participants also submitted the same data for both programs, and one quit after completing only the CO program. We ended up with data from 52 participants for the CO program and 51 for the GR program (53 total).

We provide an annotated copy of the joint dataset in the lab package. We describe the dataset schema in Appendix F and the annotation process in Appendix G. Annotation basically involved scanning by column for outliers and then by row for participants who deviated from the instructions. Additionally, of the 91 fields in the dataset, we exclude 53 from statistical analysis. Some fields we exclude because they represent meta- or qualitative data. Other fields we exclude because, given only 53 participants, we must limit the number of parameters we estimate in the statistical models (to avoid over-fitting, multicollinearity, loss of precision, etc. [174]). We also need to reduce the complexity of the models in order to enable theoretical interpretation of the results, which interpretation is necessary in order to generalize the findings across all three PatMain studies. For a list of the unused variables, some of which may be useful in other studies, see Appendix H.

Also note, E_orig defined three tasks for CO, but only two each for ST, BO, and GR. Consequently, the authors of E_repl combined CO tasks 2 and 3 to produce "a more symmetrical experimental design" [212, p. 179]. This approach is reasonable because CO tasks 2 and 3 are similar and address similar hypotheses. Like E_repl, we also combine CO tasks 2 and 3 by summing the task times and averaging the correctness scores. *From this point forward, we treat CO tasks 2 and 3 as a single task, which we collectively refer to as "CO task 2".*

3.4.2 Developer Experience (*devExp*)

To include in our models all of the developer experience variables we surveyed, we risk diluting statistical power [174, p. 347]. Some of the variables are also highly correlated (well above 0.7), thus leading to multicollinearity issues. Discarding data is not ideal, so instead we take an aggregate approach. By combining metrics, we conserve degrees of freedom, avoid multicollinearity, and hopefully "average out" measurement error [204].

To compose the aggregate metric, we average 6 component metrics: languages-used-lifetime, languages-used-often, LOC-lifetime, LOC-Java, programming-hours-per-week, and self-assessed-programming-skill. Prior to averaging, we log transform and scale each variable as needed to mitigate outliers and to prevent any single metric from dominating the average. For a detailed description of the process, see Appendix I. The result is a continuous variable ranging from 1 to 7, scaled to match the range of pattern knowledge (described below), where 7 represents high experience.

3.4.3 Pattern Knowledge (*patKnow*)

Participants estimated their knowledge of 17 design patterns on a 7-point ordinal scale (for a description of the scale, Section 3.3.3). Like developer experience, and for the same reasons, we average the pattern knowledge scores to produce a single metric. Although averaging treats the ordinal scale as an interval scale, it should reasonably differentiate the participants.

3.4.4 Time

The web portal's page timings indicate that some participants spent considerable time on the download pages. Likely, those participants began working (e.g., reading source code) before proceeding to the task description pages. *Consequently, we sum download, work, and upload page timings for all coding tasks (CO and GR task 1s).*

3.4.5 Correctness

To ensure consistency, we had all tasks graded by the same two people, neither of whom were previously affiliated with this study. The two graders were both graduate researchers with professional experience in software development, including one in software test. For the coding tasks, the graders worked together in a pair-programming style arrangement. They initially reviewed the solutions for each task, from which they decided to use (essentially) the same 5-point scale used in E_repl:

1. *Requirements misunderstood* (0%)—the solution is completely wrong; it appears that the participant did not understand the requirements.
2. *Wrong answer* (25%)—the participant appears to have understood the requirements, but did not produce the correct solution, even conceptually.
3. *Right idea* (50%)—the solution conceptually addresses the requirements, but is incomplete or contains an error and does not compile.
4. *Almost correct* (75%)—the solution conceptually addresses the requirements and compiles, but is incomplete or contains a functional error.
5. *Correct* (100%)—the solution is completely correct; it compiles and meets the stated requirements.

For the short-answer tasks, the graders chose a binary rubric (*incorrect*=0%, correct=100%). They chose the binary rubric because, in their words, "people did not appear to be guessing; they seemed to either know the answer or not." Each grader then evaluated half of the participants, after which they compared results to ensure consistency. The graders graded holistically, considering everything a participant said, rather than simply searching for a specific answer.

3.5 Analysis Methods

To address the original PatMain hypotheses, we use frequentist models, which we designed prior to viewing the data. For the post-hoc analysis of moderators, however, we use both frequentist and Bayesian models. Below, we explain our reasons for using Bayesian models. We then describe the frequentist and Bayesian models in detail. Finally, we explain how to interpret the results for each type of model. Source code (R and SAS) for all models is provided in the lab package.

3.5.1 Why Bayesian Models?

First, the Bayesian models allow us to directly compare probabilities for competing hypotheses, which means we can form conclusions even when statistical power is low. In frequentist statistics, a p-value is the probability of obtaining data at least as extreme as those observed, assuming a null hypothesis is true. Conversely, Bayesian models yield *posterior* probabilities. A posterior probability is the conditional probability that a hypothesis is true, given the data. Being a probability about the truth of a hypothesis, rather than about the likelihood of the data, we can directly compare posterior probabilities to determine which hypotheses are most likely [54].

The ability to compare probabilities is especially useful for the moderator analysis, which requires modeling high-order interactions. First, adding interactions to any model dampens statistical power by spreading the data over more parameters [174]. Second, the frequentist models yield insignificant, *but very large* effect estimates for many of the interactions we test. Thus, the frequentist results for the moderator analysis are inconclusive; the data are simply too few relative to the variance and model sizes. Conversely, using Bayesian methods, even though the statistical power is minimal, we can still identify likely moderators by comparing posterior probabilities for the appropriate hypotheses.

Second, the Bayesian models provide samples from the joint posterior distribution, which means we can use fewer parameters to estimate the same set of interactions, thus *conserving statistical power.* We estimate the Bayesian models using Gibbs sampling [54], which generates samples from the joint posterior distribution of all parameters in the model. Given posterior samples, we can use marginalization to compute, from a single high-order interaction, results for all lower-order interactions and component terms [54]. Conversely, in mixed models analysis, lower-order interactions and terms must be estimated by separate parameters [174].

Third, the Bayesian models are subject to a different set of statistical assumptions, which means they provide validation for the freqentist models (and vice versa) [54, 69, 174].

3.5.2 Frequentist Models

Table 3.1 summarizes the frequentist models. For this analysis, we use *linear mixed models* [144], an extension of multiple linear regression, which adds the ability to represent blocking (or grouping) variables as random effects. Because the CO and GR programs differ, we analyze their results in separate models. We also run separate models for each response variable. To control for participants who achieve higher correctness simply by working longer, we include *time* as a covariate in the *correctness* models (and vice versa). Also, like E_orig and E_repl, we model (via interaction effects) the impact that *patKnow* has on *variant*, which is necessary to address the original hypotheses.

To normalize the data, we log transform *time*—after which the data conform to the standard assumptions of mixed models analysis, including normality, multicollinearity, and heteroscedasticity. For a detailed assessment of model assumptions, see Appendix J. To maximize statistical power, we tune each model using a standard covariate pruning technique; the technique is essentially backward stepwise regression, but modified to avoid fishing for significance [174, p. 345]. For specific details, see Appendix K. All p-values are two-sided.

Table 3.1: Frequentist statistical models.

Model	Program	Response Variable	Explanatory Variables
CO_time	CO	time_ln (*cont*: ≥0)*	see below
CO_correctness	CO	correctness (*cont*: 0–100)	see below
GR_time	GR	time_ln (*cont*: ≥0)*	see below
GR_correctness	GR	correctness (*cont*: 0–100)	see below

Blocking Variable (random effect)	
subjectID	Accounts for multiple observations per participant.
Covariates (fixed effects)[†]	
order	Program order (*cat*: 1=first, 2=second)
task	Program task (*cat*: 1=coding, 2=comprehension)
site	Sub-replication (*cat*: BYU, FUB, UA, UPM)
devExp	Developer experience (*cont*: 1–7)
patKnow	Pattern knowledge (*cont*: 1–7)
time_ln* or correctness	In a given model, we use whichever variable is not the response variable. Controls for correlations between time and correctness.[‡]
Main Effect and Interactions (fixed effects)	
variant	Program variant (*cat*: PAT, ALT)

patKnow × variant, variant × task,
patKnow × task, patKnow × variant × task

cont = continuous variable; *cat* = categorical variable.
*Normalized via log transformation.
[†] We tested one other covariate, representing java familiarity, but found it to have almost no impact. See Appendix I for details.
[‡] E.g., achieving higher correctness simply by working longer.

3.5.3 Bayesian Models

For the Bayesian analysis, we construct additive-effects models as shown by Felt [69]. However, instead of using a single variance parameter, we include four in each model, one for each task. We estimate the task variances separately because some of the tasks require more time than others, and longer tasks typically manifest greater variance.

Additionally, we represent all explanatory variables as categorical effects. Doing so avoids linearity assumptions and, in that regard, provides validation for the frequentist models. As a drawback, using categorical effects necessitates more parameters in the models than would be needed to represent simple linear relationships. However, given the exploratory purpose of the Bayesian models, we do not want to rely on linearity assumptions in this case.

We divide each continuous variable's range into two categories, *low* and *high*, representing roughly the bottom and top halves of the data, respectively. For a complete list of the divisions, see Appendix L. Note that we could use more than two categories per variable, thus allowing the models to fit more complex relationships. However, adding additional categories causes a multiplicative increase in the number of parameters required to model interactions, which ultimately spreads the data too thin to obtain useful results. In fact, with even three categories per variable, most of the interactions involve at least one level that is completely unrepresented in the data. Conversely, with two categories per variable, all levels of all interactions are adequately represented.

Like the frequentist analysis, we run separate models for each response variable. However, because the Bayesian models allow us to easily estimate separate variances for each task, we model the CO and GR programs together. Since *time* is skewed high and cannot be negative, we model it as a gamma distribution; since *correctness* is a percentage, we model it as a beta distribution. The data conform to the assumptions inherent in the type of models we construct, including multicollinearity and heteroscedasticity. For a detailed assessment of model assumptions, see Appendix J.

For each response variable, *time* and *correctness*, we run 6 models (denoted T1–T6 and C1–C6, respectively). All models include the same set of variables, matching those for the frequentist analysis (shown in Table 3.1), plus an additional variable representing *program*. Each model also includes one interaction effect. If a variable is included in the interaction, then it is not included elsewhere in the model.[5] The models only differ by which variables are included in the interaction. Table 3.2 shows, for each model, the interaction being tested.

[5] Unlike frequentist statistics, Bayesian estimates for low-order interactions and terms can be computed via marginalization from high-order interactions. Thus, a variable need not be modeled both within an interaction and as a standalone effect.

Table 3.2: Bayesian model interactions.

Model(s)	Interaction
T1, C1	program × variant × task × site
T2, C2	program × variant × task × patKnow
T3, C3	program × variant × task × devExp
T4	program × variant × task × correctness
C4	program × variant × task × time
T5, C5	program × variant × task
T6, C6	program × variant

Concerning prior distributions, we enlisted a qualified external researcher to estimate all priors using data from E_orig. We gave our helper only two constraints (both suggestions of Felt [69]): First, we instructed him to center all priors—with the exception of the variances and base offset—at zero, thus assuming no effect by default (i.e., the null hypothesis). Second, we instructed him to select broad priors in order to minimize their impact on the results. Broad priors are ideal for post-hoc analysis, for which the objective is to generate data-driven hypotheses [69]. For a list of the exact priors, see Appendix M.

3.5.4 Results Interpretation—Frequentist vs. Bayesian

In frequentist statistics, p-values are significant when small—i.e., to reject a null hypothesis, the data must be unlikely under the assumption of that hypothesis. For posterior probabilities, however, large values are significant—e.g., 0.95 represents a 95% chance that the associated hypothesis is true, given the data. Further, we typically require very small margins of error in frequentist statistics (e.g., $\alpha = 0.05$); otherwise the results are inconclusive. Conversely, for posterior probabilities, significance depends on the context of the problem. For example, in our analysis, a posterior probability of 0.75 means we expect the associated hypothesis to hold in 75% of similar cases. Under such an interpretation, even relatively low probabilities can be meaningful.

3.5.5 Moderator Categories (low/high)

Some of the moderators we explore are continuous variables. However, in the results discussion, we generalize them in terms of *low* and *high* categories. For the frequentist models, which are linear, *low* and *high* correspond to the min and max values found in the data for the given variable. We use min/max data values rather than the theoretical limits of the given variable in order to avoid extrapolation (e.g., in the case of *patKnow*, high=5.412 rather than 7). For the Bayesian models, which make no linearity assumptions, the terms *low* and *high* roughly correspond to the bottom and top halves of the data for each variable, as described previously.

Concerning pattern knowledge, *low* and *high* correspond to the prior studies' designations of PRE and POST—meaning before and after the patterns training course. By making this comparison, we are not asserting that PRE and POST map directly onto E_joint's categories of *low* and *high*—especially since the participants' prior pattern knowledge differed across all three studies. Instead, assuming a linear effect, we expect the transition from PRE to POST to be comparable to that from *low* to *high*.

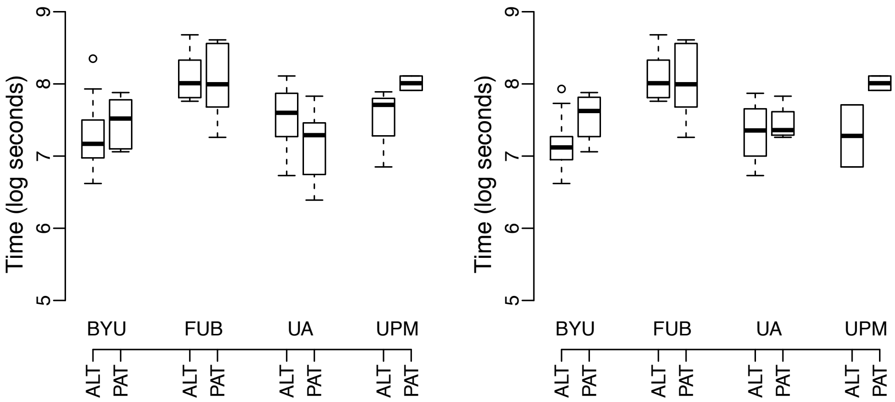
3.6 Analysis Results

In this section, we assess the heterogeneity of the results for E_joint, investigate potential moderators, and address the original hypotheses. To facilitate future meta-analysis, we provide a concise listing of all statistical results in Appendices X and Y.

3.6.1 Assessment of Heterogeneity

In this section, we show that E_joint manifests the problem of generalizability—i.e., the main effect varies across sites. We demonstrate this problem via both the frequentist and the Bayesian models. We also explore data filtering as a method to mitigate the problem. The purpose of this section is to validate that moderator analysis is needed to make sense of E_joint's dataset.

Figure 3.1: Time data for CO task 1, showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.



(a) Unfiltered data **(b)** Filtered data

The Problem

Figure 3.1a depicts the main effect, program *variant*, compared across the four sites (before filtering). Notice that the sites significantly differ—in two cases PAT takes more time than ALT, in another case no effect can be seen, and in the remaining case the effect is reversed. We find similar divergences on all tasks, for both *time* and *correctness* (the remaining plots are shown in Appendix O). *Thus, with respect to variant, the results do not generalize well across the four sites.*

In the (unfiltered) frequentist models, this problem shows up as *site* being considerably more significant than *variant* (see Table 3.3). Similarly, in the Bayesian models, we find (for all tasks) that the effect of *variant* is more significant at individual sites than when generalized across sites (uT1,C1:46–54,791–826).[6] For example, on CO task 2, the marginal

[6] We use references of this form to map specific assertions back to the Bayesian probabilities on which they are based. The references are necessary because the Bayesian analysis produces more data than we have room to list in the main paper. The references provide an audit trail, as well as a more complete working example for future replicators. The paper is written such that the references can be ignored. The reference notation is as follows: u and f signify unfiltered and filtered data—i.e., Tables Y.1 and Y.2 in Appendix Y. T# and C# indicate a specific *time* or *correctness* model—i.e., one of the models T1–T6 or C1–C6, which are represented as columns in the tables. Lastly, a colon indicates a list of row numbers—e.g., uT1:12 signifies Table Y.1 (unfiltered data), column T1, row 12. Abbreviations resulting from this notation include: uT1, uC1, fT1, fC1, uT2, uC2, fT2, fC2, uT3, uC3, fT3, fC3, etc.

Table 3.3: Frequentist model p-values for *site* and *variant*. p-values less than or equal to 0.05 are bolded.

	Unfiltered		Filtered	
Model	*site*	*variant*	*site*	*variant*
CO_time	**<0.001**	0.925	**<0.001**	**0.019****
CO_correctness	**0.003**	0.095	**0.008**	0.523
GR_time	**0.019**	**0.016**	**<0.001**	**0.025**
GR_correctness	NS	0.322	NS	0.245

NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix K.
*Min p-value for *variant* within the *patKnow* × *variant* interaction.

(i.e., generalized) probability that PAT took longer than ALT is only 0.58—compared to, for example, 0.86 at UPM (uT1:51,807).

Clearly, cross-site variance is inhibiting isolation of the main effect—otherwise, aggregating data across sites would yield an increase in statistical significance. Thus, in addition to being a replication of PatMain, E_joint evokes the broader problem of generalizability that confronts the PatMain series of studies. Moreover, E_joint reproduces that problem within the context of a single, controlled experiment. Thus, much of the variance observed across sites can likely be attributed to meaningful variables rather than to experimental artifacts. This setup is ideal for studying moderators.

E_repl's Use of Filtering

E_repl, which involved participants from 11 companies, encountered problems with variance similar to E_joint. As Marek Vokáč describes, "[Our participants] came from some of the major (even international) consultancy companies, and they were paid quite well for their efforts" (email, Oct. 14, 2012). Nevertheless, the analysis required "an expert from the statistics section of the Math faculty" because "the 'usual' methods were not enough to extract a good signal from the fairly noisy data" (email, Oct. 16, 2012). Conversely, all of E_orig's participants came from a single company, and variance was not a significant concern in that case.

E_repl used two types of data filtering to reduce variance, which we refer to as *observation* and *participant* filtering. Observation filtering—i.e., discarding all data points with a *correctness* score below 75% (*time* models only)—is too course grained to effectively reduce cross-site variance in E_joint (for further details, see Appendix N). Participant filtering, however, does account for at least some of the cross-site variance.

Participant Filtering

In E_repl, "Four subjects had consistently low-quality solutions. Inspection. . . revealed that their C++ proficiency was so low that it would significantly mask any other effect" [212, p. 162]. Consequently, Vokáč et al. discarded all data for these underqualified participants, concluding that no bias was introduced since the PAT and ALT solutions were equally affected. However, Vokáč et al. did not specify a process for identifying underqualified participants.

Therefore, to filter participants in E_joint, we determined a filtering threshold based on the participants' average task *correctness*. Averaging across tasks factors out *variant*, such that the choice of threshold does not bias the final results. To help ensure objectivity, we enlisted four *independent* reviewers. The four reviewers were all software engineering researchers, including two not affiliated with this study. Based on a plot of the averages, the reviewers unanimously selected a threshold of 25 percentage points. At that threshold, 10 participants were excluded (6 UA, 3 BYU, 1 UPM, 0 FUB). For a list of the 10 participants and a copy of the plot used, see Appendix O. All of the excluded participants received a zero on all, or nearly all tasks. A zero score is only given when *the solution is completely wrong, and it appears that the participant did not understand the requirements.* Thus the excluded participants were likely underqualified and/or insufficiently motivated.

Box plots reveal that the filtering decreased cross-site variance. For example, in Figure 3.1b, UA's filtered results no longer contradict those of BYU and UPM. Similarly, the significance of *variant* increased considerably in the CO_time model (p-value decreased from

Table 3.4: Frequentist model p-values for *devExp* and *patKnow*. p-values less than or equal to 0.05 are bolded.

	Unfiltered		Filtered	
Model	*devExp*[*]	*patKnow*	*devExp*[*]	*patKnow*
CO_time	0.076	NS	NS	**0.007**[†]
CO_correctness	0.101	NS	NS	**0.049**
GR_time	<**0.001**	NS	NS	**0.001**
GR_correctness	NS	NS	NS	NS

NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix K.

[*] E_repl also tested an experience covariate—a "pre-qualification score" [212, p. 157]—which was an aggregate metric similar to *devExp*. It was not found to be significant, but that result was obtained only after filtering low-scoring participants.

[†] Min p-value for *patKnow* within the *patKnow* × *variant* interaction.

0.93 to 0.02; see Table 3.3). We also see some improvements in the box plots for GR task 2 (shown in Appendix O).

Moreover, the filtered models make more sense. First, *patKnow* becomes significant in three of the four frequentist models (see Table 3.4), consistent with the prior two studies. Second, the *patKnow* × *variant* interaction also becomes significant (in the CO_time model, p-value = 0.029), which is consistent with the notion that pattern knowledge should be more meaningful for the PAT variant than for ALT.

Ultimately, the filtering increases statistical precision without significantly altering the main effect estimates. However, despite the improvements, we also see in Table 3.3 that the GR_time model does not significantly change, nor do either of the *correctness* models. Further, *site* remains significant in the same models as before, and the Bayesian results (fT1,C1:46–54,791–826) still show *variant* to be more significant within individual sites than across sites—though the disparity is reduced. *Thus, although participant filtering reduces cross-site variance, it does not fully explain such variation.*

3.6.2 Moderator Analysis

In this section, we investigate *moderators*. We use both the frequentist and the Bayesian models for this discussion. By *moderator*, we mean any explanatory variable that interacts with another explanatory variable in predicting a response variable [183, p. 624], [13]. For one variable to "moderate" another does *not* mean that it dampens the other's effect—rather, it means that an interaction exists, such that the latter's effect varies in response to the former. If unaccounted for, the variance induced by a moderator can mask the effect of the variable with which it interacts.

Since moderator analysis is post-hoc, it inflates the chances of a type 1 error. In other words, testing relationships that were not specified a priori increases the risk of incorrectly concluding an effect exists. *Thus, our conclusions in this section are data-driven conjectures, which need to be further investigated.* We also recognize that *correctness* and *time* are response variables, not moderators. However, in their role as covariates, they can reveal insights about moderators, so we include them in the analysis. Lastly, *all conclusions from this point forward are limited to: 1) the patterns tested in the CO/GR programs (Decorator and Abstract Factory), 2) maintenance activities, 3) maintainers that were not the original implementers, and 4) programs for which the full functionality of the patterns was not initially needed.*

Moderator Selection

By the term *moderator*, we are referring to a phenomenon inherent in nature—i.e., *one variable moderates another's effect on some outcome independent of whether either is experimentally measured or statistically modeled.* Any variable (or more loosely, factor or influence), previously known or unknown, measured or not, can be a moderator. Our goal is to discover the most influential moderators, sufficient to explain cross-site variance and to produce generalized conclusions. Thus, in this analysis, we investigate as many potential moderators as possible, *including several that were not measured as part of the experiment.*

To identify potential moderators for analysis, we used a relaxed, grounded-theory process involving coding, memoing, and the forming of categories (similar to that described by Charmaz [41]). For source data we used the workshop reports for the four sub-replications of E_joint, the reports for E_orig and E_repl, and email conversations between E_joint researchers discussing their experiences. The process yielded a list of candidate variables, from which we selected for analysis those that met the following two criteria: 1) the variable seemed theoretically meaningful, and 2) we had data available that could reasonably represent the variable. *Many of the variables we identified were not considered prior to this analysis, so they do not appear in Table 3.1.*

The variables selected for analysis include: student vs. professional status, *devExp*, *patKnow*, motivation, task difficulty, *correctness/time*, program *order*, perceived time limits, cultural variation, IDE preferences, language barriers, clarity of task instructions, and compilation/testing expectations.

For measured variables, we statistically assess the extent to which each moderates the main effect (see Table 3.5). For other variables, we glean what we can from qualitative data. We document all variables explored, even those that turned out to be innocuous.

How to Read Table 3.5

Table 3.5 shows results for the Bayesian assessment of moderators. Rows represent potential moderators; columns represent interactions. The m-columns represent $\mathit{variant} \times \mathit{program} \times \mathit{task} \times \mathit{mod}$ interactions, where *mod* is the moderator for the given row; the $\neg m$-columns represent $\mathit{variant} \times \mathit{program} \times \mathit{task}$ interactions, where the given moderator has been marginalized out. The $\neg m$-columns provide a baseline against which to compare the m-columns. The baseline values within each $\neg m$-column differ because the moderators were tested in separate models. The data were simply too few to test all of the interactions via a single model. Thus, we can quantitatively assess which variables are likely moderators, but we cannot quantitatively assess which moderators are most influential.

Table 3.5: Bayesian interaction results—moderator assessment. Probabilities are shown in black; effect estimates in gray. Probabilities greater than 0.75 are bolded.*

time Models	Unfiltered		Filtered		*correctness* Models	Unfiltered		Filtered	
	$\neg m$	m	$\neg m$	m		$\neg m$	m	$\neg m$	m
site	0.62	**0.86**	0.67	**0.88**	*site*	0.72	**0.81**	0.73	**0.87**
(T1)	196	653	309	842	(C1)	10.5	18.2	11.7	21.0
patKnow	0.64	0.67	0.71	**0.85**	*patKnow*	0.71	**0.92**	0.73	**0.86**
(T2)	153	169	298	504	(C2)	8.7	17.3	8.5	13.7
devExp	0.66	**0.90**	**0.76**	**0.89**	*devExp*	0.70	**0.85**	0.67	**0.89**
(T3)	237	561	398	600	(C3)	7.5	12.7	6.9	15.8
correctness	0.68	**0.83**	**0.81**	**0.88**	*time*	**0.81**	**0.93**	**0.82**	**0.85**
(T4)	218	408	404	821	(C4)	10.0	18.0	10.9	12.8

*See Section 3.6.2 for a description of how to read and interpret this table. For a visualization of the data, see Appendix P.

Each probability in Table 3.5 represents the max significance of *variant* to occur at any level of the associated interaction. If a variable moderates *variant*, then by definition, *variant*'s effect must vary across the levels of the moderator. Consequently, the significance of *variant* will always be greater for at least one level of the moderator than it is when the moderator is marginalized out. Thus, if a moderator is significant, $m \gg \neg m$. For example, the top-left probability in Table 3.5, 0.62, is the max significance of *variant* to occur among the four levels of *program* × *task* (unfiltered data, model T1). The next probability to the right, 0.86, is the max significance of *variant* to occur among the 16 levels of *program* × *task* × *site*. Since the max significance of *variant* substantially increases when *site* is added to the interaction, *site* represents a likely moderator.[7]

As mentioned previously, *site* does not help us to generalize because it is not a software-domain variable; thus, we need to identify other, more meaningful moderators. Also, notice that the results for *site* in Table 3.5 are consistent with the assessment of heterogeneity

[7] By *significance*, we mean the maximum of $p(\text{ALT}>\text{PAT})$ versus $p(\text{ALT}<\text{PAT})$, where p is the posterior probability. Using the greater of the two is appropriate because, for binary comparisons, non-significance is at 0.5. Thus, values such as 0.25 and 0.75 are equally significant. Further, the directionality of the ALT/PAT comparison is irrelevant. All that matters is the degree to which each moderator increases the significance of *variant* when added to the interaction.

in Section 3.6.1. Thus, Table 3.5 is a shorthand method for showing a moderator's influence. For each variable listed in the table, we could have provided a lengthy assessment similar to that given for *site* in Section 3.6.1.

Finally, each effect estimate in Table 3.5 represents the magnitude of the difference between ALT and PAT (in seconds or percentage points) associated with the corresponding probability. We provide these estimates to show the practical significance of the moderators.[8]

Students vs. Professionals

We consider the student-professional distinction first because it represents a significant experimental difference between the three PatMain studies. It is possible that E_joint's use of students caused its results to differ from those of the prior two studies. For instance, the participant filtering, which brought E_joint's results into greater alignment with the prior studies, may have effectively distilled from the student data a more professional-like sample. After all, the median experience in E_joint was much lower than in E_orig and E_repl (0 years, compared to 3.5 and 4), and accordingly, E_joint had to filter more participants (10/53, compared to 0/29 and 4/44).

Likely, our use of students did involve a higher percentage of underqualified participants. However, the prior two studies both used professionals, with similar experience, and yet they still differed in their results. Also, as previously mentioned, E_repl encountered variance problems similar to E_joint, whereas E_orig did not. *Thus, the student-professional distinction is overly simplistic and does not align well with the cross-site variance; accordingly, it is not an effective variable by which to generalize the PatMain results.* From this point forward, we explore other variables, several of which likely underlie the student-professional distinction.

[8] As described in Footnote 7, the directionality of the ALT/PAT comparison is irrelevant when assessing a moderator's influence. Thus, we use effect magnitudes—i.e., we drop negative signs.

Developer Experience

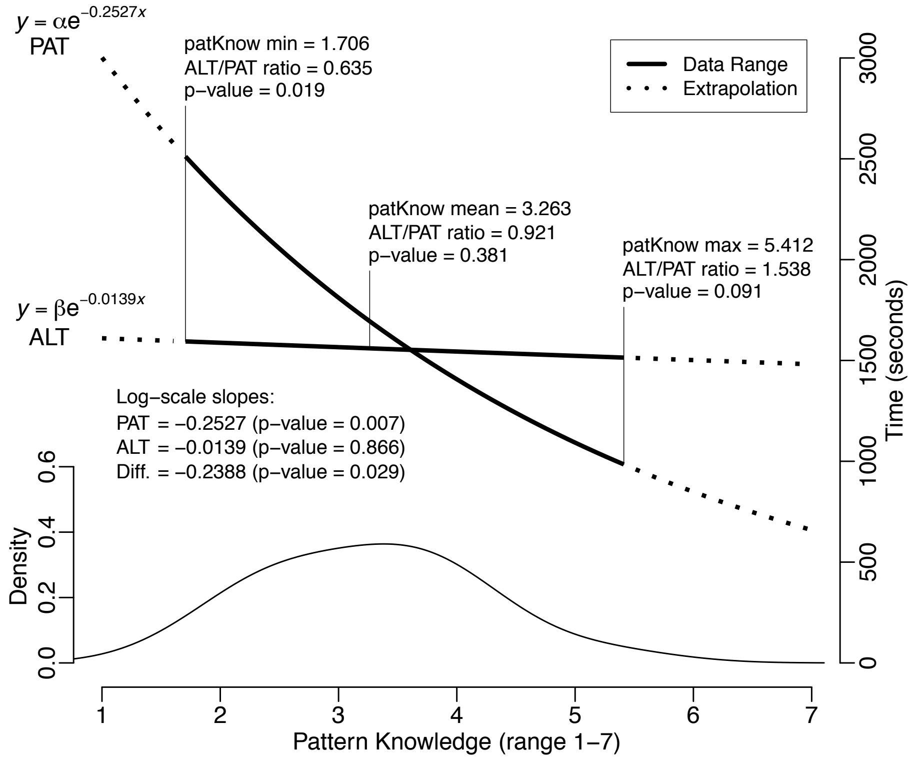
As a covariate, *devExp* is significant in the frequentist models only before filtering (see Table 3.4). A 1-unit (or approx. 17%) increase in *devExp* corresponds with a *time* decrease of 10% for CO (80% confidence interval (CI): 3–17) and 30% for GR (80% CI: 22–37),[9] as well as a *correctness* increase of 7 percentage points for CO (80% CI: 2–13).

As a moderator, we make four observations about *devExp* (see Table 3.5): 1) *devExp* strongly increases the significance (or predictive capability) of *variant* when the two are interacted; 2) *devExp* interacts with *variant* for both response variables; 3) filtering reduces the interaction for *time*, but not for *correctness*; and 4) filtering does not completely eliminate the interaction for either response variable. Thus, *devExp* moderates *variant*, both before and after filtering. Accordingly, we conclude that *generalizing across sites will likely require contextualizing the conclusions with respect to developer experience; generalizing across studies (as opposed to sites) may also require a standardized experience assessment.*

Concerning impact, we find that *variant* is most significant when *devExp* is low (*u*T3,C3:277–296 and *f*T3,C3:277–296). This is true for nearly all tasks, before and after filtering, and for both response variables. Also, when *devExp* is low, PAT tends to take longer and score lower. Thus, *using Decorator or Abstract Factory during maintenance instead of a simpler solution is likely detrimental to inexperienced developers (in the general case).* Conversely, when *devExp* is high, PAT tends to have little impact. Thus, *using Decorator or Abstract Factory during maintenance instead of a simpler solution may be preferable in industry (in the general case), as long as the developers have more experience than our student subjects.* Note that the threshold of experience needed for a specific pattern to be beneficial varies from pattern to pattern; as we show in Section 3.6.3, the threshold is higher for Abstract Factory.

[9] For the frequentist models, we normalized *time* by log transformation prior to analysis. Thus, the *time* models are linear on the log scale, but exponential on the original scale. Accordingly, a 4-unit increase in *devExp* does not equal, e.g., an impossible 120% decrease in *time* for the GR program. Instead, the stated percentage decrease must be repeatedly applied for every 1-unit change in *devExp*, such that *time* asymtotically approaches zero as *devExp* increases.

Figure 3.2: Frequentist results for the *patKnow* × *variant* interaction (CO_time model, filtered data). Since *time* was logged prior to analysis, the back-transformed results are exponential. Intercepts ($\alpha = 3865.3$, $\beta = 1632.6$) are relative to *site*=BYU, *order*=1, *task*=1, *correctness*=60. Other settings for these variables scale the y-axis, but do not alter the ratios or p-values shown.



Pattern Knowledge

In contrast to *devExp*, *patKnow* is significant as a covariate only after filtering (see Table 3.4). Thus, removing low-scoring participants enables detection of the *patKnow* effect by mitigating the effect (or interference) of *devExp*. After filtering, a 1-unit (or approx. 17%) increase in *patKnow* yields an average *correctness* increase of 11 percentage points for CO (80% CI: 4–17), as well as a 30% *time* decrease for GR (80% CI: 20–39). *patKnow* is also significant in the CO_time model, but in that case its marginalized effect is not meaningful because the *patKnow* × *variant* interaction also becomes significant (p-value = 0.029). The *patKnow* × *variant* interaction is depicted in Figure 3.2. The figure shows that *patKnow* affects only the PAT variant. We discuss the frequentist interaction results for *patKnow* further in the next section, at which point we address the original PatMain hypotheses.

The *patKnow* × *variant* interaction is strongly supported in the Bayesian models (see Table 3.5): 1) *variant* increases in significance when interacted with *patKnow*; and 2) like *devExp*, *patKnow*'s moderating influence applies to both response variables (though for *time*, only after filtering). Thus *patKnow* moderates *variant*. This result is not surprising since the original hypotheses anticipated an interaction between *patKnow* and *variant*. It makes sense that knowing more about patterns would help participants cope with the PAT variant more so than with the ALT variant. Thus, similar to *devExp*, we conclude: *generalizing across sites and studies will likely require contextualizing the conclusions with respect to pattern knowledge, and may also require a standardized pattern knowledge assessment.*

Concerning impact, *patKnow* is similar to *devExp* (*u*T2,C2:167–186 and *f*T2,C2:167–186). First, *variant* is most significant when *patKnow* is low. Second, when *patKnow* is low, the PAT variant tends to take longer and score lower. Third, when *patKnow* is high, *variant* is largely insignificant. Thus, we conclude: *using Decorator or Abstract Factory during maintenance instead of a simpler solution is likely detrimental to developers with little knowledge of patterns (in the general case); conversely, it may be preferable in industry (in the general case), as long as pattern knowledge among professional developers is greater, on average, than among our student participants.* Similar to *devExp*, the threshold of knowledge needed for a specific pattern to be beneficial varies from pattern to pattern; again, as we show in Section 3.6.3, the threshold is higher for Abstract Factory.

Motivation

We find evidence that motivation explains cross-site variance both within E_joint and across the three PatMain studies. First, according to Lutz Prechelt, the FUB participants were "true volunteers," who not only stuck around despite two reschedulings, but who were also "helping a fellow student" whose bachelor thesis depended on their participation. Thus the primary motivations at FUB were likely intrinsic. Accordingly, the FUB participants spent far more time on the experiment than any other site (47% more time on average than the next highest site); also, of the 10 participants filtered, *none* were from FUB. In contrast, the primary incentive at BYU and UA was course credit, and in both of those cases many participants had to be filtered. Moreover, the majority of the filtered participants (6 of 10) were from UA—the only site to *require* the experiment as a class assignment.

Motivation also explains why E_repl required filtering, whereas E_orig did not, even though both studies used professional consultants. For E_orig, the CEO invited the study, participation was voluntary, and the only incentive was to learn about design patterns (which at the time were still relatively new). Conversely, E_repl's participants were *selected* to participate by their consultancy firms and were paid for their time. Thus, it would seem that E_orig's participants were more intrinsically motivated. If so, motivation correlates not only with cross-site variance in E_joint, but also with the fact that E_orig was the only PatMain study to not need filtering.[10]

Prior to the moderator analysis, the possibility of motivation being a moderator did not occur to us, so we did not collect any data on the variable as part of the experiment. Thus, of the variables considered so far, motivation is the most in need of further investigation. At the very least, we can conclude that *motivation could strongly influence study outcomes, inasmuch as it affects statistical variance; however, we cannot determine, based on the available data, whether motivation directly moderates the effect of design patterns*. Accordingly, *we recommend that future design pattern studies report on motivation*. Reports should describe both the formal incentives and any other variables that may have motivated or demotivated the participants. A post-experiment survey may be useful to gather such data. We also suggest experimentally controlling motivation in some studies (e.g., by testing multiple types of incentives).

Motivation is clearly a relevant variable in industry, but anticipating the interaction of various incentives in a particular setting is difficult. For example, just because a person is

[10] E_orig was also the only PatMain study to use a paper-based format, so the format could explain the differences in filtering across studies. However, because the format was kept constant within E_joint, it cannot explain the differences in filtering across sites. Thus, motivation is the more likely explanation for the observed variance.

paid does not mean s/he is primarily motivated by that payment. Monetary compensation provides an incentive to get involved, but intrinsic motivation may be needed to maintain focus until a task is completed. In other words, developers are likely motivated by many factors, both intrinsic and extrinsic, and so we cannot assume that paying them to participate in an experiment will *always* produce results representative of industry. Certainly, it can help, but as our data suggest, it is no guarantee.[11]

Other Variables

We analyzed several additional variables, many of which correlate with cross-site variance. In most cases, the data are insufficient to test for a moderating effect. We summarize our findings below. For the detailed discussions, see Appendix Q.

- *task difficulty*: When a task exceeds a certain threshold of difficulty, relative to a developer's experience and/or motivation, the use of patterns appears to have no effect on his/her performance.
- *correctness/time*: These variables positively correlate, meaning the participants likely achieved higher scores by working longer. Also, the variables statistically interact with *variant* before filtering, but not after. Since the filtering targets (in part) undermotivated participants, the interactions suggest that motivation could moderate *variant*.
- *program order*: Performance tended to improve by a small margin on the second program (i.e., lower times and higher correctness), thus indicating a learning (or maturation) effect. The learning effect appears unrelated to design patterns, so we simply correct for it via statistical modeling.
- *perceived time limits*, *cultural variation*, *IDE preferences*, *language barriers*, *clarity of task instructions*, *compilation/testing expectations*: These variables correlate with cross-site variance, but the data are insufficient to test whether they moderate *variant*.

[11] For instance, contractors hired to participate in an experiment could easily view the quality of their work as unimportant (and thus not try as hard as they normally would), since their work product will never actually be used by a paying customer.

Summary of Results

We find evidence that both developer experience and pattern knowledge influence cross-site variance. We also find evidence that these variables moderate the effect of design patterns. Thus, both variables will likely have to be better understood and controlled in order to fully resolve the problem of generalizability. Pattern knowledge was anticipated to be a moderator prior to E_orig; developer experience, however, has not previously been considered as such.

Additionally, we have identified motivation as a strong candidate for explaining cross-site and cross-study variance. Our data are insufficient to statistically test whether it moderates the effect of patterns, but we find indirect evidence for it as a moderator via the *correctness* and *time* covariates. At the very least, motivation could strongly influence study conclusions, inasmuch as it influences statistical variance.

Lastly, we have documented several variables in Appendix Q (summarized above) that correlate with cross-site variance. Although our data are insufficient to test whether these variables moderate *variant*, some of them may prove to be important in future studies.

3.6.3 Original Hypotheses and Final Results

Table 3.6 summarizes the original PatMain programs, tasks, and hypotheses, which were the same for all three studies. Table 3.7 shows the final results compared across the three studies (see below for an explanation of how to read the table). To preserve the integrity of Table 3.7, the results for E_joint are taken exclusively from the frequentist models, which we designed prior to viewing the data and which have not been altered by the moderator analysis. We integrate the moderator analysis into the discussion, however, to resolve contradictions between the PatMain studies.

Like E_repl, our results are based on the filtered data (described in Section 3.6.1). The filtering, which targets underqualified and undermotivated participants, increases statistical precision without significantly altering the main effect estimates. Also, *the conclusions in this section are limited to: 1) the patterns tested in the CO/GR programs (Decorator and Abstract*

Table 3.6: Overview of the PatMain programs, tasks, and hypotheses. For additional details, see Appendix R.

Program	Description	Tasks and Hypotheses (based on E_orig’s published report)
Comm. Channels (CO) Key Pattern: *Decorator*	Wrapper library for establishing connections and transferring packets of data. Provides a facade to a system library. Supports optional logging, compression, and encryption. Not very complex. Simple communication primitives with similar interfaces. PAT: ~360 LOC, 6 classes. Uses a Decorator scheme to add logging, compression, and encryption functionality to a bare channel. ALT: ~310 LOC, 1 class. Uses flags and if-sequences for turning functionality on and off; the flags are set when creating a channel.	1. *Enable error correction (which is already implemented in the underlying system library) to be added to communication channels.* The ALT variant’s functionality is localized, so it will be easier to understand. However, the PAT variant’s functionality is encapsulated, so it will be easier to modify (e.g., to add a new primitive, simply add a new Decorator class). Since the latter influence should be stronger, **the PAT variant will be preferable, especially at higher levels of pattern knowledge.** 2. *Determine under which conditions a reset() call will return the ‘impossible’ result; also, create a channel that performs compression and encryption.* In the ALT variant, state changes are localized and so easier to track than in the PAT variant. Further, creating a channel with the ALT variant requires one statement, whereas PAT requires determining the correct nesting of Decorators. **The PAT groups will take longer and commit more errors.**
Graphics Library (GR) Key Pattern: *Abstract Factory*	Library for creating, manipulating, and drawing graphical objects on output devices. The output device is selected in a central class (generator). Object implementations depend on the selected output device. Objects can be grouped and manipulated like objects themselves. Data structure is partly recursive; more complex than CO. PAT: ~650 LOC, 13 classes. Uses Abstract Factory for the generator classes, and Composite for hierarchical object grouping. ALT: ~640 LOC, 11 classes. Uses a single generator class with switch statements for the different devices. Uses a quasi-Composite for object grouping—allows one level of object grouping, but no nested grouping.	1. *Add a new output device.* PAT requires adding a new factory class and extending the factory selector method. ALT requires enhancing the switch statements in all methods of the generator class. Both variants require adding two concrete product classes. Since the volume of changes is similar, the main difference should depend on comprehension. With its localized switch statements, **ALT will be easier to understand, at least for participants with low pattern knowledge. Also, pattern knowledge will help both groups deal with the Composite, though the PAT participants may profit more since they also interact with Abstract Factory.** 2. *Determine whether a given sequence of statements will result in an x-shaped figure.* This is a comprehension test on Composite, where the key is to recognize that references, and not copies of objects, are stored in an object group. The structure of both variants is similar in the region of interest, so **the ALT and PAT groups will not significantly differ. However, the task will require less time at higher levels of pattern knowledge for both variants, due to the Composite pattern.**

Table 3.7: Comparison of results across the three PatMain studies.*

Hypothesis Statement	Concrete Hypotheses	Baseline & Expectation		E_orig	Reanalysis of E_orig†	E_repl	E_joint‡	
CO Task 1, Coding	$t: P < A$	A	$-$	−38% (<.001)	negative	-	+10% (.344)§	1
The PAT variant will be preferable, especially at higher levels of pattern knowledge.	$t: PH < AH$	AH	$-$	−35%	−55% (<.05)	−49% (<.05)	−35% (.091)	2
	$t: PL < AL$	AL	$-$	−41%	−63% (<.05)	+13% (>.05)	+58% (.019)	3
	$t: H < L$	L	$-$	−3%	-	-	−36% (.090)§	4
	$t: PH < PL$	PL	$-$	+8% (.29)	+10% (>.05)	−49% (<.05)	−61% (.007)	5
	$t: AH = AL$	AL	0	−1% (.46)	−17% (<.05)	+13% (>.05)	−5% (.866)	6
	$c: P > A$	A	$+$	positive	positive	positive	−5 pp (.523)	7
	$c: PH > AH$	AH	$+$	positive	+43 pp (<.05)	+15 pp (<.05)	INS	8
	$c: PL > AL$	AL	$+$	positive	+43 pp (<.05)	+13 pp (<.05)		9
	$c: H > L$	L	$+$	no diff.	no diff.	no diff.	+39 pp (.049)	10
	$c: PH > PL$	PL	$+$	no diff.	0 pp (>.05)	0 pp (>.05)	INS	11
	$c: AH = AL$	AL	0	no diff.	+3 pp (>.05)	−3 pp (>.05)		12
CO Task 2, Comprehension	$t: P > A$	A	$+$	+72%	positive	positive	+10% (.344)§	13
The PAT groups will take longer and commit more errors.	$t: PH > AH$	AH	$+$	+50%	+65% (>.05)	+9% (>.05)	−35% (.091)	14
	$t: PL > AL$	AL	$+$	+91%	+130% (<.05)	+117% (<.05)	+58% (.019)	15
	$c: P < A$	A	$-$	negative	negative	-	−5 pp (.523)	16
	$c: PH < AH$	AH	$-$	-	−15 pp (>.05)	−4 pp (>.05)	INS	17
	$c: PL < AL$	AL	$-$	-	−35 pp (<.05)	+13 pp (>.05)		18
GR Task 1, Coding	$t: P > A$	A	$+$	+17% (.10)	positive	-	+41% (.025)	19
ALT will be easier to understand, at least for participants with low pattern knowledge; pattern knowledge will help both groups, though the PAT participants may profit more.	$t: PH > AH$	AH	$+$	+19%	+30% (>.05)	+40% (>.05)	INS	20
	$t: PL > AL$	AL	$+$	+11%	+35% (>.05)	−17% (>.05)		21
	$t: H < L$	L	$-$	−21% (.021)	negative	positive	−73% (.001)	22
	$t: PH < PL$	PL	$-$	−17% (.17)	−20% (>.05)	+62% (<.05)	INS	23
	$t: AH < AL$	AL	$-$	−23% (.031)	−22% (<.05)	+2% (>.05)		24
	$c: P < A$	A	$-$	-	-	-	−10 pp (.245)	25
	$c: PH < AH$	AH	$-$	-	+21 pp (<.05)	−3 pp (>.05)	INS	26
	$c: PL < AL$	AL	$-$	-	+3 pp (>.05)	+34 pp (<.05)		27
	$c: H > L$	L	$+$	-	-	-	NS	28
	$c: PH > PL$	PL	$+$	-	+10 pp (>.05)	−9 pp (>.05)	INS	29
	$c: AH > AL$	AL	$+$	-	−11 pp (>.05)	+25 pp (<.05)		30
GR Task 2, Comprehension	$t: P = A$	A	0	−21% (.085)	negative	negative	+41% (.025)	31
ALT and PAT will not significantly differ; the task will require less time at higher levels of pattern knowledge for both variants.	$t: PH = AH$	AH	0	−26%	−20% (>.05)	−39% (>.05)	INS	32
	$t: PL = AL$	AL	0	−20%	−30% (>.05)	−9% (>.05)		33
	$t: H < L$	L	$-$	−21% (.091)	negative	positive	−73% (.001)	34
	$t: PH < PL$	PL	$-$	−26%	−17% (>.05)	+11% (>.05)	INS	35
	$t: AH < AL$	AL	$-$	−20%	−28% (>.05)	+66% (>.05)		36
	$c: P = A$	A	0	-	no diff.	-	−10 pp (.245)	37
	$c: PH = AH$	AH	0	-	0 pp (>.05)	−31 pp (<.05)	INS	38
	$c: PL = AL$	AL	0	-	+5 pp (>.05)	+1 pp (>.05)		39

*See Section 3.6.3 for a description of how to read and interpret this table.

† E_repl reanalyzed E_orig's data. This column summarizes the results of that reanalysis.

‡ In E_joint, the *task* interactions are all insignificant. Thus, the results shown for the two tasks in each program are the task-independent results repeated as necessary.

§ These values are taken from the filtered CO_time model, as defined in Section 3.5, but with all interactions dropped.

Factory), 2) maintenance activities, 3) maintainers that were not the original implementers, and 4) programs for which the full functionality of the patterns was not initially needed.

We discuss each of the four tasks in turn. Recall that *coding tasks* required modifying the code, whereas *comprehension tasks* tested comprehension of the code.

How to Read Table 3.7

Due to ambiguity in the original hypotheses, E_orig and E_repl tested slightly different things. To facilitate comparison of their results, we define concrete hypotheses, where:

- t, c = *time*, *correctness* response variables.
- P, A = PAT, ALT variants.
- H, L = high, low pattern knowledge.

Results for each row are computed relative to the baseline. For example, on the first row, −38% means that E_orig estmiated PAT to cause a 38% reduction in time relative to ALT. The expectation (+, −, 0) represents the hypothesized direction of the results relative to the baseline (up, down, or no effect). Non-numeric entries indicate that no statistical results were reported, but data were given about the direction of the effect. A dash (-) means no data were given at all about the effect. Gray entries had to be extracted from plots; their magnitudes are subject to a margin of error (about ±5).

Other abbreviations are as follows:

- pp = percentage points (0–100%).
- (I)NS = (interaction) not significant—i.e., the exact values are not available because the variable (or interaction) was removed during model tuning due to non-significance. For details on model tuning, see Appendix K.

Additionally, note that p-values are provided in parentheses where available; all p-values are two-sided. As discussed in Appendix S, the statistical methods are sufficiently similar between the studies that we can directly compare the numerical results. The CO and GR task 2 hypothesis statements do not address all twelve of the possible concrete hypotheses. Results for the remaining combinations are shown in Appendix T.

CO Task 1 (Decorator Pattern, Coding Task)

The results for this task significantly contradict across the three PatMain studies. However, given our findings from the moderator analysis, we can resolve most of the contradictions. First, the moderator analysis found that the PAT variant was most beneficial (or least harmful) when pattern knowledge was high. On CO task 1, we see this effect manifest in both E_repl and E_joint (Table 3.7, rows 2–3). Second, if we rank the studies by their participants' average pre-experiment pattern knowledge (in decreasing order: E_orig, E_repl, E_joint), we see that the benefit of the PAT variant decreases across the studies, following the order of decreasing pattern knowledge (Table 3.7, rows 2–3, 8–9). Thus, although the results significantly differ across the studies, they agree when considered in reference to the moderating effect of pattern knowledge. As predicted by the moderator analysis, *the time (and possibly also correctness) benefits of the Decorator pattern positively correlate with pattern knowledge on this coding task.*

Further, notice that the pattern training had little impact in E_orig—i.e., before training, the PAT effect was −63%, compared to −55% after (Table 3.7, rows 2–3). This trend makes sense given that, of the three PatMain studies, E_orig's participants had the most prior pattern knowledge. However, according to E_orig's published report, only 52% actually had prior pattern knowledge. Thus, not only does pattern knowledge positively correlate with the benefits of the Decorator pattern (as shown above), but *it appears that only minimal prior knowledge of the Decorator pattern is needed in order to realize a substantial benefit on this coding task*. In fact, the only case in which ALT was significantly better than PAT on this task occured for the *least knowledgable* E_joint participants—i.e., *students* with almost no practical pattern experience whatsoever.

Ultimately, given an understanding of pattern knowledge as a moderator, the original hypothesis for CO task 1 is confirmed. *The Decorator pattern is indeed preferable on this task, especially at higher levels of pattern knowledge—the only caveat being that if pattern knowledge is too low, the time and correctness benefits may be negated.*

CO Task 2 (Decorator Pattern, Comprehension Task)

For this task, the three studies mostly agree. First, PAT took significantly *longer* than ALT in all three cases when pattern knowledge was *low* (Table 3.7, row 15). Second, when pattern knowledge was *high*, none of the *time* effects were statistically significant (Table 3.7, row 14). Third, the *correctness* effects were nearly all insignificant, regardless of pattern knowledge (Table 3.7, rows 17–18). Thus, we conclude that, *for developers with little pattern knowledge, the ALT variant is preferable (at least in terms of time, but possibly also correctness) on this comprehension task*. We also tentatively conclude that, *for developers with high knowledge, the PAT variant is probably no worse than ALT (in terms of both time and correctness)*. Thus, the hypothesis for CO task 2 is confirmed for developers with low pattern knowledge, but tentatively contradicted for those with high pattern knowledge.

Also, as with task 1, pattern knowledge positively correlates with the benefits of the Decorator pattern. However, a greater minimum level of knowledge is required on this *comprehension* task for PAT to be beneficial, than was needed on the *coding* task (task 1). Additionally, we recognize that large estimates, even though statistically insignificant, may still represent meaningful effects (e.g., +65%; Table 3.7, row 14). Thus, we label the second conclusion for this task as "tentative." Greater confidence requires either larger sample sizes or reduced within-study variance.

GR Task 1 (Abstract Factory Pattern, Coding Task)

For this task, the *time* effect mostly agrees across the studies—PAT takes *more time* than ALT (Table 3.7, rows 20–21). However, the results are statistically significant for only one of the three studies, E_joint. Coincidentally, E_joint's participants had the *least* developer experience of all the PatMain studies. Further, the one estimate showing a reverse effect (PAT takes *less time* than ALT) occurred in E_repl—i.e., for the participants with the *most* developer experience. Since the moderator analysis identified developer experience as a likely moderator, we tentatively conclude: *For developers with little professional experience, ALT will likely take less time on this coding task than PAT, but with sufficient experience, the reverse may be true.* However, given that E_repl's participants had a median of 4 years (mean 6.6), the level of developer experience needed for PAT to outperform ALT is likely high. Possibly, for developers with greater pattern knowledge (than E_repl's participants), the minimum level could be less.

As for *correctness*, the estimates are mostly small and insignificant (Table 3.7, rows 26–27). Thus, Abstract Factory likely has little impact on *correctness* for this coding task (in the general case). However, two estimates, one for E_orig and one for E_repl, are large (+21 and +34 percentage points; rows 26 and 27, respectively). Also, the results for pattern knowledge are scattered and inconsistent (Table 3.7, rows 23–24, 29–30). Since neither of these divergences are explained by the moderator analysis, further investigation is needed.

For now, we tentatively conclude that *PAT likely does not impact correctness in general on this coding task, but in some (as yet unknown) cases, it may promote higher correctness.* Concerning pattern knowledge, we conclude: 1) *pattern knowledge is not always helpful on this coding task (in terms of both time and correctness)*; 2) *the conditions under which it is helpful are still unknown*; and 3) *when it is helpful, it does not appear to help the PAT group more than the ALT group.* Thus, the original hypothesis for GR task 1 is (tentatively): partially confirmed for *time*, rejected for *correctness*, and at least partially rejected for pattern knowledge.

GR Task 2 (Abstract Factory Pattern, Comprehension Task)

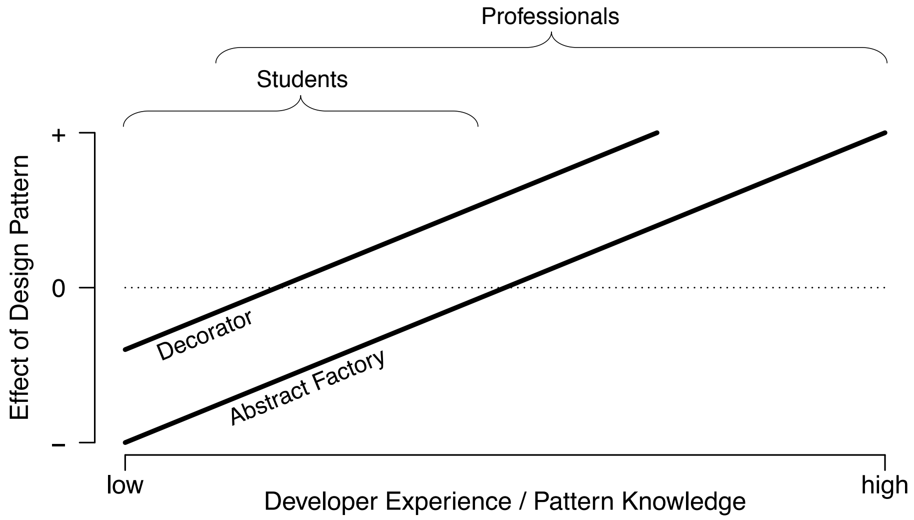
On this task, the *time* and *correctness* estimates are mostly insignificant (Table 3.7, rows 31–39), which supports the original hypothesis that ALT and PAT will not differ. However, we again encounter the problem of insignificant, but large effect estimates (e.g., +66%; Table 3.7, row 36), which means we cannot confidently conclude the null hypothesis. We also find significant contradiction in estimates for both *time* and *correctness*, for which the moderator analysis provides no resolution. In the few cases of statistical significance, especially those of E_joint, PAT took longer and scored lower than ALT. Thus, without further investigation of moderators, we can only tentatively conclude: *In general, PAT likely does not significantly impact time or correctness on this comprehension task; however, in at least some contexts (possibly those of low developer experience), PAT probably does have a harmful influence*; also, *in at least some (as yet unknown) contexts, developers do take less time on this task at higher levels of pattern knowledge, but in other contexts they do not.* Accordingly, the original hypothesis for GR task 2 is (tentatively): partially confirmed for *time*, *correctness*, and pattern knowledge, but also partially rejected for pattern knowledge.

Summary of Results

We can summarize the results for each program as follows:

- *Communication Channels (CO), Decorator. Expectation*: Delocalization of functionality should make the PAT variant easier to modify, but more difficult to analyze and call. *Result*: Using the Decorator pattern instead of a simpler solution is preferable during maintenance, as long as the developer has at least some prior understanding of the pattern. Given even minimal knowledge, the PAT variant is easier to modify; given sufficient knowledge, code comprehension is not negatively affected.
- *Graphics Library (GR), Abstract Factory. Expectation*: Architectural similarities between PAT and ALT should cause only minor differences in the results; where differences

Figure 3.3: Relative impact that developer experience and pattern knowledge have on the effect of Decorator and Abstract Factory, generalized across all three PatMain studies (E_orig, E_repl, and E_joint). A positive effect for a design pattern (+) means that the pattern leads to lower work times and higher quality solutions.



occur, ALT should outperform PAT due to the comprehensibility of its more localized structure. *Result*: Due to several unexplained divergences between the three studies, coupled with the incidence of insignificant, but large effect estimates, the GR results are tentative. Nevertheless, the results suggest that a simplified solution is often equivalent to or better than using Abstract Factory—although given sufficient developer experience (4+ years), the reverse may be true.

Overall, using a pattern where a simpler solution would be possible can be advantageous during maintenance, but only if the developer performing the maintenance has a sufficient understanding of the pattern (and/or a sufficient level of developer experience); also, the critical level of knowledge (or experience) required for the pattern to be helpful appears to be higher for Abstract Factory than for Decorator (Fig. 3.3 depicts the general form of these conclusions). Thus, as Vokáč et al. state, "each design pattern. . . has its own nature, so that it is not valid to characterize patterns as useful or harmful in general" [212, p. 191].

3.7 Threats to Validity

In this section, we describe major threats to validity. We follow a standard validity framework consisting of four parts: construct, conclusion, internal, and external validity [44, 217].

3.7.1 Construct Validity

The extent to which the protocol, treatment operationalizations, and metrics accurately represent the concepts under study. We used a web portal to asynchronously administer the experiment to participants. This protocol allowed participants to work in their own environments, rather than in an unfamiliar room with unfamiliar tools. However, it also introduced the potential for interruptions, browser back-button usage (leading to ambiguous time recording), and collusion. Concerning interruptions, few participants reported any problems, and for those that did, we adjusted task timings. Concerning the back button, we instructed the participants not to use it; based on final comments, most appear to have heeded that instruction. We also aggregated the timings for the download, work, and upload pages to mitigate any impact from work orderings that deviated from the instructions. As for collusion, we conducted an extensive investigation of the participants' solutions, but found no evidence of sharing—the only two dubious similarities between participants eventually turned out to have a convincing technical explanation.

To ensure consistency, we had the same two people grade all tasks (as described in Section 3.4). For the coding tasks, the graders worked in a pair-programming style arrangement. However, for the short-answer tasks, the graders worked separately, each on half of the responses. Thus, the short-answer scores could be inconsistent across participants. In hindsight, we could have asked each grader to grade more than half of the participants and then used the overlap to compute an inter-rater reliability score. Having not done this, we note it here as a limitation and recommend it for future studies. That said, based on three factors, we believe the risk of inconsistency is low. First, unlike the coding tasks, the short-answer tasks involved fairly straightforward, unambiguous answers. Second, the graders initially graded several solutions together, from which they established both a grading rubric and an understanding of how that rubric was to be applied. Third, upon completing their work, the graders cross-checked each other's results specifically for the purpose of ensuring consistency.

3.7.2 Conclusion Validity

The extent to which we can infer relationships in the data, particularly considering the statistical methods used. The moderator analysis is a post-hoc analysis. Post-hoc analyses are useful for exploring experiment instability. However, they inflate the chances of a type 1 error—i.e., the chances of incorrectly concluding an effect exists. Thus their findings must be tested in future studies. Additionally, analyzing moderators requires modeling large interactions, which reduces statistical power. To mitigate this problem, we used Bayesian methods, which allow us to directly compare probabilities for competing hypotheses to determine which are most likely.

Concerning the results in Table 3.7, data dredging (i.e., fishing for significance) is not a problem; those results rely on pre-planned statistical models to address a priori hypotheses. Instead, the primary concerns for Table 3.7 are extraneous variance within studies and heterogeneity of results across studies—which problems the moderator analysis is designed to help resolve. However, inasmuch as the final conclusions from Table 3.7, presented in Section 3.6.3, rely on the moderator analysis, they are also tentative.

For the Bayesian analysis, we enlisted a qualified external researcher to estimate the prior distributions. We instructed him to choose broad priors in order to minimize the weight of those priors on the final results. In general, choosing broad priors leads to broader posteriors, but for a post-hoc analysis, sacrificing some precision is an acceptable tradeoff—i.e., a bias toward type 2 errors is appropriate given that the post-hoc nature of the analysis inflates the chances of a type 1 error.

In addition to choosing broad priors, we also could have enlisted a second external researcher to independently select priors. Given multiple estimates for priors, we could have performed a sensitivity analysis to verify our expectation that the priors have little influence on the results. Not having done this, we recognize it here as a limitation and recommend it for future studies conducting similar analyses.

3.7.3 Internal Validity

The extent to which we know that the treatment caused the observed changes. In E_joint, we assessed pattern knowledge via a survey (an observational assessment) instead of a training course (a randomized, controlled assessment). Thus, causality inferences relating to pattern knowledge are tentative. However, since we synthesize our results with those of the prior two PatMain studies, this threat is not a significant concern for the final conclusions.

3.7.4 External Validity

The extent to which the results can be generalized to other situations and people. Our conclusions are limited to the patterns tested in the CO/GR programs (Decorator/Abstract Factory), maintenance activities, and programs for which the full functionality of the patterns was not initially needed. The maintainers were also not the original designers/implementers, and real programs are typically larger and less well commented than the PatMain programs.

E_joint's participants also had little developer experience and pattern knowledge. However, since the final conclusions are based on all three PatMain studies, they represent a fairly broad range of developers.

Additionally, for both E_repl and E_joint, the replicating researchers interacted considerably with prior experimenters, which means the likelihood of shared bias is high among all three studies. For a description of the cross-study interactions, see Appendix V. Shared bias is not necessarily a bad thing. It helps in the early stages of testing in order to reduce unexpected variance across experiments. However, it does indicate a weakness in external validity, which can only be addressed by eventually replicating the study with little or no cross-study interaction—other than the use of published reports and (possibly) lab packages.

Ultimately, given that we have successfully generalized the results across all three PatMain studies—which collectively represent 58 students and 68 professionals from 17 institutions and 4+ countries—the most significant threats to external validity are not population related; rather, they concern the size and complexity of the software programs being tested, as well as the fact that the developers worked in isolation, rather than in team environments.

As discussed in Appendix Q, we found evidence in E_joint that program complexity could impact the outcome of the PatMain experiment. Although inconclusive, the data suggest that if the difficulty of a problem exceeds a certain threshold (relative to a developer's experience and/or pattern knowledge) then design patterns will have little impact on work time or solution correctness. However, we do not know whether this observation would hold in industry, much less whether it would be experimentally repeatable. Quite possibly, the reverse is true—that the real value of design patterns is only manifest for big software of the magnitude found in industry. To answer these questions, additional studies are needed explicitly targeting the issue of program complexity.

Concerning the issue of programming environments, it is highly possible that isolated developers would respond differently to design patterns than developers working in a team. Possibly, via the sharing of knowledge, team dynamics may compensate for any negative effects of design patterns. On the other hand, team dynamics may also attenuate any benefits, thus causing patterns to have little impact overall. At this point, we cannot say what impact team-oriented development practices would have on the effects of patterns. To answer this question, additional studies are needed.

3.7.5 Other Threats to Validity

For additional threats to validity which, due to space constraints, we could not include here, see Appendix U.

3.8 Conclusions

In this section, we present conclusions. We divide the discussion into subsections matching the structure of the contributions set out in Section 3.2.

3.8.1 Design Patterns

We find that both *developer experience* and *pattern knowledge* moderate the effect of design patterns, such that a higher level of either tends to enhance the benefits of patterns (or reduce their harm) during maintenance. We also find indirect evidence for *motivation* as a moderator. At the very least, we can tentatively conclude that lack of motivation increases statistical variance, which in turn can confound the comparison of results across studies. Whether and to what degree motivation impacts the effect of patterns outside the experimental setting is still unclear.

Based on the two moderators for which we have quantitative data—developer experience and pattern knowledge—we were able to fully resolve conflicts for one of the two patterns studied (Decorator) and partially for the other (Abstract Factory). Each of the final conclusions (summarized below) generalizes across all three PatMain studies (E_orig, E_repl, and E_joint), involving 126 participants from five universities and twelve software companies—thus covering a broader set of contexts than has previously been achieved in the study of design patterns. Such a high level of generalization would not have been possible without the moderator analysis.

1. The Decorator pattern is preferable to a simpler solution, as long as the developer has at least some prior knowledge of the pattern.

2. For Abstract Factory, the simpler solution is mostly equivalent to the pattern solution.
3. Abstract Factory requires a higher level of pattern knowledge and/or developer experience than Decorator for the pattern to be beneficial.

In general, using a pattern where a simpler solution would be possible can be advantageous during maintenance, but only if the developer performing the maintenance has a sufficient understanding of the pattern.

3.8.2 Research Methods

Our replication was itself a test of a new method for conducting replications, which we term *joint replication*. We recommend joint replication for investigating the context sensitivity of experimental results. By evoking heterogeneity within a controlled setting, in which many of the typical confounding variables are controlled (e.g., time between studies), joint replication enables the evaluation of moderators in greater detail than typically possible.

We also used Bayesian statistics and post-hoc methods for the analysis of moderator variables. Bayesian statistics are useful in cases where power is limited due to small sample sizes and high variance. Post-hoc methods help to uncover relationships that would otherwise remain undetected given only a-priori methods. We recommend both of these approaches for studying moderators. In particular: 1) the Bayesian methods allowed us to analyze larger interactions in greater detail than was possible via the frequentist methods; 2) the post-hoc nature of the analysis allowed us to identify several variables which we had not previously considered; and 3) the post-hoc analysis helped us to narrow the vast search space of potentially meaningful variables down to a few good candidates for the next round of investigation.

Ultimately, the joint replication and Bayesian analysis empirically and statistically ground our conclusions, providing measured confidence that we have identified meaningful moderators, rather than experimental or chance artifacts.

3.8.3 Research Methodology

Joint replication, post-hoc analysis, and Bayesian statistics represent a general method for resolving contradictory experimental results. As such, they help fill the gaps between experimentation, confirmation, and knowledge production. However, *the process of exploring moderator variables also naturally leads to explanatory theory.* The benefit of such theory is that it allows researchers to shift the goal of replication from results *reproducibility*, which often fails, to experiment *predictability*, which can more easily cross contextual boundaries.

In fact, lack of context awareness is one reason why external replications (i.e., replications performed by researchers unaffiliated with the original study) have been far less successful at reproducing results than internal replications [49, 197]. For example, according to a recent survey of 16,000+ software engineering papers from 1994–2010 [49], only 26% of external replications were able to reproduce the original results, whereas 82% of internal replications were able to do so. In other words, getting the protocol right in a replication is necessary, but not sufficient to obtain usable knowledge from the results; context matters [40, 135].

Despite its difficulty, external replication is important because it represents one of the most effective mechanisms available for mitigating researcher bias. In turn, joint replication facilitates external replication by striking a balance between the internal/external replication tradeoff—i.e., by coordinating a set of otherwise independent research teams in such a way as to enable industry-relevant context variables to be mapped across sites—the result of which enables the development of context-sensitive theory.

Essentially, moderator variables encapsulate and formalize contextual information as part of the experimental framework. In turn, joint replication, post-hoc analysis, and Bayesian statistics provide a process for identifying and articulating moderator variables. Together, the concepts yield a research strategy which, at the very least, shows promise for facilitating generalizability across closely replicated experiments. However, with additional development, it could potentially lead to higher-level generalization as well.

Ultimately, as our results indicate, effective generalization (at least for highly variable contexts) requires testable theory about context. In developing such theory, we argue that replication can and should play an *exploratory* role. The utility of replication is not limited to the most basic forms of validation and generalization (relying solely on results reproduction). Rather, in the pursuit of those goals, replication's most immediate benefit is its ability to propel us into the domain of our ignorance—to show us where our knowledge is deficient and, at the same time, to provide data by which to investigate those deficiencies.

3.8.4 Future Work

Due to space constraints, future work appears in Appendix W.

3.9 Acknowledgments

We are grateful to: Martin Liesenberg for building the web portal; Ulrich Stärk for expert advice to Martin; Martin Liesenberg and Christian Bird for the Java and C# program translations, respectively; Alexander MacLean and Landon Pratt for grading all solutions to ensure consistency in the scoring across sites; Paul Felt for sharing source code and advice concerning his Bayesian models; Gilbert Fellingham for expert advice on the Bayesian analysis; Marek Vokáč for providing data and historical information about the first replication of the PatMain experiment, as well as for member checking our analysis and reporting of his experiment; RESER 2011 reviewers and participants for encouragement and valuable feedback; and all study participants for their contributions. This research has been partially supported by the grant TIN-2011-23216 (Spanish Ministry of Economy and Competitiveness).

Chapter 4

A Method for Generalizing across Contexts in Software Engineering Experiments

Context. Experimental software engineering notoriously struggles to produce usable (i.e., transferable) knowledge. Unlike internal replications, the vast majority of external replications fail to reproduce prior results. A primary cause for failed replications is contextual variation. Thus, to produce usable knowledge, we need methods for identifying, evaluating, and theoretically integrating context variables. **Objective.** Develop a method to produce a maximum of knowledge about context variables via a modest number of replications. **Methods.** In this paper, we present a Tractable method for Context Analysis (TCA), which we developed via the replication of a seminal experiment on design patterns (known as PatMain). TCA involves three components: joint replication, post-hoc moderator analysis, and Bayesian models. For each component, we describe its theoretical background and practical implementation (using PatMain as a working example). **Results.** The PatMain series of studies were all close replications, yet their results diverged considerably. TCA resolved the divergences sufficient to produce general conclusions (representing 126 participants from five universities and twelve software companies). **Conclusions.** As our results indicate, effective generalization (at least for highly variable contexts) requires testable theory about context. TCA facilitates the development of such theory by enabling the investigation of context variables in greater detail than previously possible.

4.1 Introduction

The software engineering literature is replete with statements advocating empirical replication—such as, “replication is a key feature of experimentation in any scientific or technological field” [104, p. 295], “replication is a basic component of the scientific method, so it hardly needs to be justified” [108, p. 219], or more simply, “The value of experimental replications is evident. . . ” [40, p. 1]. As a research community, we believe that replication is essential to knowledge production.

The purpose of replication is twofold: 1) to establish confidence in a previous result (i.e., the reliability test); and 2) to explore sources of variability that influence a result (i.e., the generality test) [11, 190]. When effectively practiced, replication has the power to build knowledge [32, 190]. As Juristo and Vegas note, “After several replications have increased the credibility of the results, the small fragment of knowledge that the experiment was trying to ascertain is more mature” [102, p. 356].

However, replication in software engineering is complicated by the fact that it often fails to reproduce prior results [49, 102, 104], especially in the case of *external* replications [49]—i.e., replications performed by researchers other than those who conducted the original study [4, 32]. According to a recent survey [49], only 65% of replications from 1994–2010 were able to fully reproduce the original results. As for *internal* replications, the success rate is much higher (82%), but for *external* replications, it is considerably lower (26%). Although the survey does not differentiate between non-confirmation and contradiction, even in the best case, the vast majority of external replications (74%) still at least partially fail to reproduce results, and almost half (46%) completely fail. Thus, despite our success with internal replications, *a significant portion of our experiments are failing to produce usable (i.e., transferrable) knowledge*; either key details are not being sufficiently communicated across studies or our conclusions are too brittle to generalize across contexts.

Communication, or knowledge-sharing issues (e.g., reporting standards, lab packages, and the sharing of tacit knowledge) are one of the most frequently cited explanations for replication failure [24, 40, 105, 189, 191]. However, much work has been done in the last decade to improve inter-study communication, as well as experimental guidelines (e.g., [40, 96, 109, 113, 189, 191, 198, 199]), and yet the problems persist [49].

Another cause of replication failure—and one which also explains why external replications fail more often than internal replications—is contextual variation [49, 135]. In software engineering, factors such as programming experience, motivation, languages, and tools all vary considerably from project to project. Such variation affects most software engineering studies to some degree [19, 74, 137], but more particularly those involving human participants [188].

Contextual variation inhibits not only the success of individual replications, but also the synthesis of results across replications [196]. Meta-analysis is the current standard for synthesizing quantitative results across studies [108, 160]. However, meta-analysis is unstable when study results are too heterogeneous [56, 147]. Meta-analysis also yields global generalizations, which are less meaningful in highly variable contexts because they may be inaccurate in any specific context.

Thus, software engineering is confronted by a *problem of generalizability*—i.e., the tendency of many results to be unstable across varying contexts, so that considerable knowledge of context variables is required before generalizable statements can be made. The most damaging consequence of this problem is that, until solved, *the majority of our experimental results cannot be trusted to apply in industry.* To produce usable knowledge, we need methods for identifying, evaluating, and theoretically integrating context variables. The aim of this paper is to describe one such method.

4.1.1 The Tractability of Context Analysis

Worth addressing at the outset of this paper is the concern that context analysis is doomed to failure because it involves a *very large* search space. Dybå et al. [60, 63], for instance, argue the intractability of *discrete* (i.e., variable-oriented) context analysis, pointing out the vast number of potentially relevant variables (and combinations of variables) that must be navigated under such a perspective. In response, they propose an *omnibus*, or broadened journalism-style perspective structured around five axes: who, what, when, where, and why. In particular, they argue against the use of checklists of potentially relevant context variables (e.g., that proposed by Petersen and Wohlin [159]).

We agree with Dybå et al. that the contextual search space is vast, and more particularly, that we must be intelligent about how we analyze that space if we are to succeed. However, we disagree that an omnibus perspective is objectively better than a discrete perspective. At the very least, not enough is yet known about either perspective to make such a claim.

In defense of the discrete perspective, we note that 1) other fields have used it with success (e.g., social psychology [13]); 2) we ourselves have used it with success [127], which results we summarize in this paper; and 3) recent work in several fields [139] indicates a general trend among complex systems (including biological systems and animal behavior, e.g., [5, 152, 202, 203]) to be predictable via a small subset of key variables (more on this in Section 4.4.2).

Likely, with sufficient development, both perspectives (discrete and omnibus) can be leveraged effectively in software engineering. Additionally, although discrete, our approach to context analysis coincides with many of the philosophical points that Dybå et al. present in support of the omnibus perspective. Our method also parallels many of their practical guidelines, including those concerning the use of qualitative data, the reporting of contextual information, and the use of distributions rather than just means.

4.1.2 Research Question

Our work investigates the following research question: *How can we produce a maximum of knowledge about context variables via a modest number of replications?*

4.1.3 Contributions

In this paper, we describe a method for identifying and evaluating context variables, which we refer to as TCA (*a Tractable method for Context Analysis*). TCA involves three components: joint replication, post-hoc moderator analysis, and a specific type of Bayesian model.

- *Joint replication* is an idea originally proposed by Lutz Prechelt in connection with the RESER 2011 workshop [126]. The idea is to organize a multi-site study, performed by separate research teams whose efforts are coordinated (to minimize execution-related differences), yet each team acts independently in performing its own replication.
- *Post-hoc moderator analysis* involves two elements: post-hoc methods and moderator variables (or moderators for short). Post-hoc methods are commonly used in other fields to uncover relationships that would otherwise remain undetected given only a priori methods. Moderators are explanatory variables that interact with other explanatory variables in predicting a response variable [183]—i.e., moderators are a type of context variable.[1]
- *Bayesian models* are *not* commonly used to analyze experimental data in software engineering. However, such models are useful in cases where statistical power is limited due to small sample sizes and high variance [54], which are not only common problems in software engineering, but also key barriers to context analysis. We apply a specific type of *additive-effects* model described by Felt [69], which avoids linearity assumptions.

As we show in this paper, TCA enables investigation of context variables in greater detail than previously possible. Thus, it can be used to reconcile contradictory experimental results, to improve the conduct of future replications, to resolve heterogeneity in meta-analysis studies, and to guide researchers in selecting information for reports and lab packages. With

[1] We use the terms *explanatory* and *response* instead of *independent* and *dependent* to refer to variables because, in many cases, the explanatory variables are not statistically independent. For this reason, we need to study moderators.

sufficient development, we anticipate the method could be used to produce generalized conclusions that are broadly and demonstrably applicable to industry.

4.1.4 Approach

We developed and tested TCA via the replication of a design patterns experiment. We closely replicated a 2001 study by Prechelt et al. [168] assessing the impact that design patterns have on software maintenance (abbreviated as "PatMain"). We chose design patterns because many studies (including replications) have investigated patterns, and to date their results are mostly irreconcilable [6, 222]—likely due to the influence of context variables. We chose PatMain specifically because it is the second controlled experiment ever undertaken to study design patterns (the first being "PatDoc" [169]). PatMain has also been replicated once previously (in 2004 by Vokáč et al. [212]) with divergent results—thus making it a good case for study.

Not surprisingly, the results of our replication diverged from those of the prior two studies—thus making three replications, each with contradictory results. *However, the method we describe in this paper ultimately resolved the divergences sufficient to produce generalized conclusions.* Thus, our replication produced two deliverables: 1) a set of generalized conclusions concerning design patterns, and 2) a new method (TCA) for investigating context variables. The results on design patterns are published in a separate paper [127], along with details sufficient to replicate the study. In this paper, we focus on methodology, discussing only those aspects of PatMain necessary to illustrate the method.

4.1.5 Structure of this Article

In Section 4.2, we discuss related work. In Section 4.3, we summarize the PatMain experiment, which we use throughout the paper as a working example. In Sections 4.4 and 4.5, we describe the TCA method and show how its output can be used to resolve contradictory experimental results. In Section 4.6, we discuss limitations, and finally, in Section 4.7, we conclude. *All* *appendices are included in the supplemental materials for this paper, which can be downloaded from the publisher's website.*

4.2 Related Work

In this section, we review current methods for synthesizing results across studies, including methods for context analysis. The purpose of this section is to establish a need for TCA, as well as to show how it relates to other similarly-purposed methods.

4.2.1 Frameworks for Relating Studies

Three frameworks are described in the literature for relating studies in order to build knowledge:

Daly et al. [51, 148, 219] describe a multi-method approach that calls for a series of studies, each of which relies on a different empirical method (e.g., structured interviews, followed by a survey, followed by a laboratory experiment). By varying the research method, Daly et al. argue, the results can be shown to be more robust—at least inasmuch as they agree across studies.

Basili et al. [19] propose a framework for organizing *families* of experiments. The idea is to group studies that address similar concepts for the purpose of forming meta-level conclusions and identifying experimental gaps in the underlying theory. In a sense, Basili's concept of *family* is simply a broadening of the traditional concept of replication.

Krein and Knutson present a framework for relating replications [120]. The framework is essentially a conceptual diagram showing how various replication types fit into the overall knowledge-building process of science.

Ultimately, each of these three frameworks adds insight to the issue of replication, but none provides a clear-cut practical method for synthesizing results.

4.2.2 Systematic Literature Reviews

Systematic literature reviews (SLRs; e.g., [61, 111]) represent another possible solution to the problem of synthesis. According to Kitchenham, SLRs are “a means of identifying, evaluating and interpreting all available research relevant to a particular research question” [107, p. 1]. Unfortunately, Cruzes and Dybå [48] report finding *synthesis* to be the single most challenging (and neglected) component of SLR research—nearly half of the studies surveyed did not contain any synthesis, and of those that did, two thirds performed only basic thematic and narrative synthesis. Further, of the synthesis methods identified by Cruzes and Dybå, most are heavily or entirely qualitative; only one, meta-analysis, was found to be fully statistical. Below we summarize meta-analysis, as well as two other approaches not mentioned by Cruzes and Dybå.

4.2.3 Methods for Quantitative Aggregation

Meta-analysis is the current standard for aggregating quantitative results across studies [108]. Meta-analysis is a process of pooling data to increase the number of observations, thereby reducing statistical error [56, 160]. Meta-analysis can be used to combine data even in cases where studies report contradictory results, as long as the overall variance is not too extreme [160]. However, experiment variables must match and, to some degree, the studies must be independent [108].

Bayesian methods [54, 205] are an alternative to traditional meta-analysis which allow data to be accumulated over time from a series of experiments by incorporating past results as prior knowledge into the analysis of future replications. Bayesian methods can also be used to combine results such that all data are treated as current observations, in which case additional parameters can be used to account for differences between studies. Either way, Bayesian methods yield posterior probabilities, which can be preferable over p-values in a variety of circumstances (e.g., when statistical power is low, as we discuss in Section 4.4.2). Bayesian methods also naturally handle missing data [54].

A third option is to simply analyze all data together using traditional frequentist methods (as is done by Runeson et al. [180])—e.g., analysis of variance (ANOVA), multiple regression, or mixed models. Like the Bayesian approach, additional variables can be incorporated to account for differences between studies.

The difficulty of the latter two options (i.e., the Bayesian and frequentist options) is that they are both a form of *raw-data aggregation*, and so both require raw data to be available from all studies involved, which is often not the case in software engineering. Further, even when the necessary data are available, those data may be poorly documented or inconsistently measured [23], thus making the analysis fraught with assumptions and error prone. Consequently, meta-analysis is currently the de facto standard.

However, meta-analysis is not without limitations. In 2000, Miller [147] applied it to a set of software engineering experiments, but found the results to be highly unstable. Miller concluded that the root cause was cross-study contextual variation. Other software engineering researchers have also reported finding high levels of contextual variation [19, 74, 137, 188].

Despite the problem of contextual variability, Dieste et al. [56] argue that meta-analysis can still be effective given sufficiently large datasets. Accordingly, they propose that publishers should accept small-scale replications, the rationale being that such replications are easier to perform, thus leading to more published studies and larger aggregate datasets.

Another option, proposed by Miller [147] (as well as others [130]), is to transform a heterogeneous dataset into a group of homogeneous datasets by accounting for moderators. This latter option, however, requires quantitative methods for identifying moderators, which is a difficult problem and the primary purpose of the present article. In other words, the output of the method we describe in this paper can be used as input to resolve heterogeneity in meta-analysis studies.

4.2.4 Context-Sensitive Generalization

Ultimately, as Lindsay and Ehrenberg argue [135], meta-analysis is no silver bullet. It only works when variance is sufficiently small relative to the size of the dataset [56]. The problem for software engineering is that experimental outcomes notoriously vary from context to context, and as da Silva et al. have shown [49], we cannot expect a dramatic increase in the number of replications for specific studies any time soon. Further, for most research questions, it is not yet clear how large the dataset would need to be in order to obtain stable results since the extent of the contextual variance cannot be known without additional data.

In light of such challenges, psychologists Lindsay and Ehrenberg [135] argue in favor of studying context variables in order to synthesize across replications. In their view, replication is precisely a process of context-sensitive generalization. Verification (or the reliability test), for instance, is really just a narrow form of generalization. As Lindsay and Ehrenberg explain, "we need to cash in on such differences in the conditions of observations as do occur. . . rather than to try to sweep them under the carpet" [135, p. 220].

Traditional meta-analysis does not solve the context-sensitive generalization problem. Rather than establishing the generalizability of the results under the various conditions of observation, it simply seeks to reduce sampling errors by pooling data [135]. Moreover, it "does not tell us in what way its scope (that is, its generalizability) has increased with a successful replication, or how it has been circumscribed or negated with an unsuccessful one, and what, in either case, one might therefore want to do next" [135, p. 219]. Thus, *studying context variables to facilitate synthesis provides a clear advantage over using meta-analysis alone; the former approach not only enables meta-analysis in cases where it would not otherwise be possible, but also provides groundwork for theory construction* [98]. Ultimately, as Lindsay and Ehrenberg point out, replication "is needed not merely to validate one's findings, but more importantly, to establish the increasing range of radically different conditions under which the findings hold, and the predictable exceptions" [135, p. 217].

4.2.5 Methods for Context Analysis

In keeping with Lindsay and Ehrenberg's perspective, Juristo and Vegas [102, 104] propose a process for identifying context variables in non-exact replications. The process concerns all four phases of a replication: definition and planning, operation and analysis, results interpretation, and contribution evaluation. In a related paper, Juristo et al. [105, 211] outline a method for leveraging communication between researchers to identify context variables. This process—which involves structured meetings, as well as unstructured phone calls and emails—can be nested within and used as part of the broader process mentioned above.

Both processes provide a helpful framework for structuring the investigation of context variables. However, both also suffer from three key deficiencies: 1) they are specified at fairly high levels, leaving researchers the burden of figuring out how to apply them in practice; 2) they rely on subjective qualitative methods to pick out candidate variables; and 3) they say little about what to do once a candidate variable is identified, other than to conduct additional replications.

Alternatively, context variables can be quantitatively investigated by aggregating raw data from multiple studies into a single statistical analysis, as described previously. However, context data are even less likely than experiment data to be available, consistently measured, and adequately documented (even in the case of close replications; e.g., see [180]; also compare [168], [212]).

A third option, fUSE [42], is built on top of meta-analysis, and so does not require raw data. fUSE adds two tests to meta-analysis, one for assessing whether moderators need to be investigated and one for evaluating the explanatory potential of candidate moderators. fUSE is modestly robust to both missing data and to the problem of metric standardization.

However, being based on meta-analysis, fUSE inherits several weaknesses: 1) it requires independence between studies for the results to be accurate; 2) it does not specify a process for discovering moderators—i.e., it does not address the tractability problem described by Dybå et al. [63]; 3) it can only be used to test a moderator for which each individual study represents one cohesive level or subgroup of that moderator; and 4) since it uses experimental runs as the unit of measure, instead of individual observations, it requires a large number of replications. This last limitation is especially problematic for the analysis of moderators because such analysis requires subdividing the data (e.g., see [42], in which a set of 21 experimental runs was still insufficient to achieve satisfactory results).

4.2.6 The State of Results Synthesis

In summary, current replication frameworks are abstract and do not define concrete methods for results synthesis. Systematic literature reviews are by definition intended to handle results synthesis, but thus far deal more with the collection of relevant articles than with synthesis itself. Synthesis methods from other fields are mostly qualitative and/or involve subjective components, with the exception of two general options: 1) meta-analysis, and 2) raw-data aggregation.

The primary problem with meta-analysis (at least for software engineering) is that its results are unstable in highly variable contexts. fUSE represents a possible solution to that problem, but being based on meta-analysis, requires a large number of replications (likely 30 or more) to achieve satisfactory results. Thus, in software engineering, meta-analysis is confronted by a major roadblock—the cost to obtain a dataset large enough to address most questions of interest is impractically high. Unfortunately, raw-data aggregation is also inhibited by problems of data availability—though in that case, the problems are due to the difficulty of curating and reconciling data across past experiments.

The ideal solution to these problems requires obtaining sufficient, current, and clean *raw data* from the right selection of participants, so as to minimize the number of samples and studies necessary to produce generalized conclusions. As we show in this paper, such an objective can be accomplished, at least in a heuristical manner. Given a context-aware sampling technique, data-driven processes for identifying likely moderators, and statistical tools that are robust to high variance, we can reduce the problem to a handful of studies involving a sample size not much greater than that of most current software engineering experiments.

4.3 Working Example (PatMain)

In this section, we summarize the PatMain experiment, which we use as a working example throughout the paper. We provide only the minimally-necessary details here. For additional information, see [127].

The purpose of the PatMain experiment is to investigate the following research question: *For a given problem, if using a design pattern is "overkill" (i.e., the pattern provides more functionality or flexibility than necessary), will the resulting solution be more difficult to maintain than if a simplified solution were implemented instead?*

The PatMain experiment has been executed twice before, including the original study by Prechelt et al. [168] and one replication by Vokáč et al. [212]. We conducted our replication as a *joint replication*—which is essentially a multi-site study, wherein each site works independently, yet the sites are coordinated to minimize execution-related differences (see Section 4.4.1 for further details on joint replication). We conducted our replication as part of the 2011 *Workshop on Replication in Empirical Software Engineering Research* (RESER) [75, 124, 175]. Initially eight research teams expressed interest, of which four submitted data: Brigham Young University (BYU) [126], Freie Universität Berlin (FUB) [167], The University of Alabama (UA) [155], and Universidad Politécnica de Madrid (UPM) [103].

From this point forward, we refer to the original PatMain study (by Prechelt et al.) as *E_orig*, the first replication (by Vokáč et al.) as *E_repl*, and the joint replication (our work) as *E_joint*. Also, to reference these experiments generically, we use the term *study*, as in, "the three PatMain studies." To generically reference sub-replications of E_joint (i.e., BYU, FUB, UA, and UPM), we use the term *site*, as in "the four E_joint sites."

4.3.1 The Original Study (E_orig)

E_orig consisted of four C++ programs, implementing five Gamma et al. [77] design patterns:

- Stock Ticker (ST): *Observer*
- Boolean Formulas (BO): *Visitor*, *Composite*
- Communication Channels (CO): *Decorator*
- Graphics Library (GR): *Abstract Factory*, *Composite*

Each program was implemented in two variants: one using design patterns (PAT), the other using a simplified, alternative design (ALT). The simplified design discarded all patterns not required for the program. The experiment was *paper-based* and included 2–3 tasks for each program. Some tasks required modifying code; others tested comprehension. The experiment began with a pre-test involving two programs (one PAT, one ALT). A patterns training course was then administered, followed by a post-test, involving the remaining two programs (again one PAT, one ALT). All participants received all four programs. Program and variant orderings were alternated.

This design resulted in six explanatory variables: *program*, *task*, *variant*, *patKnow* (i.e., pattern knowledge), program *order*, and *subjectID*. The experiment assessed two response variables: *time* and *correctness*. For each *program* and *task*, 2–3 hypotheses addressed the impact of *variant* and *patknow* on *time* and *correctness*.

The 29 participants were all volunteers, software professionals from the consultancy firm sd&m in Munich, Germany. The median industry experience was 3.5 years (mean 4.1), including 2 years (mean 2.4) with object-oriented programming. Fifteen (52%) of the participants had prior experience with patterns.

4.3.2 The First Replication (E_repl)

In general, E_repl closely replicated E_orig's design. All changes were designed to increase the realism. Changes included: 1) participants worked on computers rather than on paper;

2) participants were selected by consultancy firms to participate, rather than being volunteers; and 3) participants were paid.

E_repl's 44 participants included 39 software professionals from 11 consultancy firms and 5 graduate students. The median industry experience was 4 years (mean 6.6), including 2 years (mean 2.4) with object-oriented programming. Seventeen (39%) of the participants had prior experience with patterns.

4.3.3 The Joint Replication (E_joint)

Three motivations prompted E_joint: 1) we wanted to test a new method for performing distributed replications based on closely coordinated, small-scale instances (i.e., joint replication); 2) we were interested to see how homogeneous the results would be across sites; and 3) we wanted to address the issue of contradictory results among design pattern studies.

In general, we strove for a close replication. However, being a joint replication, researcher participation and consistency across sites were important considerations. To facilitate these objectives, we made two key changes. First, we administered the experiment via a web portal. The portal managed experiment groups, administered the tasks and pre/post-questionnaires, and facilitated the download/upload of source code. The portal also tracked work times by measuring the time spent on each page. Second, we simplified the burden on researchers and participants by eliminating the training course and post-test, thus reducing the experiment from four programs (ST, BO, CO, GR) to two (CO, GR). In lieu of the training course, we assessed pattern knowledge via a pre-questionnaire.

Other changes included: 1) like E_repl, our participants worked on computers, rather than on paper; and 2) our participants worked in Java instead of C++.

Our protocol preserved all variables from E_orig with only minor changes to *program* and *patKnow*. We also added explanatory variables to represent developer experience (*devExp*) and *site*. For *patKnow* and *devExp*, we aggregated data from the pre-questionnaires to form continuous variables on a fixed range (1–7).

The four sites independently solicited participants. All participants were students, including 27 undergraduate and 26 graduate (MS or PhD) students (53 total). The median industry experience was 0 years (mean 1.5). Almost all of the participants had broad *exposure* to patterns, but almost none had any *practical* (especially industry) experience with them.

4.4 The TCA Method

In this section, we describe the TCA method, which involves three components:

- *Joint replication*: a sampling process for obtaining sufficient, current, and clean raw data on context variables without requiring an excessive number of participants or studies.
- *Post-hoc moderator analysis*: pre/post-experiment processes for identifying salient context variables.
- *Bayesian models*: a quantitative process for evaluating context variables that supports conclusions even when statistical power is low.

We present each component in turn, discussing its theoretical background and practical implementation. The overall strategy is to use Bayesian models to investigate moderators within the context of a joint replication. However, the components can be selectively applied as needed (e.g., if suitable data are available from past studies, then joint replication may not be necessary).

4.4.1 Joint Replication

The idea of a *joint replication* is to organize a multi-site study, performed by separate research teams whose efforts are coordinated, yet the researchers at each site act independently in performing their own replication. To minimize execution-related differences, the research teams explicitly communicate about important aspects of the experiment, including adopting a common definition of the experiment. However, each team gathers participants, collects data, and performs initial data analysis separately, after which the datasets are then merged and analyzed together. Note that joint replication is similar to *multi-site randomized controlled trials* (RCTs) in social work research [200, pp. 173–176].

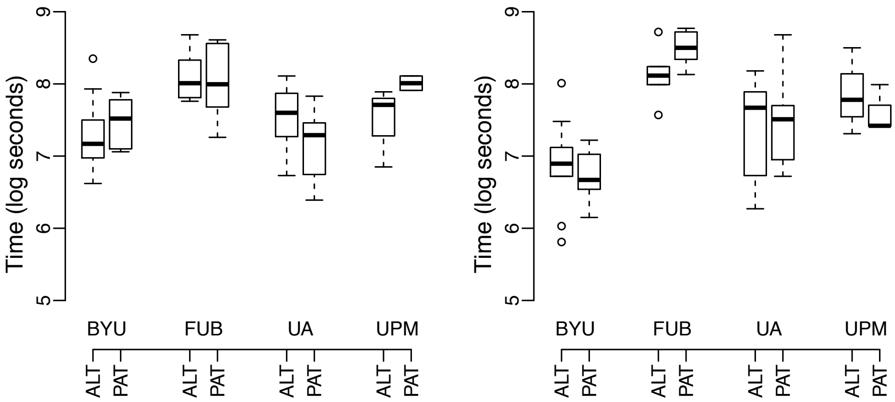
A joint replication can be considered a replication in two respects. First, it is a replication in that a specific study is being mirrored (or replicated) across multiple sites. Second, it is a replication in that it repeats the work of a previous study. However, the latter notion of replication is not an absolute necessity. An original experiment could follow the format of a joint replication, and in so doing, realize many of the same benefits. Nevertheless, Kraemer [119] strongly recommends that, as a general rule, single-site studies should precede multi-site studies in order to refine the protocol, as well as to justify the larger-scale effort.

A number of variations on our definition of joint replication can be imagined. For example, rather than having independent teams replicate the study at each site, the same team could travel between sites. We encourage researchers to explore such variations and document the corresponding tradeoffs; however, a comprehensive discussion of taxonomy is beyond the scope of this paper. Instead, we document the strengths and weaknesses of a specific type of joint replication (which we have tested in practice) and describe tooling options that can be used to mitigate its weaknesses.

Strengths of Joint Replication

The primary purpose of joint replication is to elicit, in a controlled setting, sufficient heterogeneity to study context, but without requiring an impractical number of participants or studies. E_joint successfully achieved this objective. First, for all tasks and response variables, the results significantly diverged across sites (e.g., see Figure 4.1). Second, and more importantly, the main effect was more significant at individual sites than when generalized across sites, which indicates the presence of cross-site variance (as opposed to random noise). Thus, E_joint evoked the problem of generalizability (which confronts the PatMain series of studies) within the context of a single controlled experiment, and it did so via a modest

Figure 4.1: Excerpt of E_joint time data showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.



(a) CO Task 1

(b) GR Task 1

number of participants and sites (53 and 4, respectively). Further, given the experimental controls, much of the variance was likely due to industry-relevant factors rather than to experimental artifacts.

Another strength of joint replication is that it involves conducting studies in parallel, rather than in series. The parallel approach creates two benefits for context analysis: 1) it eliminates the potential for confounding variables due to the passage of time; and 2) it allows for analysis to occur while the details of each site are still fresh in the minds of the researchers. Conversely, the series approach may be confounded by contextual evolution, and it requires reconstructing most context information from artifacts or distant memory. Tacit knowledge, in particular, is difficult to identify and account for across replications [189, 191]. However, via joint replication, such knowledge is largely shared across sites; and where it is not shared, it is at least current, and so can be explored during analysis.

A further benefit of the parallel approach is that it facilitates the use *triangulation* [47], which is a qualitative technique for ensuring validity in a subjective analysis. As we explain in Section 4.4.2, triangulation can be used to narrow the vast search space of potentially relevant context variables down to a handful of likely candidates. In this way, joint replication addresses the tractability problem identified by Dybå et al. [63]. Triangulation can be used to cross-analyze both series and parallel replications, but it is much less effective for series replications due to two factors: 1) current publication standards allow very little space for the documentation of context variables, and 2) from the perspective of *individual* studies, researchers cannot know which variables most need to be documented.

Other strengths of joint replication include:

- The results of a joint replication are more robust to researcher bias than those of a standalone study, especially if all of the research teams contribute to the combined analysis. Concerning the analysis, a single team may need to coordinate [200, p. 174], but all teams should at least provide peer review.

- Joint replication reduces the burden on any one research team of gathering sufficient participants. For example, of the 53 participants enlisted for E_joint, no more than 21 came from any one site (BYU=21, FUB=12, UA=14, UPM=6). Obviously, more participants are better, but as we show, joint replication can still work even for small sample sizes (in part because participants tend to be more similar within sites than across sites).

- Joint replication facilitates metrics that are comparable across sites. This is particularly important for subjective assessments, such as quality, which are difficult to fully standardize. In E_joint, for instance, the participants' solutions were graded by both the individual sites and by a central grader. Correlating the two sets of scores, we found only marginal correspondence. For one task at FUB the correlation was perfect, but most were in the range 0.25–0.75, and one was only 0.13.

- Joint replication facilitates peer review of a study's materials in a much more thorough manner than typically possible via the publication process. For example, in E_joint, FUB provided a replication framework, but to use that framework, BYU, UA, and UPM had to thoroughly test (and critique) it.

Weaknesses of Joint Replication

The primary weakness of joint replication is that it requires more time than does a standalone study. In total, E_joint required 18 months to complete (not counting downtime or time spent refining the methods). To some extent, the individual workload is reduced by the collaboration of several teams. However, that collaboration requires a non-trivial amount of effort in the form of recruiting, training, and coordinating. We discuss these issues further in the next subsection.

Another weakness of joint replication is the tendency of the research teams to recruit students. In E_joint, we made no restrictions on the type of participants used, professionals versus students, and not surprisingly, all four teams enlisted the latter. Fortunately for E_joint, the student demographic was useful because both prior PatMain studies employed professionals. However, if professionals are particularly needed, then appropriate guidelines may need to be established.

Tools and Other Lessons Learned

To encourage teams to participate in E_joint, we simplified the experiment (as described in Section 4.3.3). Effectively, we traded breadth of protocol for increased coverage of contextual variables and greater diversity in the population sample. We also implemented the experiment via a web portal, which provided several benefits: 1) it made execution of the experiment relatively trivial, thus lowering the barrier to entry for research teams; 2) it facilitated uniformity across sites; 3) it allowed participants to take the experiment on their own time, thus reducing scheduling constraints; 4) it allowed participants to work with their own tools in their own environments; 5) it reduced the burden of collecting and merging data across sites; and 6) it allowed researchers to create an arbitrary number of experimental instances, thus facilitating pilot studies.

The primary drawback of the web portal—which occurred as a consequence of the participants taking the experiment at a time/location of their own choosing—was an increased potential for measurement error and collusion. We did identify a few cases of measurement error (due to interruptions), but based on the post-questionnaire data, we were able to correct those errors. Also, after extensive investigation, we found no evidence of collusion. Thus, although we cannot make general statements about the take-home format, we believe its benefits outweighed the risks for E_joint.

To train the research teams at each site, we set up both a project website [75] and a mailing. On the website, we summarized the history of PatMain, the format of the joint replication, and operation of the web portal. However, we did not standardize the pre-experiment instructions to participants. In hindsight, we recommend standardizing these instructions because, in E_joint, we found evidence that they influenced the length of time the participants' were willing to spend on the experiment. We also recommend using a project mailing list. For E_joint, the mailing list facilitated asynchronous communication, which was ideal for teams spread across time zones. Via the mailing list we were able to answer questions, resolve concerns, and peer review the web portal.

4.4.2 Post-hoc Moderator Analysis

The second component of TCA is *post-hoc moderator analysis.* As we describe it, *moderator analysis* is a concept or framework for extracting context data from one or more studies and packaging that data for use in generalizing across studies. We believe that the concept of moderator analysis can help structure the investigation of context variables in software engineering, as well as facilitate such investigation to become a standard part of the experimental process.

The concept of *moderator variables* (or *moderators* for short) comes from sociology and psychology [13, 183]. A moderator is any explanatory variable that interacts with another explanatory variable in predicting a response variable [183, p. 624]. Thus, moderators are a type of context variable. However, for one variable to "moderate" another does *not* mean that it dampens the other's effect; rather, it means that the latter's effect varies in response to the former. If unaccounted for, the variance induced by a moderator can mask the effect of the variable with which it interacts. Further, if disjoint subsets of a moderator's range are represented in two different studies, then the two studies can produce inconsistent or even contradictory results.

Note that when we say "moderator," we are referring to a phenomenon inherent in nature—i.e., one variable moderates another's effect on some outcome independent of whether either is experimentally measured or statistically modeled. Any variable (or more loosely, factor or influence), previously known or unknown, measured or not, can be a moderator. The goal of a moderator analysis is to discover the most influential variables, sufficient to produce generalized conclusions. Thus, in such an analysis, we investigate as many potential moderators as possible, including variables that were not considered a priori.

Concerning the post-hoc nature of the analysis, such methods do inflate the chances of a type 1 error—i.e., testing relationships that were not specified a priori increases the risk of incorrectly concluding an effect exists. However, post-hoc methods are commonly used in other fields (e.g., medicine [209]) for uncovering relationships (e.g., moderators) that would otherwise remain undetected given only a priori methods. Thus they are appropriate for cases in which new insights are needed, or where the search space is vast and the researcher needs better-informed hypotheses. Nevertheless, it must be recognized that post-hoc analyses produce data-driven conjectures, which must be further tested—i.e., the output of a post-hoc moderator analysis is *not* a set of final conclusions, but a list of likely moderators for use in the next round of testing.

Importance and Tractability of Moderator Analysis

As Diesing explains, "We cannot test one hypothesis about flatworms, or students, or therapy patients. . . until we have learned many things about the interacting factors affecting their behavior" [55, p. 338]. In other words, identifying moderators is critical to the research process, and until we have done so, we cannot trust our conclusions. In fact, "it has been argued that the amount of progress in any discipline can be indexed by the degree to which its theory and research have considered the role of moderators" [183, p. 624].

Accordingly, and considering our participants are human, we should not be surprised to find disagreement in early replications. After all, replication is never just a test; rather, it is always "part of a larger search and discovery process" [55, pp. 337–338]. Thus, we should view contradictory results not as failures, but as an opportunity to explore moderators. Behavioral scientists, for instance, required twenty years and approximately four hundred replications to uncover the seventy context variables necessary to control flatworms in an experimental setting. Apparently, "Flatworms are very sensitive creatures" [55, p. 337].

But four hundred replications, really? And seventy moderators! To require so many replications would effectively label most problems in software engineering as intractable—not to mention that seventy moderators would be virtually impossible to apply in practice.

Not to be discouraged, the flatworms example is an *unlikely* worst-case scenario. A recent article in *Science* [139] describes a phenomenon called *sloppiness*, which is the nature of many complex systems (including biological systems and animal behavior, e.g., [5, 152, 202, 203]) to be predictable via a small subset of key variables. These "stiff" variables, as they are called, encapsulate the vital components of the system, such that the other "sloppy" variables can be ignored. Of course, not all phenomena will be predictable based on just two or three variables, but the research indicates that many should be. In fact, as we show in Section 4.5, many of the PatMain results are likely generalizable via two key moderators (*developer experience* and *pattern knowledge*); and, we were able to identify those moderators via only a handful of replications.

Additionally, in software engineering we have an advantage over the 1960s behavioral scientist. For us, moderator analysis is not a matter of figuring out how to properly grease a flatworm tank, just so the real learning can then begin. The majority of our participants are human developers—i.e., they *are* the population of interest. Thus, many of the moderators we explore in the laboratory will also be relevant to industry. In other words, moderator analysis *is* real learning, especially in the case of joint replication, for which the experimental protocol is controlled across sites.

Types of Moderators

We recognize two categories of moderators, which we term *native* versus *experimental*. Native moderators are those inherent to the domain of the study—including most demographic and environmental variables (e.g., a developer's skill or preferred IDE). Native moderators should be investigated, documented, and developed into theoretical/predictive models for use in research and industry. Experimental moderators are those that exist only as a consequence of the study itself (e.g., the treatment order in a repeated measures design). Experimental moderators should be investigated, documented, and better controlled as part of future studies, or at least accounted for during data analysis.

Qualitative Methods for Moderator Analysis

This paper is primarily about quantitative methods. However, qualitative methods are also needed, at the very least, to bootstrap the quantitative—i.e., we need a systematic way to select an initial set of variables to measure. Additionally, after a quantitative analysis is complete, especially if a joint replication was involved, a considerable amount of new information is available about non-measured and previously unknown variables. That data needs to be distilled into a list of potentially meaningful moderators and documented before it is forgotten. If the measured variables prove to be inadequate, such data will be critical for informing the next round of quantitative testing. Thus, qualitative analysis of moderators is needed both prior to and immediately following a quantitative analysis.

When qualitative analysis is conducted prior to a study, the source data must be taken from the existing literature. In the case of E_joint, for example, we used the published reports for E_orig [168] and E_repl [212]. Ultimately, for E_joint we measured only the most obvious variables which had been explicitly mentioned in the prior work (i.e., *developer experience* and *pattern knowledge*). We could have implemented a more elaborate selection process (e.g., a systematic literature review), but having never conducted a joint replication before, we felt it important to keep things simple.

When qualitative analysis is conducted at the end of a study, the experiences and outcomes of the study itself act as the source data. For instance, in our post-analysis of E_joint we used the workshop reports for the four sub-replications (BYU [126], FUB [167], UA [155], UPM [103]), the reports for E_orig [168] and E_repl [212], and email conversations between E_joint researchers.

Whether the analysis is pre or post, a number of qualitative methods are available (e.g., cross-case analysis, narrative synthesis, thematic analysis, grounded theory, etc. [47, 48, 104, 105]). Of the various methods, we find grounded theory to be particularly promising.

For the post-analysis of E_joint, we used a relaxed grounded-theory process involving coding, memoing, and the forming of categories (similar to that described by Charmaz [41]). The categories, which evolved throughout the process, ultimately became the variables we reported. At the midpoint of the process, each category (or variable) was supported by a set of evidence, consisting of memos, specific examples, and data references. We then selected for further analysis all variables that met the following two criteria: 1) the variable seemed theoretically meaningful, and 2) we had data available that could reasonably represent the variable. For the selected variables, we distilled the data to produce a set of stories. Finally, as a form of validation, we member-checked our findings by reviewing them with researchers from E_orig, E_repl, and E_joint. For additional information on grounded theory, see [1, 41, 45, 47, 82].

Overall, grounded theory was highly effective at helping us to view the data from new angles, as well as to perceive variables of which we were previously unaware. For example, we found participant *motivation* to be the most promising variable for further investigation. However, *motivation* had not been considered in either of the previous PatMain studies, nor did it occur to us until we had juxtaposed data from all three studies. Ultimately, we found evidence for eight potential moderators based on our qualitative analysis—including, *motivation*, *task difficulty*, *perceived time limits*, *cultural variation*, *IDE preferences*, *language barriers*, *clarity of task instructions*, and *compilation/testing expectations*.

Many of the variables we qualitatively identified for E_joint will hopefully be unnecessary to adequately generalize the PatMain results. However, such variables still need to be documented for several reasons: 1) despite the quantitative data we have on *developer experience* and *pattern knowledge*, some of the PatMain results are still not fully reconciled across the three studies; 2) we do not yet know which variables will be important in future studies involving new contexts; 3) studies related to PatMain (e.g., other design pattern studies) can likely benefit from the information; and 4) comparing such results across research topics can potentially reveal important insights regarding the general problem of experiment variability [63]. Also, in order to prevent the unnecessary repetition of work, documenting variables that are *not* likely moderators is as important as documenting those that are [63].

Other qualitative methods for identifying moderators, besides grounded theory, include two processes proposed by Juristo and Vegas [102, 104, 105, 211]. The Juristo-Vegas (JV) processes principally concern researcher communication at the time of a replication and the postmortem analysis of replication outcomes (for additional description of these processes, see Section 4.2). Both processes are compatible with the grounded theory approach we suggest. If used together, either the JV processes can be used to generate additional data for the grounded theory analysis, or the grounded theory analysis can be used to operationalize elements of the JV processes (e.g., the replication postmortem) [102, 104].

Cruzes and Dybå [48] have also identified several qualitative methods (e.g., cross-case analysis, narrative synthesis, thematic analysis), which are used in other fields to synthesize results across studies. Although some of the methods may need to be adapted, most appear applicable to moderator analysis. We leave further investigation of these methods to future work.

Lastly, *triangulation* of evidence is a standard qualitative technique [47] that can be used to further narrow a list of candidate variables. Triangulation is essentially the principle of comparing multiple and disparate data sources. Inasmuch as the data sources agree, we can have greater confidence in the findings. Triangulation underlies many qualitative methods (e.g., grounded theory, via the principle of *constant comparison*). Triangulation is also particularly applicable to joint replication, given joint replication's multi-site design.

Quantitative Methods for Moderator Analysis

Quantitative analysis of moderators requires statistically assessing the *interaction* between two or more explanatory variables. For a set of explanatory variables to *interact* means that the combined influence of the variables is not additive—i.e., the effect of each depends on the values of the others [3].

Both frequentist and Bayesian methods can be used to model interactions. Since many standard frequentist methods support interaction analysis—e.g., analysis of variance (ANOVA) [174], generalized linear models (GLM) [143], and linear mixed models [144]—we do not describe such methods in this paper. Instead, we compare the outcomes of the frequentist models we used for E_joint (which did *not* work well) with those of the Bayesian models (which did work well), and discuss why the Bayesian approach was superior. Then, in the next section, we describe the Bayesian models in detail.

For E_joint, we used linear mixed models [144], an extension of multiple linear regression, which adds the ability to represent blocking (or grouping) variables as random effects. Since the PatMain experiment defines multiple tasks for each of several programs, and since the hypotheses anticipate differing outcomes across the programs and tasks, assessing the effect of *variant* (i.e., the main effect) required modeling three-way interactions. In turn, to assess moderators required four-way interactions. Although 206 observations can be sufficient to model four-way interactions, given the heterogeneity of the results across sites, almost all of the interactions we tested produced insignificant p-values. Since many of the associated effect sizes were also large, the frequentist results for the moderator analysis were inconclusive. The data were simply too few relative to the variance and model sizes to obtain meaningful results via frequentist methods.

Conversely, the Bayesian models produced useful results despite the cross-site variance. Bayesian methods are superior for moderator analysis for two reasons:

1. *Bayesian models produce posterior probabilities (as opposed to p-values), which means the researcher can form conclusions even when statistical power is low.* In frequentist statistics, a p-value is the probability of obtaining data at least as extreme as those observed, assuming a null hypothesis is true. Conversely, Bayesian models yield *posterior* probabilities. A posterior probability is the probability that a hypothesis is true, given the data. Being a probability about the truth of a hypothesis, rather than about the likelihood of the data, posterior probabilities can be directly compared to determine which hypotheses are most likely [54]; such is not possible in frequentist statistics. Consequently, even when statistical power is low, the researcher can still identify likely moderators by simply comparing the appropriate probabilities.

2. *Bayesian models provide posterior samples, which means the researcher can use fewer parameters to estimate the same set of interactions, thus conserving statistical power.* Via Gibbs sampling [54], which we describe in Section 4.4.3, the researcher can generate samples from the joint posterior distribution of all parameters in a Bayesian model. Given posterior samples, the researcher can then use *marginalization* (also described in Section 4.4.3) to compute, from a single high-order interaction, results for all lower-order interactions and component terms [54]. Conversely, in a frequentist analysis, lower-order interactions and terms must be estimated by separate parameters, which increases the total number of parameters in the model and thereby reduces statistical power [174].

4.4.3 Bayesian Models

The third component of TCA is *Bayesian models*. In this section, we describe an additive-effects Bayesian model that we adapted from Felt [69], which we have found to be effective for moderator analysis. We describe the general form of the model below, using E_joint as an example. For a complete mathematical specification of the model, as applied to E_joint, see Appendix Z. For an additional example, see the work of Felt [69], in which he applies the model to the evaluation of a natural language processing technique.

Background

As previously mentioned, a *p-value* is the conditional probability of obtaining data at least as extreme as those observed, assuming a null hypothesis (H_0) is true. We can represent a p-value as $p(data|H_0)$, where p means "probability of" and the vertical bar means "given" or "conditioned upon". Thus, a p-value is *not* the probability that a particular hypothesis is true; rather it characterizes the likelihood of observing some set of data, given the assumption of a specific hypothesis. More generally, a p-value can be represented as $p(Y|\Theta)$, where Y represents the data (or observations) and Θ represents the hypothesis (or parameters). This formulation is often referred to as the *likelihood*.

Conversely, in Bayesian statistics we are primarily interested in *posterior* probabilities—i.e., $p(\Theta|Y)$, which is the probability distribution for some set of parameters, taking into account the occurrence of some set of observations. For example, in the case of PatMain, we may want to know the posterior probability that ALT takes more time (t) than PAT, i.e., $p(t_{ALT} > t_{PAT}|data)$.

Bayes' theorem describes the relationship between the posterior, $p(\Theta|Y)$, and the likelihood, $p(Y|\Theta)$, as:

$$p(\Theta|Y) = \frac{p(Y|\Theta)p(\Theta)}{p(Y)}$$

where $p(\Theta)$ and $p(Y)$ are often referred to as the *prior* and the *marginal*. The *prior* is the probability distribution of the parameters estimated independent of the observations—which can be thought of as "prior belief" or that which we believed prior to having viewed the data. Similarly, the *marginal* is the probability of observing the data independent of any hypotheses.

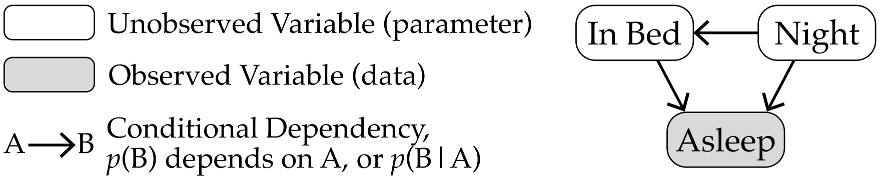
To estimate posterior probabilities for a given model, we do not compute Bayes' theorem directly. Instead, we use a standard algorithm, *Gibbs sampling* (with Metropolis or Metropolis-Hastings) [54, 69, 79], to jointly infer the posterior distribution of all parameters in the model. From a computer science perspective, Gibbs sampling is an estimation algorithm that runs on a network of nodes known as a *Bayesian network*—a directed acyclic graph, wherein nodes denote random variables and edges denote influence between the variables[2] (e.g., see Figure 4.2). Nodes (or variables) can be either observed or unobserved. Observed nodes are those for which we have data; unobserved nodes are the parameters for which we wish to estimate the joint (i.e., multivariate) posterior distribution.

Gibbs sampling approximates the joint posterior by iteratively generating samples from that distribution. We refer to samples generated by Gibbs as *posterior samples* and note that Gibbs infers posterior samples from data samples. A single sample consists of a set of values, one for each parameter in the model. Gibbs begins by assigning an arbitrary starting value to each parameter. Each time a sample is generated, the values of the new sample replace the previous parameter values. With successive iterations, the sampling chain moves toward higher density regions of the posterior probability space, eventually converging on the joint posterior distribution. The early samples, produced during the "burn-in" period, are typically discarded as unrepresentative of the target distribution.

Gibbs sampling is proven to approximate the joint posterior in the limit of the number of samples generated. The number of posterior samples required for Gibbs to converge depends on the degree of auto-correlation inherent in the model. Gibbs can be tuned to compensate

[2] Technically, the lack of an edge denotes conditional independence.

Figure 4.2: Example Bayesian network. The probability of *Asleep* depends on *In Bed* and *Night*.


Unobserved Variable (parameter)
Observed Variable (data)
A⟶B Conditional Dependency, p(B) depends on A, or p(B | A)
In Bed
Night
Asleep

for auto-correlation, but after reasonable efforts, the final solution is simply to average out the bias by generating more samples. Assuming the model is correctly implemented and convergence is achieved, the final set of samples are collectively as though they were drawn from the joint posterior. For efficiency in post-analysis, the final samples can be uniformly thinned (e.g., 500,000 samples could be thinned by a factor of 50 to yield 10,000 samples for analysis).

In the case of E_joint, we discarded (or "burned") the first 50,000 samples for each model. We then generated 16 million samples per model, which was the minimum number required for most of the models to converge. We used standard methods to assess convergence (e.g., trace plots); for information on such methods, see [46, 79]. Finally, for each model, we uniformly thinned the samples by a factor of 1600, thus leaving 10,000 samples for analysis. In the context of the PatMain study, 10,000 samples provided more than sufficient precision.

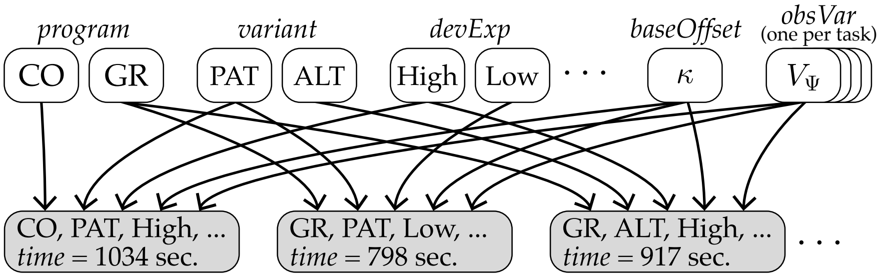
For Gibbs sampling the marginal, $p(Y)$, can be ignored [69, 79]; thus, to define a model, *we only need to specify a likelihood function and prior distributions.*

Model Overview

Figure 4.3 shows the proposed model depicted as a Bayesian network applied to the context of E_joint. Notice that each explanatory variable (i.e., *program*, *variant*, *devExp*, . . .) is represented as a set of disjoint categorical parameters. By *disjoint* we mean that each observation is conditionally dependent on (or connected to) one and only one parameter for each variable.

We use categorical parameters for two reasons. First, doing so allows the sub-ranges of any continuous variables to be estimated independently of one another, thus avoiding

Figure 4.3: Bayesian network depicting the proposed model applied to E_joint (as an example without interactions).



linearity assumptions. Second, fitting categorical variables simplifies results interpretation; to compare the influence that a set of categories have on the response variable, we simply need to compare the corresponding posterior distributions.

However, fitting a categorical model requires discretizing continuous variables (e.g., *devExp* and *patKnow*). Several standard methods are available to accomplish this task [22, p. 551], [118], such as clustering or equal-width/height binning. For E_joint we primarily used visual clustering to split the continuous variables into *low* and *high* categories. We used a binary categorization to keep the model sizes as small as possible (at least for the initial investigation).

The base offset (*baseOffset* or κ) in Figure 4.3 represents an average portion of the response variable common to all observations; thus, all observations are conditionally dependent on κ. For example, in E_joint, κ could be thought of as an average work *time* or *correctness* score, depending on the model. The observation variance (*obsVar* or V) is a set of disjoint parameters. However, unlike the explanatory variables, the cardinality of V can meaningfully be one, meaning all observations can be modeled with the same variance if such fits the circumstances of the study. In E_joint we included four variance parameters, one for each task; we did so because the tasks required differing amounts of time and longer tasks typically manifest greater variance.

Likelihood Function

Based on the setup shown in Figure 4.3, we define the likelihood (or density) of a single observation (y) as:

$$\begin{aligned} & y_{abc\ldots} | A_a, B_b, C_c, \ldots, \kappa, V_{\Psi \subseteq \{abc\ldots\}} \\ \sim \Delta\big(\mu, \sigma^2\big) \equiv {} & \Delta\big(A_a + B_b + C_c + \ldots + \kappa, V_{\Psi \subseteq \{abc\ldots\}}\big) \end{aligned}$$

where capital letters (A, B, C, $\ldots$) represent explanatory variables, lowercase letters (a, b, c, $\ldots$) represent parameter selectors for the explanatory variables (corresponding to the given observation, $y_{abc\ldots}$), and the *obsVar* selector, Ψ, is a composition of zero or more of the explanatory variable selectors (a, b, c, $\ldots$). For example, in the case of E_joint $\Psi = cf$ (as defined in Appendix Z), where $C = \mathit{program}$, $F = \mathit{task}$, and $|C| = |F| = 2$, thus producing four *obsVar* parameters.

Δ represents any distribution and should be selected as appropriate for the given response variable. For example, in E_joint we modeled the *time* data as a gamma distribution (strictly greater than zero with the possibility of a right skew) and the *correctness* data as a beta distribution (constrained to the range of 0 to 1). If the selected distribution is not by default parameterized by mean (μ) and variance (σ^2), the distribution can be re-parameterized using *method of moments* [54], as shown in Appendix Z for E_joint.

From the density formula above, it can be seen that σ^2 is simply the assigned variance parameter (V_Ψ) for the given observation, whereas μ is the sum of all other assigned parameters, including κ. Thus we refer to the model as an *additive-effects* model because each explanatory parameter adds to or subtracts from the average response (κ). The combination of parameters produces a variety of likelihood distributions. Each distribution fits itself to the observation it represents, while accounting for the influence of other observations bearing similar parameter assignments.

Finally, note in Figure 4.3 that the observations are conditionally independent of one another (i.e., no arrows directly connect them). Thus we can represent the likelihood (*lik*) of the data (Y) as the product of the density of each individual observation (y):

$$\begin{aligned} & \mathit{lik}(Y|A, B, C, \ldots, \kappa, V) \\ = & \prod_{y \in Y} p\left(y_{abc\ldots}|A_a, B_b, C_c, \ldots, \kappa, V_{\Psi \subseteq \{abc\ldots\}}\right) \end{aligned}$$

Prior Distributions

A prior distribution must be specified for each parameter in the model, including one for each explanatory parameter (A_a, B_b, C_c, $\ldots$), one for the base offset (κ), and one for each variance parameter (V_Ψ). The gamma and inverse gamma families of distributions are both commonly used to model variance because the domain of their non-zero ranges (i.e., their support) is always strictly greater than zero. As for the base offset and explanatory parameters, we recommend normal distributions unless the context of the study justifies non-symmetric noise. In E_joint, for example, we modeled the variance parameters with gamma distributions and all other parameters with normal distributions.[3]

Concerning the means and variances of the priors, we make two recommendations (both suggestions of Felt [69]): First, we recommend centering all priors for the explanatory parameters at zero, thus assuming no effect by default—the purpose being to let the data pull the posteriors left or right to indicate the effect of the parameter. Second, we recommend choosing large variances; doing so allows the posterior distributions to move more easily in response to the data, thus minimizing the weight of prior biases. The general purpose of both recommendations is to let the data drive the conclusions. Choosing broad priors does lead to broader posteriors and thus a greater chance for type 2 errors (i.e., failing to identify a true moderator). However, a type 2 bias is not altogether undesirable for post-hoc analysis, given

[3] Note that the base offset does *not* represent the distribution of the response variable. Rather, it represents a centering point or average for all observations, from which individual observations deviate, presumably, in response to the explanatory variables. Thus, symmetric noise is a reasonable general-case assumption.

that such analyses naturally tend toward type 1 errors (i.e., finding local patterns in the data that do not actually generalize).

If the analysis involves a replication study, then the priors can be estimated using data from previous iterations of the experiment. For example, in E_joint we used mean and variance data from E_orig to estimate priors. We simply assumed that our student participants would take a bit longer and exhibit greater variance than the professionals of E_orig. However, if the analysis does not involve a replication, or if all iterations of the experiment are to be included in the same analysis, then the priors must be estimated from scratch. In that case, the experiment could be administered to a few test participants from which priors could be estimated, or the researcher could simply choose what s/he believes to be reasonable priors. Generally speaking, choosing broad variances allows for a fairly large margin of error in selecting priors.

Lastly, note that priors should be selected in a non-biased way—which can be accomplished by selecting them before looking at the data or, as we did for E_joint, by having an external researcher select them. Additionally, it is advisable to have two or more researchers independently select priors, compute results based on each set of priors, and then compare the results to confirm that the analysis is not unduly sensitive to the precise choice of priors.

Interactions

Testing moderators requires modeling interactions between explanatory variables. Since all explanatory variables are represented as categorical parameters, interactions can also be represented as categorical parameters. Thus, interactions can be modeled in the same way as explanatory variables, but via a larger set of categories to represent all combinations of the component variables. For example, to test *devExp* as a moderator in E_joint, we had to model the interaction $variant \times program \times task \times devExp$, which required sixteen parameters: one each for ALT-CO-Task1-Low, ALT-CO-Task1-High, ALT-CO-Task2-Low, etc.

Letting I be an interaction term, the likelihood of a single observation (y), involving five explanatory variables (A, B, C, D, E), could be represented as:

$$\begin{aligned} &y_{abcde} | A_a, B_b, I_{cde}, \kappa, V_{\Psi\subseteq\{abcde\}} \\ \sim \Delta\big(\mu, \sigma^2\big) \equiv{} &\Delta\big(A_a + B_b + I_{cde} + \kappa, V_{\Psi\subseteq\{abcde\}}\big) \end{aligned}$$

where C, D, and E are the explanatory variables being interacted. Notice that C, D, and E do not appear in the model as standalone effects. In general, *if a variable is included in an interaction, it does not need to be included elsewhere in the model.* Since Gibbs produces samples from the joint posterior, we can simply use *marginalization* (described below) to compute, from a single high-order interaction, estimates for all lower-order interactions and component terms; thus, we can reduce the number of parameters in the model and thereby increase the model's statistical power [54, 174].

Ideally, all potential moderators should be modeled in the same interaction with the main effect. By doing so the researcher can compare them to determine which is most influential. However, if the moderators are tested in separate models, then directly comparing them is risky because a model's statistical power is influenced by both its size and by the distribution of observations over its parameters [174, p. 347].

In some cases, the number of parameters needed to represent all moderators in the same model/interaction can be too many relative to the size and variance of the dataset. In such cases the results will appear washed out, largely insignificant, and in particular, the posterior distributions will not have deviated much from the broad priors. The alternative is to test each moderator in a separate model, and thus to forgo quantitative comparisons between moderators. All moderators should still be included in every model, since they are believed to be relevant to the response variable, but only the moderator being tested should be included in the interaction. Thus, the models will only differ by which moderator is included in the interaction (e.g., see Table Z.1 in Appendix Z).

The situation can arise that only one or two parameters are *in*sufficiently supported by data, which is especially likely to occur in interactions. In these cases, the researcher may not want (or be able) to decompose the model further. However, if broad priors are used, then a data deficiency does not mislead the results. Rather, the posterior distributions affected by the deficiency simply approximate the broad priors, such that any comparisons involving those posteriors are relatively insignificant, as they should be. The consequence of insufficient data is thus the same as for using broad priors—a greater chance of type 2 errors. Incidentally, the problem of insufficient data highlights why both the volume *and the heterogeneity* of the data are important factors in moderator analysis.

Posterior Probabilities and Estimates

Each posterior sample generated by Gibbs corresponds to a set of values (one for each parameter in the model), such that the final result is a table, wherein rows represent samples and columns represent parameters. Most probabilities and estimates can be computed by simply comparing columns within the results table.

For instance, in the case of E_joint, *variant* could be represented as two parameters, one for PAT and one for ALT. To compute the posterior probability that the treatment (PAT) improves performance by reducing the work *time* (t), $p(t_{ALT} > t_{PAT}|data)$, we would calculate the percentage of table rows in which the PAT column's value is less than that of the ALT column. Similarly, to estimate the magnitude of the treatment's impact, we would compute, across all table rows, the average difference between the PAT and ALT column values.

In general, to compute the posterior probability of a certain condition, simply count the samples matching that condition and divide by the total number of samples. To compute an estimate, calculate the associated metric (e.g., a difference between columns) for each sample, and then average across all samples. If a more complex perspective is needed, posterior distributions (i.e., table columns) can be visualized using kernel density estimation

(KDE) [59, 192]—e.g., using the `density()` function in R [173]. KDE can also be used to visualize the distribution of an estimate by computing the associated metric across samples and then, rather than averaging, applying KDE. In general, examining distributions prior to computing summary metrics is a good practice in order to look for unexpected results (e.g., multi-modal distributions). Most other statistics can be computed in fairly intuitive ways—e.g., to compute an 80% confidence interval for a given distribution, simply find the limits about the mean within which 80% of the samples occur.

When computing statistics for variables involved in an interaction, however, we must first *marginalize* (or factor) out any irrelevant variables. Given the tabular form of the results, marginalization simply requires consolidating columns by concatenating them in a consistent order. For instance, in the case of E_joint, *variant* × *devExp* could be represented as four parameters: PAT-Low, PAT-High, ALT-Low, ALT-High. To compare program variants (ALT and PAT), we would concatenate the PAT-Low column to the end of the PAT-High column and, similarly, the ALT-Low column to the end of the ALT-High column—thus yielding two columns (or vectors of data), one for PAT and one for ALT. The concatenation order is not important, except that it be matched across the resultant columns. The new, larger columns can then be compared as previously described.

Results Interpretation—Frequentist vs. Bayesian

In frequentist statistics, p-values are significant when small—i.e., to reject a null hypothesis, the data must be unlikely under the assumption of that hypothesis. For posterior probabilities, however, large values are significant—e.g., 0.95 represents a 95% chance that the associated hypothesis is true, given the data.

Further, we typically require very small margins of error in frequentist statistics (e.g., $\alpha = 0.05$); otherwise the results are inconclusive. Conversely, for posterior probabilities, significance depends on the context of the problem. For example, we could view a posterior probability of 0.75 as providing only marginal confidence in the associated hypothesis, or we could interpret it as an expectation that the hypothesis will hold in 75% of similar cases. Under the latter interpretation, even relatively low probabilities can be meaningful (e.g., learning that a certain technique leads to better code could be useful, even if true only 75% of the time).

Assessment of Moderators

Recall that for one variable to moderate another means that an interaction exists, such that the effect of the latter (i.e., the *target*) varies in response to the former (i.e., the *moderator*). Based on this definition, a moderating relationship exists if and only if the target's significance is greater for at least one subrange (or category) of the moderator than when estimated independent (i.e., across the full range) of the moderator.

Thus, one way to assess whether a particular variable (i.e., a *candidate*) is a moderator is to compare the impact that the candidate has on the significance of the target when the two are interacted. Since the target's significance need only be greater for *at least* one subrange or category of the candidate, then the test can be performed based on two significance values: 1) the target's *general significance* (GS), as estimated across the full range of the candidate; and 2) the target's *maximum interaction significance* (MIS), being the maximum significance to occur for the target among all subranges or categories of the candidate. If $MIS \gg GS$, then the candidate likely moderates the target.[4]

To compute GS and MIS requires first understanding how to compute, given the output of the Bayesian model, the significance of an arbitrary explanatory variable. Since all explanatory variables are represented as categorical effects, the significance of an explanatory variable can be defined as the maximum significance to occur for any pair of its categories. In other words, if even one pair of categories differ with respect to the response variable, then the explanatory variable should be considered meaningful.

[4] Note that this test is a Bayesian implementation of *moderator-oriented subgroup analysis* [42, 130]; for additional guidelines concerning subgroup analysis, see the comments of Lau et al. [130, p. 823].

Letting X be an explanatory variable with three categories (a, b, c) and R be the associated response variable, we define the significance (sig) of X as:

$$\begin{aligned} sig(X) &= max\big(\ sig(a,b),\ sig(b,c),\ sig(a,c)\ \big), \text{ where} \\ sig(i,j) &= max\big(\ p(R_i > R_j|data),\ p(R_j > R_i|data)\ \big) \\ &\qquad\qquad\qquad\qquad \text{for } i,j \in \{a,b,c\} \text{ and } i \neq j \end{aligned}$$

Note that the directionality of the R_i/R_j comparisons (i.e., $R_i > R_j$ versus $R_j > R_i$) is irrelevant to determining significance because, for binary comparisons, non-significance is at 0.5; values such as 0.25 and 0.75 are equally significant. Thus, we can apply the max function in the computation of $sig(i,j)$ without losing information. The max function ensures that all sig values are in the range $[0.5, 1.0]$, such that greater significance is always denoted by larger values.

Given the above process, and assuming the Bayesian model's interaction term (denoted I_{cde} in Section 4.4.3) contains only the main effect and one candidate, GS can be calculated by marginalizing out the candidate and then computing $sig(X)$, where X is the main effect. Conversely, MIS would be computed *without* marginalization, by calculating the significance of the main effect separately for each level of the candidate, and then taking the max of those values. Remember, both GS and MIS should be computed from the same model, using the same interaction term, so GS can act as a baseline against which to compare MIS.

If all candidates are modeled in the same interaction, then all but one must be marginalized out prior to computing GS and MIS. In such cases, the GS (or baseline) computation will be identical for all candidates. Consequently, the MIS values can be compared to determine which candidates are most influential.

In some cases, the significance of the main effect may depend on additional, ancillary variables which are already known to be moderators. These variables do not need to be tested, but to correctly model the main effect, they must be included in the interaction. In such cases, GS and MIS must be computed separately for each combination of the ancillary variables. For example, in E_joint, the effect of *variant* depended on *program* and *task*, thus requiring four-way interactions, *variant* × *program* × *task* × *candidate*. Accordingly, for each candidate we had to compute four GS/MIS pairs, one pair for each task of each program. To identify moderators, we ultimately factored out *program* and *task* by taking the max of the four GS and MIS values, thus yielding a single pair of significance values for each candidate. However, for candidates found to be likely moderators (e.g., *devExp* and *patKnow*), we also examined their significance values on a per-task basis to be sure we understood the moderating relationship.

In addition to significance values, we recommend computing effect sizes, which can be helpful for gauging the practical impact of a moderator. To compute the effect size (es) for a given significance value (GS or MIS), identify the $sig(i, j)$ term from which the GS or MIS value is derived (i.e., the $sig(i, j)$ chosen by the *max* function in the calculation of $sig(X)$, defined above); then, for the same i and j, compute $es(i, j) = |R_i - R_j|$, where $R_i - R_j$ is the estimated difference in response between categories i and j. Note that, like significance, the direction of es is not relevant for identifying moderators, thus we take the absolute value. In E_joint, the (task-independent) GS and MIS values for *devExp* were 0.66 and 0.90, thus indicating a strong overall likelihood that *devExp* moderates *variant*. Since the corresponding effect sizes were 4.0 and 9.4 minutes (relative to 15–30 minute tasks), we could also conclude that *devExp* was practically meaningful.

Lastly, candidates found to be likely moderators will need to be examined in more detail to determine the exact nature of their relationships with the main effect. The simplest way to accomplish this level of analysis is to examine the values underlying GS and MIS (i.e., the values abstracted away by the *max* functions). However, when evaluating moderators (as opposed to identifying them), directionality matters, so significance values should not be compressed to the range $[0.5, 1.0]$ and effect sizes should retain any negative signs. In the case of E_joint, by examining significance values and effect sizes across tasks, we were able to quantitatively show that *devExp* and *patKnow* both *positively correlate* with the benefits of design patterns. As we discuss in the next section, it was this level of information that we needed in order to reconcile the contradictory PatMain results.

4.5 Application of TCA Results

Initial results for the three PatMain studies (E_orig, E_repl, and E_joint) diverged considerably. Since the divergences did not correlate with any one study, we could not resolve them by simply discarding a study as invalid. Also, since 28% of the tests were statistically significant at $\alpha = 0.05$ (i.e., far more than would be expected by random chance due to repeated measures), we could not conclude that the underlying effect was insignificant. Thus, to produce general conclusions we had to account for heterogeneity in the data.

From the moderator analysis, we learned that *devExp* and *patKnow* both interact with *variant* (at least for the participants in E_joint). Accordingly, we examined the PatMain results across studies to see whether the observed divergences could be explained by the moderating relationships identified in E_joint. For example, consider one of the hypotheses for CO task 1: $t_{PAT,Low} < t_{ALT,Low}$—meaning, when pattern knowledge is low, we expect PAT to take less time (t) than ALT. Results for this hypothesis were:

E_orig	E_repl	E_joint
-63% ($p<.05$)	$+13\%$ ($p>.05$)	$+58\%$ ($p<.05$)

Since ALT is the baseline for these results, it appears that E_orig confirms the hypothesis (being significantly negative), E_joint contradicts the hypothesis (being significantly positive), and E_repl is inconclusive.

However, noting that the participants' prior pattern knowledge was greatest at E_orig and least at E_joint, it becomes clear that the hypothesis is actually false in its strictest sense. For participants with (almost) no pattern knowledge (i.e., the students from E_joint), PAT takes longer. However, noting further that only 52% of E_orig's participants had prior experience with patterns, and then only on a limited basis, it also becomes clear that minimal knowledge is all that is needed to realize a substantial benefit (on average) from the Decorator pattern. Thus, the intuition behind the original hypothesis was largely correct.

Based on this type of analysis, we were able fully resolve conflicts for the CO program (Decorator pattern) and partially for the GR program (Abstract Factory pattern). Further, given only two key moderators, we were able to produce minimally-qualified, but highly generalized conclusions, such as:

> *CO program, Decorator pattern*: Using the Decorator pattern instead of a simpler solution is preferable during maintenance, as long as the developer has at least some prior understanding of the pattern. Given even minimal knowledge, the PAT variant is easier to modify; given sufficient knowledge, code comprehension is not negatively affected.

Conclusions of this form are sufficiently simple to be useful in practice, and yet generalize to a broad population. For instance, the above conclusion generalizes across all three PatMain studies, involving 126 participants from five universities and twelve software companies. Such a high level of generalization would not have been possible without the moderator analysis.

4.6 Limitations

In this section, we discuss limitations of the TCA method. We organize this discussion in terms of TCA's three components: joint replication, post-hoc moderator analysis, and Bayesian models. We also discuss limitations of our process for applying TCA results.

Joint replication requires considerable time and resource commitments relative to a standalone study. In particular, it requires recruiting, training, and coordinating multiple research teams. It also necessitates a more complex analysis because it involves multiple sub-replications. As discussed in Section 4.4.1, tooling can mitigate, but not eliminate these challenges.

Post-hoc analysis is useful for exploring experiment instability. However, it inflates the chances of a type 1 error—i.e., the chances of incorrectly concluding an effect exists. Thus its findings must be tested in future studies. That said, note that applying post-hoc results to past or future studies does provide a certain level of validation (similar to a test set in machine learning).

Moderator analysis requires modeling large interactions, which reduces statistical power. This problem can be mitigated, but not eliminated by using Bayesian methods. Bayesian methods allow interactions to be modeled via fewer parameters. They also yield posterior probabilities, which (unlike p-values) can be directly compared; thus, the researcher can form conclusions even when statistical power is low.

Bayesian models often require a sampling process to approximate the desired posterior distributions. Thus, they can require more time and resources to estimate than frequentist models. For example, in the case of E_joint, the processing time and memory requirements were substantial due to parameter coupling, which necessitated 16 million samples per model to achieve convergence. Additionally, we implemented E_joint's models from scratch in R, which turned out to be complicated and time consuming. As an alternative, we suggest using a Bayesian inference library such as Stan [201]. Stan is open-source, provides a convenient syntax for representing models, and includes interfaces for R, Python, and the command-line. Stan also uses Hamiltonian Monte Carlo instead of Metropolis-Hastings, which can converge more quickly under certain conditions (e.g., that of coupled parameters).

Applying TCA results to a set of contradictory replications requires matching interaction relationships with observed divergences. Admittedly, the process we describe in Section 4.5 is somewhat ad hoc, and so may benefit from further systematization (which effort we leave to future work). However, in the spirit of post-hoc analysis, such approaches are not without merit. The primary purpose of post-hoc analysis, after all, is to help researchers uncover new relationships; and ultimately, new relationships can only be validated by additional observations and/or tests.

4.7 Conclusions

The idea that hundreds of context variables must be understood in order to resolve contradictory experimental results is likely more hyperbole than fact. Via the TCA method, we have been able to resolve significant divergences in the results across a series of design pattern studies, the final conclusions of which rely on only two key moderators. Moreover, we were able to identify the necessary moderators via only a handful of studies involving a modest number of participants.

The TCA method includes three components: joint replication, post-hoc moderator analysis, and Bayesian models. Essentially, moderator variables encapsulate and formalize contextual information as part of the experimental framework. In turn, joint replication, post-hoc analysis, and Bayesian statistics provide an empirically and statistically-grounded process for identifying and articulating moderator variables. Together, the concepts yield a research strategy which, at the very least, shows promise for facilitating generalizability across closely replicated experiments. However, with additional development, it could potentially lead to higher-level generalization as well.

Outcomes of the TCA method can include: a ranked list of variables found to moderate the main effect; a list of other variables found to not moderate the main effect; a reformulation of the original theory/hypotheses taking into account the identified moderators; an updated description of conditions necessary for a study to be considered a *close* replication; and a set of generalized conclusions, qualified by both the level of generality achieved and by recommendations for addressing points of weakness in future work.

Additionally, by attempting to explain divergent results, rather than eliminate them, TCA naturally leads to explanatory theory. The benefit of such theory is that it allows researchers to shift the goal of replication from results *reproducibility*, which often fails, to experiment *predictability*, which can more easily cross contextual boundaries. Ultimately, as our results indicate, effective generalization (at least for highly variable contexts) requires testable theory about context. TCA facilitates the development of such theory by enabling the investigation of context variables in greater detail than previously possible.

4.8 Acknowledgments

We are grateful to: Paul Felt for sharing source code and advice concerning his Bayesian models; and Gilbert Fellingham for expert advice on Bayesian methods.

Chapter 5

The Humanity of Science versus the Complexity of Reality: A Theoretical Study of Replication and Knowledge Production

> *It is no different with the faith with which so many materialistic natural scientists rest content nowadays, the faith in a world that is supposed to have its equivalent and its measure in human thought and human valuations—a "world of truth" that can be mastered completely and forever with the aid of our square little reason. What? Do we really want to permit existence to be degraded for us like this—reduced to a mere exercise for a calculator and an indoor diversion for mathematicians? Above all, one should not wish to divest existence of its rich ambiguity: that is a dictate of good taste, gentlemen, the taste of reverence for everything that lies beyond your horizon.*
>
> —Nietzsche (1887) [157, p. 335]

Context. In empirical software engineering, we frequently discuss the importance of replication. However, most published studies are not easily replicable and/or present results that are not reproducible. Such difficulties are not surprising considering that 1) we inherit our notions of replication from other scientific fields, and 2) our literature contains no systematic syntheses of that external theory. Given recent and past concerns with the effectiveness of replication in our field, addressing this theoretical gap in the literature would seem a natural next step to take. **Objective.** In this paper, we investigate the theoretical foundations of replication (by exploring a limited perspective on several epistemological theories and their relevance to replication). In particular, we focus on the relationship between replication and knowledge production. **Methods.** For our source material, we investigate theories from other disciplines—sociology, psychology, linguistics, architecture, philosophy, and the natural sciences—which we distill into a consolidated *Theory of Conceptual Frameworks.* We use this integrated theory in a preliminary analysis to examine replication concerns in empirical software engineering. **Results.** Via the theory of conceptual frameworks, we define the role of replication and show that replication is far more than a simple process of validation; rather, it is a central mechanism for learning in science and, as such, is always practiced in some form, recognized or not. Furthermore, a key objective for science is not just to define methods for executing isolated replications, but to systematize replication as a governing process for cross-study synthesis. To this end, our findings suggest a framework for developing a unified taxonomy of replication types, on the basis of which we present a disambiguated working model of replication. Lastly, our findings discourage drawing fixed lines around replication methodologies in such a way as to discredit novel experimentation. **Conclusions.** Further work is needed to comprehensively integrate these findings with the empirical software engineering literature.

5.1 Introduction

The critical necessity of replication to knowledge production has become a tenet of faith in empirical software engineering. The literature is replete with statements such as, “replication is a key feature of experimentation in any scientific or technological field” [104, p. 295], “replication is a basic component of the scientific method, so it hardly needs to be justified” [108, p. 219], or more simply, “The value of experimental replications is evident. . . ” [40, p. 1]. As a research community we clearly believe that replication is essential to knowledge production. But *why* do we believe this? On what grounds—specific to our domain—do we justify a need to replicate?

Most references to the above question in software engineering cite general science texts or the work of Brooks et al. [31, 32], which in turn relies on assertions from other fields. Karl

Popper [164], for instance, is an important figure whose name frequently appears in discussions of replication. However, Popper's philosophy, as with all epistemological arguments, is subject to flaws and tradeoffs [55]. Ultimately, the problem is *not* that we take ideas from other fields, but rather that we take them at face value. We have been citing other fields on replication for nearly twenty years [31], and yet our literature still includes no systematic reviews of that material, much less a comprehensive synthesis of the underlying theories.

As Mulkay explains:

> What is to count as a "replication" depends on scientists' theories about the phenomena under study and on their view of the factors which may influence the observational situation. Consequently, as theoretical frameworks evolve and experimental techniques develop, so the way in which the general criterion of "replicability" is applied in any given area necessarily alters… When we look at all closely at the observational criterion of "replicability" we find that it is almost a mere formality. *It has no content until it is put into practice in a particular scientific context.* [154, pp. 50, 52, emphasis added]

Thus, the meaning of "replication" necessarily depends on the particular scientific theories under investigation, as well as on the specific context (including frameworks and tools) of those investigations (for practical examples, see [154, pp. 50–52]). Accordingly, to rely on the philosophical tenets of other disciplines without reflection and without adaptation almost certainly leads to methodological problems and substandard progress as a scientific field.

In software engineering, for example, several factors inhibit and/or discourage replication studies, including: 1) a persistent perception that replication studies are less valuable than original studies; 2) the frequent unavailability of data sets; 3) insufficient detail in published reports to allow replication [40, 96]; and 4) the unavailability and/or inoperability of research tools, rendering precise replication impractical [31, 32, 108, 189, 190]. Additionally, innovation in our field, though not necessarily more rapid than elsewhere [194], complicates replication. For instance, programming languages, hardware, and development environments have all changed significantly over the last decade. Studies that rely on such factors are both more difficult to replicate and, when their conclusions are taken seriously, are in greater need of replication. Further, exact (or strict) replication is infeasible for all but the most small-scale, artificially constrained studies, and even in those cases we often cannot reproduce results [103, 126, 137, 155, 167]. Although some work addresses these issues (e.g., [19, 102]), we still struggle to find practical methods for conducting *useful* replications (i.e., those that produce transferable knowledge) [49, 102]. In particular, most attempts to generalize results across replications fail in the face of excessive heterogeneity (e.g., [42, 147]).

The net effect of these difficulties is that we, as a research community, continue to pursue new problems to the neglect of developing comprehensive, generalizable, and transferable knowledge. As Basili explains, "the balance between evaluation of results and development of new models is still skewed in favor of unverified proposals" [19, p. 456], [98]—and, it would seem, badly skewed. Though the rate of replication has risen slightly over the past few years (mostly an increase in internal replications), a recent literature survey by da Silva et al. [49] reports—from a search of over 16,000 articles[1]—finding only 96 papers (or 0.6%) describing a replication. Further, Cruzes and Dybå [48] report finding *synthesis* to be the single most challenging (and neglected) component of systematic review studies; nearly half of the studies they surveyed did not contain any synthesis at all, and of those that did, two thirds performed only basic thematic and narrative synthesis.

In light of these issues, we are left with a nagging philosophical question—one which has not been adequately addressed in the empirical software engineering literature: *Why are we (or should we be) so concerned in principle about something that has as yet proven ineffective in practice?* In other words, is it possible that we have been asking the wrong questions all along? For instance, although replication is (claimed to be) important for other

[1] Sources included: ACM Transactions on Software Engineering Methodologies, IEEE Transactions on Software Engineering, Empirical Software Engineering Journal, Information and Software Technology Journal, Int'l Conference on Software Engineering, Int'l Conference on Evaluation and Assessment of Software Engineering, Int'l Symposium on Empirical Software Engineering and Measurement, Int'l Workshop On Replication in Empirical Software Engineering Research, ACM Digital Library, IEEE Xplore Digital Library, ScienceDirect, Scopus, JSTOR, and the literature reviews of Almqvist [4], Carver [40], and Sjøberg et al. [197].

fields, perhaps it is not appropriate to the unique circumstances of software engineering? Perhaps something altogether different awaits discovery that would be more suited to building knowledge in our field? On the other hand, *if replication is truly indispensable to knowledge production*, then perhaps we simply need a more effective conceptualization of it. As Juristo and Vegas have indicated, "we might be dealing with the issue of [software engineering] experiment replication from too naive a perspective" [102, p. 356].

5.1.1 Contributions

In this paper, we make two contributions to the empirical software engineering literature:

1. We distill external theory on replication (from the fields of sociology, psychology, linguistics, architecture, philosophy, and the natural sciences) into a consolidated theory of knowledge production, which we refer to as the *Theory of Conceptual Frameworks*.
2. We present a preliminary application of the theory of conceptual frameworks to problems of replication and knowledge production in empirical software engineering.

5.1.2 Scope

Although the latter half of this paper discusses practical aspects of replication, this paper is, first and foremost, theoretical. We do not present a set of specific methods for replication; rather, we explore theories of knowledge production, which are foundational to replication. *The material in this paper is important because it addresses fundamental assumptions that underlie our current beliefs about replication.*

Additionally, we need to be clear that the assertions presented in this paper are based on a <u>*limited*</u> dataset. We incorporate a diverse set of ideas from a number of scientific fields, but by no means do we account for the whole of the philosophy of science. This paper is in *no way* intended to be a comprehensive treatise on truth, knowledge, epistemology, or ontology. Furthermore, we recognize that not all philosophers would agree with all of the assertions we make. However, that is okay! Our purpose in writhing this paper is not to provide a final answer (because such is simply not possible at this point given the magnitude of the topic we address). Instead, our purpose is to put a stake in the sand, to establish a starting point for discussion, and to show that much can be gained by investigating the theoretical foundations of replication. Currently, textbooks are simply not being written on the topic, in any field, even in “hard” sciences such as physics [186]; rather, what little discussion is available tends to be spread out across the scientific literature in piecemeal fashion.

Thus, the theoretical position we develop in this paper represents a *limited* perspective on only a handful of epistemological theories. The *theory of conceptual frameworks*, in particular, is simply a lens through which to view a set of problems. In fact, the theory ultimately predicts its own imperfection, in that it cannot be both true and absolutely comprehensive since it dictates that all theories must necessarily be selective and positioned. Therefore, we ask the reader not to judge too harshly the underlying theory and remember that we agree with you—this is not the final word!

With respect to validity, note that we espouse a pragmatic perspective. Ultimately, our primary concern is not whether reality is precisely as the theory in this paper purports it to be; rather, our primary concern is whether or not the theory leads us to useful insights about replication. Inasmuch as the latter objective is accomplished, we consider the exercise to have been a success (and as we show in Section 5.6, the ideas in this paper do in fact lead to several key insights about replication which inform a range of practical replication concerns).

Lastly, note that we do not weigh down the discussion with caveats at every assertion to reiterate the above-described limitations. Instead, we recognize once, at the outset, that each assertion, though frank in its position, is only valid within the framework of the evidence presented in this paper (and even within that framework, many of the assertions can be variously stated or otherwise interpreted).

5.1.3 Structure of this Article

In Sections 5.2 and 5.3, we outline the *Theory of Conceptual Frameworks* and explore its relationship to truth and knowledge. In Section 5.4, we introduce Georg Simmel's concept of the *transcendent character of life*.[2] Simmel's transcendency, which we take from sociology, provides a bridge between the theory of conceptual frameworks and the concept of replication. In Section 5.5, we apply Simmel's transcendency to map the theory of conceptual frameworks onto the concept of replication. In Section 5.6, we use the theory of conceptual frameworks in a preliminary analysis to examine replication concerns in empirical software engineering. In Section 5.7, we discuss additional ideas that need further development, and finally, in Section 5.8, we conclude.

5.2 Conceptual Frameworks

Central to the thesis of this paper is the concept of conceptual frameworks. Conceptual frameworks (or schemes) are important because they fundamentally impact how we plan studies, create metrics, interpret and adjust data, and conceive theories. As such, the basic ideas of these frameworks show up in theoretical discussions from many fields. We borrow the term *conceptual framework* from an epistemological discussion by Paul Diesing [55], which explores how social science functions in practice. The term, as used by Diesing, refers to Thomas Kuhn's work [129] on the nature of scientific revolutions. To understand the meaning behind the term, it is helpful to briefly review Kuhn's thesis.

According to Kuhn's study of the natural sciences [129], the evolution of a science involves four stages—protoscience, normal science, crisis, and revolution:

- In the protoscience stage, a science has not yet established a paradigm. Scientists disagree about everything from fundamental principles to methods and techniques. Theories at this stage are based primarily on philosophical speculation and few if any

[2] Georg Simmel (1858–1918) was a German scholar, considered one of the founders of sociology, particularly for his theories on individuality and social forms.

are grounded in empirical evidence. Most empirical observations are eventually exposed as trivial, irrelevant, or mistaken.

- At some point, one theory emerges dominant, and all competing theories either die out or are branded as unscientific. During the process, a school forms around the winning theory, and as the exemplar of the new school, the founder's work is epitomized and imitated—a new discipline is born. Eventually, the school codifies the founder's work into a *Disciplinary matrix* [55, p. 57], which then serves as a curriculum for students of the discipline. Students are taught to apply the founder's principles to problems from the protoscience stage in order to prove that the theory can explain those phenomena. Unsolvable problems, however, do not disprove the theory; they can be set aside to wait for the theory to mature.

- Once normal science sets in, scientists become specialists, puzzle solvers and technicians whose job it is to apply the patterns of the disciplinary matrix to one problem after another. Communication flows freely in this stage. Scientists understand each other well because they share a common disciplinary matrix. They are now able to build on each other's work.

- Despite the progress of normal science, some phenomena persistently refuse to be explained by the core theory. These anomalies require increasingly divergent changes in the theory in order to explain them, which in turn break established explanations for other phenomena. Eventually, persistent anomalies create a sense of uneasiness, leading scientists to question the fundamental tenets of the discipline. If unresolved, the crisis stage can result in revolution, in which several new guiding frameworks are proposed, a small-scale protoscience era ensues, and the winner establishes a new discipline, after which normal science then resumes.

The important concept to take from Kuhn's model is that of the *paradigm*, which as Kuhn describes, dominates a scientific field once normal science has set in. Kuhn likens paradigm shifts to the perceptual experience of a gestalt switch [129, pp. 85, 111–114, 150]—that is, each paradigm is an entirely new way of perceiving the world, rather than a mere adjustment in perspective. Consequently, two people, each immersed in a different paradigm, may look at the same phenomenon and literally see different things. In Diesing's words, Kuhn is saying "that perception of facts is an active process in which our perceptual categories and expectations shape and organize stimuli into facts," and since "it is hardly possible that any one *conceptual framework* perfectly imitates the real structure of nature. . . the progress of science requires a series of revolutions in which the blind spots or distortions of one set of concepts are corrected by a later set" [55, pp. 60–61, emphasis added].[3] Thus a conceptual framework is like an ontology in that it establishes what the world looks like, including what things can exist in the world and what the relationships between those things are.

However, one important distinction, relating to the scope of the term, separates Kuhn's use of the concept from Diesing's. In Kuhn's analysis, the concept is termed a *conceptual box* [129, pp. 5, 152], referring to the external ontological model that is imposed on the individual by an established discipline. Diesing, on the other hand, implicitly widens the scope to include the internal model as well, which may be partly a product of the imposition of a discipline, but which is in fact a broad composition of numerous individual perceptual experiences. Thus a conceptual box that spans a scientific discipline—i.e., a Kuhnian paradigm or worldview—may be thought of as a collective, roughly shared conceptual framework, reified in the physical world through the operational mediums of the discipline (e.g., in the form of textbooks and conference proceedings). Kuhn, therefore, is speaking primarily of the external notion of paradigms, whereas Diesing's term hints at the internal, individual ontology as well. In this paper, when we refer to conceptual frameworks, we intend Diesing's meaning, but with our primary focus being on the internal model.

Based on this discussion, *a conceptual framework can be thought of as a personal classification scheme, a taxonomy of the entities that can (and by exclusion, cannot) exist*

[3] This concept relates to Popper's work [164]—in particular to the role that falsification plays in progressing science toward truth. For a brief summary of Popper, see Diesing [55, pp. 29–54].

in the world, including the relationships between those entities, according to and within the mind of a particular individual. Conceptual frameworks are so foundational to thought and consciousness that we are only aware of the "things" which our frameworks explicitly identify; objects outside an individual's conceptual framework do not, and indeed *cannot*, exist for that individual (either conceptually or perceptually) without first an adjustment to the framework. Conceptual frameworks are therefore independent of conscious epistemological persuasions because they interact with us at a deeper level. They preempt our conscious efforts to come to know truth, and as we discuss in the next section, they both enable and inhibit our knowing the world. A paradigm or worldview, on the other hand, is a roughly shared conceptual framework, often manifesting itself concretely in some form of action and/or substance.[4]

We have already hinted (with support from Diesing) that conceptual frameworks act as selective filters, through which we interpret and manage the complexity of the world—that is, they enable us to see the world, in turn facilitating our coming to know the world. The idea that we parse and filter the world through a classification scheme in order to understand it reaches back to Kant's a priori categories of cognition [193, pp. xv, 6–7]. However, contrary to Kant's categories, conceptual frameworks (as we define them) "inform not only the cognitive realm, but any and all dimensions of human experience," and "they are not fixed and immutable, but emerge, develop, and perhaps disappear over time" [193, p. xv]. Further, this evolution of frameworks applies both to individuals and to the social constructions of groups.

[4] The relationship we outline between science and the individual (based on the concepts of paradigms and conceptual frameworks) parallels Simmel's distinction between society and individuals. Of this distinction Levine comments, "...Simmel manages to sidestep the age-old controversy between sociological realism and nominalism: whether society is an entity with a character and properties of its own or whether society is merely a name for the aggregation of a multiplicity of individual actions. Simmel rejects both views, arguing on the one hand that the idea of a societal substance, of an independent collective entity (*Volkseinheit*), does not correspond to anything that can be observed. The place where all societal events occur is within the minds of individuals. On the other hand, there is a way of looking at those psychic events that is not psychological, but that is able to perceive the synthetic *realities* of processes and relations through which individuals act upon and with one another" [193, pp. xxxiii–xxxiv]. The same applies to science, that it can be viewed both as an independent entity and as a cumulative system (or collection) of interacting individuals; both constructions represent important perspectives. In this paper we primarily discuss the latter perspective, as driven by conceptual frameworks within individual minds. Many historians and philosophers, however, treat science (with good reason) as an independent entity—e.g., see work by Mulkay [154]. We could, of course, analyze replication from that perspective as well.

We borrow this particular extension of Kant's ideas directly from Simmel's theory of social forms. Social forms, as Simmel describes them, roughly express the idea of conceptual frameworks, though Simmel never explicitly identifies them as such. Simmel's ideas on forms are worth noting because they are instrumental in bridging conceptual frameworks with his transcendence theory and, subsequently, with replication.

To define *forms*, it is helpful to distinguish them from another of Simmel's concepts, *contents*. In Donald Levine's words, contents are "those aspects of existence which are determined in themselves, but as such contain neither structure nor the possibility of being apprehended by us in their immediacy," whereas forms are "the synthesizing principles which select elements from the raw stuff of experience and shape them into determinate unities" [193, p. xv]. More simply stated, forms are the categories of cognition that provide shape to a world of structureless contents. Content is that which exists, forms are the frameworks by which we represent (and enact) that existence.[5] As we mentioned previously with respect to conceptual frameworks, Simmel's forms are identical to Kant's a priori categories of cognition, except that 1) they inform all aspects of human existence (not just the cognitive) and 2) they can change over time. With respect to the former distinction, Simmel shows that forms structure not just the individual's perceptions and understandings, but also the individual's actions and interactions with other individuals [193, p. 24]. Thus Kant's cognitive categories are a subset (functionally) of social forms, which in turn are a subset of conceptual frameworks—that

[5] To make the distinction between forms and contents even more clear, note Simmel's references to similar dichotomies in other sciences, particularly geometry [193, pp. 27–28], which Levine summarizes as follows: "Grammar studies the pure forms of language, abstracted from the linguistic contents through which these forms come to life. Logic and epistemology study the pure forms of knowing, abstracted from the multitude of cognitions of particular things. Geometry studies pure spatial forms, abstracted from the physical objects which embody them. Sociology as the science of social forms relates to the special sciences which deal with the various contents of social life, such as economic activity, sexual behavior, education, law, or religion, much as geometry relates to the various physical sciences. The great difference between the two is that spatial forms, though only approximated in physical objects, can be isolated, absolutely identified and logically derived in geometric thought, whereas, owing to the fluctuations and complexities of social life, the status of the forms of social interaction which sociology abstracts is more ambiguous. In general, however, sociology may be regarded as the geometry of social forms" [193, pp. xxiv–xxv].

is, Simmel's forms are conceptual frameworks applied specifically to the context of social interaction.[6]

In the next section, we address the relationship between conceptual frameworks, truth, and knowledge—that is, the nature of reality, as opposed to our apprehensions and representations of it—including the impact that conceptual frameworks have on validity, sense perception, and thought. Once these connections are established, we introduce Simmel's transcendency and explore it with respect to replication in empirical software engineering.

5.3 Relationship to Truth and Knowledge

To understand the relationship between conceptual frameworks, truth, and knowledge, we turn to Simmel's discussion of *fragmented man*, which attempts to explain how individuals relate to and compose society. Simmel, of course, was a sociologist, but as we show, his ideas can be extended to conceptual frameworks, and ultimately, to replication.

The basic idea of Simmel's fragmented man is that in the act of knowing others, we form artificially-unified, distorted pictures of them. These distortions, as Simmel puts it, "are not simple mistakes resulting from incomplete experience, defective vision, or sympathetic or antipathetic prejudices," but "are fundamental changes in the quality of the actual object perceived" [193, p. 9].[7] Simmel explains that we generalize the singular individual and fit him to a category, which *fitting* is forced upon us because, first, we cannot fully understand an individuality from which we differ, and second, we cannot fully objectify an individuality to which we are at all similar. Simmel therefore points out that we require both similarity to and dissimilarity from other individuals in order to fully know them—*which is to say that in this world we must be close to an object to truly understand it, and yet simultaneously distanced from that object in order to form it objectively in our mind.* Being unable to exercise

[6] For a deeper discussion of Simmel's contents and forms as they relate to philosophy (Kant, etc.), see Simmel's essay, "On the Nature of Philosophy" [218, pp. 282–309].

[7] Although Simmel divides these fundamental changes into two types—the societal and the anti-societal components of our individualities—we treat them together because the underlying mechanism, which relates to conceptual frameworks, is the same.

both criteria fully, we simply cannot completely know the individuality of another, nor, by extension, that of ourselves [193, p. 10]. Consequently, we think of individuals as both singular and as generalized others. Our conceptions blur contours, adding external relationships (to other objects) to the uniqueness of the individual. And it is by this unconscious act of blending fragmented images that we construct unified wholes, wholes which we are otherwise unable to obtain directly from nature. The net effect of this process is, in Simmel's words, "to know a man, we see him not in terms of his pure individuality, but carried, lifted up or lowered, by the general type under which we classify him" [193, p. 10].

The key point to take from Simmel's analysis is that the process of knowing the world, at its core, is subject to the same limitations as that of knowing other individuals (who are a part of that external world). Thus Simmel's ideas about knowing individuality, which derive from and culminate in the creation and exercise of forms, have their parallel in conceptual frameworks. Of fragmented man, Simmel concludes, "just as we compensate for a blind spot in our field of vision so that we are no longer aware of it, so a fragmentary structure is transformed by another's view into the completeness of an individuality" [193, p. 11]—or in other words, we not only construct individuals (our knowing them) within existing forms, but we simultaneously construct new forms out of those individuals (our knowing the world). *In terms of conceptual frameworks, we construct objects (or our understandings of them) within and based upon our existing conceptual frameworks, but we also constantly reconstruct our frameworks based on those same objects.*[8] Thus *knowing* never yields a fixed structure or image—knowledge is not static, nor can it be fully decontextualized. As with Simmel's forms, which are always incomplete representations of the world, conceptual frameworks are both fundamental and prerequisite to knowledge. They not only reflect the world, they also fundamentally impact our perception and knowledge of it. Ultimately, conceptual frameworks both *enable* and *bound* our knowing the world, and it is this point—a culmination and

[8] This idea is reflective of Jean Piaget's work on the psychological development of children [72, 81]. See Section 5.3.3, including Footnote 14 for further discussion.

generalization of Simmel's ideas on individuality—that we clarify and defend throughout the remainder of this section.

Because the argument in this section is long, we have divided it into four subsections, each of which begins with a summary of key ideas (italicized) followed by a detailed discussion. It may be helpful to first scan the subsection summaries, followed by the concluding subsection (5.3.5), before reading the detailed discussions.

5.3.1 Knowledge is Discrete and Partial

The world is continuous, whereas our representations (i.e., conceptual frameworks) of it are discrete, fragmented, and partial. We aggregate these (Simmelian) fragments into artificially reunified continuities of thought. If conceptual frameworks are discrete, partial representations, then all knowledge, being built on conceptual frameworks, is discrete and partial.

According to Nietzsche, to be a thinker means one "knows how to make things simpler than they are" [157, p. 205]. Nietzsche explains that "in truth we are confronted by a continuum out of which we isolate a couple of pieces, just as we perceive motion only as isolated points and then infer it without ever actually seeing it. The suddenness with which many effects stand out misleads us; actually, it is sudden only for us. In the moment of suddenness there is an infinite number of processes that elude us" [157, p. 173].

The notion that the world is continuous, but that our perceptions of it are always discrete, fragmented, and partial is the topic of a book by Eviatar Zerubavel [221]. Zerubavel's book is the most inclusive reference to this concept that we find in the literature. Although we do not agree with all of Zerubavel's arguments, his basic premise echoes the ideas of many thinkers (such as Nietzsche, quoted above), and so is worth some discussion.

Zerubavel's thesis [221] argues the non-concreteness (or "boundarylessness") of reality, and the fact that as humans we must apply classifications (or boundaries) in order to see, think, and act. In order to discern a "thing," Zerubavel explains, we must delineate it from that which lies around it, or in other words, "we must distinguish that which we attend from that which we ignore" [221, p. 1]. Zerubavel gives the examples of color-blindness tests and camouflage, pointing out that when entities are not clearly delineated from their surroundings, they are effectively invisible. Boundaries give a "thing" its meaning apart from all other "things." Thus it is the boundaries, first and foremost, that enable perception, recognition, and consequently, cognition. As examples, Zerubavel shows how we carve up the world to form our existence—chunks of space (e.g., countries, cities, homes), blocks of time (e.g., hours, days, weeks), frames of reality (e.g., the experience of professional work as opposed to work around the house), mental fields (e.g., the delineation of sacred from profane), and so forth. We revisit this idea of a world demarcated by artificial boundaries in the discussion of Simmel's transcendency. The key point here is that the world outside of us is not so divided:

> You get the illusion that all those parts are just there and are being named as they exist. But they can be named quite differently and organized quite differently depending on how the knife moves... It is important to see the knife for what it is and not to be fooled into thinking that [things] are the way they are just because the knife happened to cut it up that way. [161, p. 66]

Despite what our minds tell us, the world is not simply a set of insular objects waiting to be apprehended. To the contrary, it is an enigmatic flowing of existence, continuous, and infinitely more complex than that which we can represent. It presents itself "not in black and white, but, rather, in subtle shades of gray, with mental twilight zones as well as intermediate essences connecting entities" [221, p. 62].

The idea that knowledge is the discretization of a world fundamentally different from our knowing it is not foreign to Simmel's thinking. For Simmel, the discretization of reality (or rational thought) is an obvious backdrop to his theory of social forms. In Simmel's own words, social forms are "one of the organizers and flexible instruments with which the mind gives structure to the mass of all that is mass which, in its immediate unity, is structureless" [218, p. 288]. Expanding on this idea, Edward Sapir observes that speech elements, such as "house," are symbols "not of a single perception, nor even of the notion of a particular object, but of a 'concept,' in other words, of a convenient capsule of thought that embraces thousands of distinct experiences and that is ready to take in thousands more" [184, p. 13]. He goes on to explain that the flow of speech "may be interpreted as a record of the setting of these concepts into mutual relations" [184, p. 13]. Thus we can think of *concepts* as aggregations of (Simmel's) fragments of experience into discrete chunks, which we interrelate to form a unity. One's system of mental concepts, with their relationships one to another, constitutes one's conceptual framework—though conceptual frameworks obviously run deeper than articulated language. The key point to take from Sapir is that concepts are containers for experience; through the application of concepts (i.e., taxonomy) we impose boundaries on a world of otherwise continuous experience, thus creating the insular "object." Further, these judgments typically occur at a pre-conscious stage in the thought process. Consequently, our conceptual frameworks act as information filters (or discretizers), which simplify the infinitely complex and continuous reality of the world that surrounds us into manageable, disambiguated and insular chunks.

What then are the consequences to knowledge production of this discretization of reality? At the very least, if classification is fundamental to rational thought, then science (as we currently define it) depends on discretization to understand a continuous reality. Consider again the example of language, which breaks concepts up into paragraphs, sentences, and ultimately insular words and word-parts, each of which carries individualized meaning relative to the world. We combine *discrete* words to convey a *continuity* of thought. If the world is continuous while language is discontinuous, but the world is understood by us through the discontinuity of language (and rational thought)—that is, all of our devices for guaranteeing truth, such as logic, are based on a thing that is fundamentally different from the thing which it is meant to characterize—then how can we ever believe that we really understand the world at all? In Nietzsche's words, "But how could we possibly explain anything? We operate only with things that do not exist: lines, planes, bodies, atoms, divisible time spans, divisible spaces. How should explanations be at all possible when we first turn everything into an *image*, our image!" [157, p. 172].

5.3.2 Conceptual Frameworks Selectively Represent Reality

If all knowledge is discrete and partial, then how are we to know truth?—we must espouse a "correspondence theory of truth" with "selective representation of reality" [90]. There are many equally valid ways of representing the world within our conceptual frameworks. Because the world is more complex than our frameworks have the capacity to represent, we require many snapshots to even begin capturing it, each taking advantage of and conveying specific features, though always suppressing others. Thus many snapshots, each characterizing the same phenomenon, may contain valid features different from (though related to) all the others. Nevertheless, not all ways of representing the world are equally "good"; there does exist a reality and a correspondence to that reality. In Poggi's words, "There is more than one way of skinning a cat—but there are not that many ways!" [163, p. 26]. This is the tradeoff or paradox of taxonomy, and by extension, of conceptual frameworks—that they are always both valid and invalid at the same time. Taxonomies are always both necessary and yet insufficient to represent the world, which fact insinuates the following concept of validity: Under the theory of conceptual frameworks, we are forced to espouse a pragmatic view of the validity of knowledge. For pragmatism [45, p. 4], [55, pp. 75–103] validity is in the consequences; valid knowledge is useful knowledge, knowledge that leads to good consequences, consequences that work.[9]

Man as knower is distinctly different from man as something known; according to Simmel, the latter is made by nature and history, but the former *makes* nature and history [193, pp. 3–4]. This is precisely the conclusion that we have come to in exploring knowledge as the discretization of reality. As knowers we cannot duplicate reality because "the whole of existence in the real sense is accessible to no one [without first being] assembled from

[9] The definition of *work*, of course, is contextual and value-based; thus science is political.

those fragments of reality which are the sole data. . . ” [218, p. 285]. Ethnography admits this fact freely—for example, Emerson et al. convincingly argue that ethnographic fieldnotes are always selective, positioned, and stylized [66, pp. 66–107]—and so too are all other man-made representations in science, both conceptual and physical. Perspective always matters, and we cannot escape context entirely.[10] Poggi explains the situation well:

> It is not given to humans to live and act on the basis of a direct and full apprehension of reality itself. . . First, as Kant has taught us, reality as such is unknowable, except within the framework of subjective processes which select and order cognitively various phenomenal aspects of it. More generally, the reality within which human beings exist is too complex, ambiguous and threatening to enable them to orient not only cognitively but practically to that reality in its totality. They can only locate themselves and function within it by ordaining it subjectively, initially by attaching meaning to particular, *selective* aspects and moments of it. Such meaning, however, is not intrinsic to any given aspect, objectively built into it. It is unavoidably up to the human beings themselves to attribute such meaning, *to interpret reality by considering some aspects and moments of it as central to their own existence, and others as peripheral or indifferent.* [163, p. 21, emphases added]

Thus, as a consequence of the notion that reality is too complex to represent in its totality by any individual, we are forced to *selectively* conceptualize it.

A corollary to this conclusion is that there are multiple valid ways of representing the world through our conceptual frameworks. For starters, Simmel maintained that to understand the world we must explore it at multiple levels of specificity [193, pp. xxxiv, 28]. In fact, one of Simmel’s primary techniques of apprehension (as is evident in his writing) is to analyze phenomena from an array of conceptual distances. This technique, of course, seeks to

[10] Accordingly, logical empiricists of the early twentieth century have (quite unintentionally) demonstrated that absolute truth cannot be discovered by systematic aggregation of finite, provable facts and that metaphysics cannot be removed entirely from science [55, pp. 3–28].

balance closeness, similarity, and intimacy against distance, difference, and objectivity—both perspectives being simultaneously necessary to knowledge, as we have previously discussed. Outlining this aspect of Simmel's methodology, Levine states, "It is wrong to think that a more detailed view of something is thereby 'truer' than a more distant view… Each distance has its own correct picture and its own margin for error" [193, p. xxxiv]. Thus seemingly incommensurable worldviews can often be reconciled (or at least understood) by clearly identifying their goals, as well as the various levels of abstraction from which they approach those goals. It is as though our conceptual frameworks each represent only a partial, two-dimensional snapshot of a three-dimensional world. It takes many snapshots to represent the complexity of a three-dimensional space, especially when that space is also changing over time. Each snapshot may characterize reality accurately, but differently, each with a particular margin of error. The error, of course, is an artifact of construction. Assimilation of these fragments yields a comprehensive and useful, though always incomplete, representation—one that (hopefully) corresponds with reality, but is never able to perfectly mimic it.

The idea of multiple valid frameworks introduces an important question: Does selective representation mean that "truth" is effectively independent of reality? To answer this question, we find the position of Martyn Hammersley helpful. In a methodological analysis of ethnography [90, pp. 57–78], Hammersley discusses three different criteria schemes for judging ethnographic research—quantitative, aesthetic, and no criteria possible. After arguing why he believes each is inadequate, he then presents his own criteria: validity and relevance. With respect to these criteria, Hammersley states,

> In conceptualizing validity, I adopt a position of what might be called subtle (as opposed to naive) realism. I use 'validity' as a synonym for what seems to have become a taboo word for many social scientists: 'truth'. An account is valid or true if it represents accurately those features of the phenomena that it is intended to describe, explain or theorize. Assumed here, then, is a correspondence theory of truth, but the correspondence involves selective representation rather than

> reproduction of reality. Furthermore, I recognize that we can never know with certainty whether (or the extent to which) an account is true; for the obvious reason that we have no independent, immediate and utterly reliable access to reality. Given that this is the situation, we must judge the validity of claims on the basis of the adequacy of the evidence offered in support of them. [90, p. 69]

Although truth (or valid knowledge), for us, can be no better than selective representation, it nevertheless can (and should) correspond with reality. Thus despite the selective, constructed nature of our conceptual frameworks, we cannot simply make up whatever we want. Returning to Zerubavel, "[W]hile boundaries and mental fields may not exist 'out there,' neither are they generated solely by our own mind. The discontinuities we experience are neither natural nor universal, yet they are not entirely personal either. We may not all classify reality in a precisely identical manner, yet we certainly do cut it up into rather similar mental chunks with pretty similar outlines" [221, p. 76]. Though the world offers (quite possibly) infinite variation across many levels, it still is as it is (i.e., *absolute truth does exist*), and our representations of it (i.e., our knowledge) can be more or less faithful. Thus some knowledge simply does not work in any practical sense, whereas other knowledge works spectacularly well. Assuming we want knowledge that works, the pursuit of knowledge is not entirely ambiguous, relative, or meaningless, and everything does not devolve into radical relativism. A substantial object does exist for science to pursue; that much, at least, is clear from the history and application of its achievements.

Thus, the primary struggle of science, which is inherent to the pursuit of knowledge, is fundamentally one of representation—not only a struggle to formulate representations for that which we now see, but to come to know those things which we are currently incapable of representing.[11] Indeed, at any given moment the things we seek are all things for which we currently have no representation; *they are hard to see precisely because we have never previously differentiated them from that which surrounds them.* Science is the disciplined

[11] For an interesting characterization, see Phillip Armour's taxonomy of the "orders of ignorance" [7, 8].

process of expanding (or simply evolving) our representational power. We look and see what we already know how to see, plus a bit more, which bit seems strange and confusing and may not appear to fit. In fact, because our frameworks are partial representations, there is (and always will be) a lot that does not fit, but much of which we are blind to. By repeated exposure, however, the tiny fragment of strangeness that we sense pushes us out of our current rut, forcing realignments of our conceptual frameworks. We are then able to see new things and new relationships, and as the process continues, our knowledge expands and evolves.[12]

Although knowledge can expand through a process of repeated experience, it nevertheless remains grounded in the representational limitations of our conceptual frameworks—always partial, always selective. This is the tradeoff or paradox of taxonomy, that it is always necessary for cognition, and yet never sufficient to know anything fully. To illustrate this tradeoff, consider work by Lidz and Fleck [134], who present a theory of schizophrenia disorders. In their theory they discuss the impact that a breakdown of proper category formation has on one's relationship to reality. They state, "Experience is continuous, whereas categories are discrete, so that boundaries [for the normal individual] must be established between categories by repressing what lies between them. . . " [134, p. 431]. In the schizophrenic, however, intracategory repression is often impaired and ethnocentric thinking is primary. According to Lidz and Fleck, the results of this state "are far-reaching and devastating. The categorizations. . . developed are now seriously impaired, and with the filtering function of categories lost, inappropriate associations intrude and derail thought and communication" [134, p. 430]. This example is important because it illustrates two points: first, conceptual frameworks enable and facilitate cognition, and second, as taxonomies, conceptual frameworks simultaneously

[12] The extent to which our knowledge expands versus evolves is a difficult question. The theory of conceptual frameworks insinuates (presumably) that knowledge cannot expand forever—at the very least because conceptual frameworks are always partial. Each expansion, it would seem, must also create some new limitation. Since every framework acts as a screen or filter, and if all frameworks are by nature and purpose representationally limited, then all frameworks blind us to some degree. Thus to have our attention drawn to one thing is also to have it drawn away from another. This is the tradeoff of taxonomy, which we discuss in the next few paragraphs. Nevertheless, it certainly seems reasonable to conclude that knowledge must expand to some degree, at least as we progress from infancy to maturity. For the purposes of this paper, it is sufficient to conclude that knowledge acquisition involves a balance of expansion and evolution, wherein absolute perfection is impossible, at least within the lifespan of any particular individual.

impose boundaries onto that same cognition. It is true that "[w]hen analytic thought, the knife, is applied to experience, something is always killed in the process" [161, p. 70], and it is also true that this tradeoff is fundamental to human cognition.

Consequently, within every conceptual framework (as with all taxonomy) *intra*category differences are minimized or ignored, whereas *inter*category boundaries are conceptually inflated. For this reason, being a native Hebrew speaker, the sociologist Eviatar Zerubavel reports struggling for years to distinguish jam from jelly [221, p. 63]. This is also why for a carpenter a nail is not just a "nail." There are in fact hundreds of different kinds of "things," all commonly called nails, used to affix materials. To a carpenter, all and yet none are really nails; the layman, on the other hand, probably has no idea what a "twelve penny sinker" is. Similarly, to a child, not only can socks be gloves, but the primary difference between a bus and an airplane may be the fact that one does not pay the driver upon entering an airplane [221, pp. v, 65]. *Thus, knowledge-building is ultimately an optimization problem. The selective representation of reality means that we must optimize our conceptual frameworks to fit reality as best we can, while still addressing the requirements that are placed on the resulting knowledge.* Referring back to Hammersley's definition of validity, valid knowledge is knowledge that works, given a specific conceptual focus, a specific set of goals, and a specific level of abstraction. Knowledge is always contextual.

We conclude this subsection by asserting that no conceptual framework can exist in absolute, perfect fidelity to the world. All conceptual frameworks are approximations of reality, each optimized to fit specific features of the system it describes, while discriminating against others. Thus, many different conceptual frameworks can simultaneously and reasonably represent the same particular aspect of the world, each pivoting on specific foci, each pressing specific goals. This means that both our conscious knowledge and our conceptual frameworks underlying that knowledge are always subject to reinterpretation. As Simmel puts it, "The one-sidedness of the great philosophies brings to most unambiguous expression the relation between the infinite ambiguity of the world and our limited capacities for interpretation" [193, pp. 357–358]. Conceptual frameworks are selective representations of reality.

5.3.3 Conceptual Frameworks are Prerequisite to Knowledge

We construct all knowledge in the form of, and on the foundation of, conceptual frameworks. Though individually they are imperfect, conceptual frameworks are both necessary and prerequisite to all thought and experience. In Nietzsche's words, "We see that science also rests on a faith; there simply is no science 'without presuppositions"' [157, p. 281].

The construction of knowledge is fundamental to the human experience. This is true in part because our perception is never perfect or all-encompassing, which we specifically discuss in the subsection that follows. In a more subtle way, however, humans *must* construct knowledge because our means of representation are fundamentally limited relative to the complexity of nature—which limitations seem to be as much strengths as they are weaknesses. We refer again to the tradeoff or paradox of taxonomy, and by extension to conceptual frameworks. Appealing to functionalism, it is as though the purpose of conceptual frameworks is expressly to limit our view so that we may think rationally about things. The sea of all that is reality is vast; without focus, the experience of it would be overwhelming and useless. As templates (or models), conceptual frameworks are the tools by which we focus thought, thus allowing it to be *rational*. Conceptual frameworks enable us to identify (or reconstruct) specific relationships in nature, so we can examine and leverage those relationships.

Viewed this way, it becomes clear that there is an act of human construction involved in knowledge. As Thomas Schwandt, an expert in qualitative inquiry methodology from the field of Education, describes:

> In a fairly unremarkable sense, we are all constructivists if we believe that the mind is active in the construction of knowledge. Most of us would agree that knowing is not passive—a simple imprinting of sense data on the mind—but active; mind does something with these impressions, at the very least forms abstractions

> of concepts. In this sense, constructivism means that human beings do not find or discover knowledge so much as construct or make it. We invent concepts, models, and schemes to make sense of experience and, further, we continually test and modify these constructions in light of new experience. [187, p. 237]

Nature in conjunction with previously acquired conceptual frameworks may persuade us to notice one particular relationship over another, but ultimately it is a human act, even if subconscious, when we form a relationship into our frameworks. As we have discussed, disparate frameworks may each represent truth simply by focusing on contrasting relationships; there is no external, objective reason why we have to look at the world from one particular perspective, over all others. For this reason, pragmatists "are careful to emphasize that acts of knowing embody perspectives. Thus, what is discovered about 'reality' cannot be divorced from the operative perspective of the knower, which enters silently into his or her search for, and ultimate conclusions about, some event" [45, p. 4]. Nevertheless, just because all knowledge is positioned does not mean that truth is somehow fake—at least not in the sense of validity that we have discussed. A particular conceptual framework should indeed be considered invalid if it does not correspond with reality, but a framework is not necessarily invalid simply because its articulation involves value judgments and human decisions.

It is reasonable at this point to ask whether these arguments apply equally well to the natural sciences as they do to the social sciences (especially considering that many of our sources come from the social sciences). Perhaps the natural sciences have a better hold on truth, due to some invariant or deterministic property, which inheres in their subject matter? In this regard we refer back to Kuhn's analysis—*which is entirely based on the natural sciences*—of which Diesing makes the following critical observations:

> From [Kuhn's] generalized history, we can induce several characteristics that seem to be needed for any halfway decent science. First, to deal systematically with anything we need a conceptual scheme that will organize our perceptions and guide our dealings with our subject matter. Kuhn is not saying that if you do not have a

> conceptual scheme you will make a poor scientist, so you ought to get yourself one next week. No, he is saying that everyone deals with the world in an organized fashion and that is part of being human. Kuhn's lesson is rather that we should be aware that we make our own data. They are not sense data that come to us directly from nature. Kuhn emphasizes direct perception, but he could easily have added measuring instruments and experimental apparatus constructed according to the specifications of our conceptual scheme—also statistical techniques for adjusting raw data, and data quality control criteria. We cannot make data as we please, but nature is flexible enough to fit into many different conceptual schemes, more or less. [55, pp. 61–62]

Kuhn's history clearly demonstrates, contrary to the perceptions of popular culture, that the natural sciences do not offer knowledge which is necessarily truer than that of the social sciences. This is because all sciences ultimately rely on the capacity and nuances of human cognition. In other words, the limitations of knowledge inhere in the nature of conceptual frameworks, which are cognitive tools exclusive to human reasoning rather than properties of a given realm of reality.[13]

Because knowledge is constructed on the foundation of and out of the same material as conceptual frameworks, conceptual frameworks are necessary and prerequisite to cognition. This deduction is consistent with Jean Piaget's extensive studies on the psychological development of children [72, 81]. According to Piaget's theory,[14] a critical barrier to learning is

[13] Similarly, with respect to physics, David Grandy [89] contrasts the discrete, discontinuous, and selective nature of human knowledge against the continuous, unitary flow of reality. In his writings he undercuts the everyday assumption that the world consists, first and foremost, of discrete parts, which parts are then thought to be *fully represented* by part-privileging taxonomies and modes of discourse. As Grandy demonstrates, the dialectic between continuity and discontinuity is a key feature of *quantum reality*. For additional examples from the natural sciences, see work by Mulkay [154].

[14] Piaget describes intellectual development in terms of two general principles: *organization* and *adaptation* [81, pp. 18–22]. Organization is "the tendency common to all forms of life to integrate structures, which may be physical or psychological, into higher-order systems or structures" [81, p. 18]. Piaget refers to these structures as *schemas*. Schemas develop out of and facilitate our interaction with the environment; thus schemas help us to interpret and understand the world. According to Piaget, schema formation begins in infancy with the development of organized patterns of behavior (for Piaget, intelligence is critically grounded in behavior and motor interaction with the environment), but later develops into organized patterns of

developing a mental *schema* that approximates a novel concept (or experience) sufficiently to enable that concept to be incorporated. If the schema differs too significantly from the concept then assimilation is not possible and accommodation of the schema is unimaginable—the gap is simply too large to close and confusion, misperception, or even impercipience is the likely result. However, once the student possesses an approximate schema (relative to the concept or experience), s/he can begin to assimilate the concept, thereby articulating relationships, as well as testing and, when necessary, modifying the schema to accommodate novel relationships [81, pp. 221, 223–224]. Anyone who has taught an introductory course has seen this process in vivo. Cognitive progress in a novel space depends on a mental bootstrapping that, once accomplished, significantly accelerates apprehension and (to an extent) originality.

The notion that conceptual frameworks are prerequisite to cognition helps explain creativity. Nietzsche points out that originality is "[t]o see something that has no name as yet and hence cannot be mentioned although it stares us all in the face" [157, p. 218]. "The way men are," he goes on to say, "it takes a name to make something visible for them" [157, p. 218]. Nietzsche believed that the names we give things are often more relevant to experience than what those things actually are. Create new names, he said, and in the long run you create new things [157, pp. 121–122]. Thus creativity can be defined as one's flexibility at manipulating one's own conceptual frameworks. Accordingly, one's capacity to be creative is determined by one's mental dexterity—the ability to flow between conceptual frameworks, trying on different classifications of things to find those that are particularly useful and

thought as well. Intellectual adaptation is accomplished through the complementary processes of *assimilation* and *accommodation*. As with their biological counterparts, assimilation involves incorporating elements of the external world into one's own structures, whereas accommodation occurs as the individual modifies his or her structures in response to external demands. Thus on the psychological level, assimilation means incorporating concepts by fitting them within an existing schema; accommodation, on the other hand, entails incorporation by modification of the schema itself. In general, the two processes are inextricably intertwined, and both interact to produce a net effect. In Ginsberg and Opper's words, "Faced with novel experiences, the child seeks to assimilate them into his existing mental framework. To do this, he may have to adjust and modify the framework, or accommodate to the requirements of novel experience. New knowledge is never acquired in a discontinuous fashion, but is always absorbed into preexisting structures in such a way that prior experience is used to explain novelty, and novelty is adapted to fit previous experience. Mental development is more than a mere accumulation of isolated and unrelated experiences; it is a hierarchical process with the latter acquisitions being built upon, and at the same time expanding upon the earlier ones" [81, p. 225].

meaningful (in a human sense), those that accentuate useful properties of the world and how it works. Discussing the importance of creativity in research, C. Wright Mills states, “The sociological imagination, I remind you, in considerable part consists of the capacity to shift from one perspective to another, and in the process to build up an adequate view… You try to think in terms of a variety of viewpoints and in this way to let your mind become a moving prism catching light from as many angles as possible” [149, pp. 211, 214].[15]

Simmel, of course, also discusses the idea that our conceptual frameworks are fundamental to thought [193, pp. xxxvii, 6]. Recall that for Simmel, nature “consists of innumerable contents which are given determinate identity, structure, and meaning through the imposition of forms which man has created in the course of his experience” [193, p. xxxii]. Simmel’s forms, which are conceptual frameworks, artificially “form immediate data into new objects, but they alone make the given world into a knowable world” [193, p. 12].

5.3.4 Conceptual Frameworks Influence Perception and Thought

Conceptual frameworks configure or color our perception and silently persuade our thinking. They impact what we can and cannot see, which begins to touch on why this matters for replication.

In Simmel’s words:

> It is impossible for man to begin entirely at the beginning. He always finds, within or outside himself, a reality or a past which supplies a basis for his conduct, a starting point, or at least something which is hostile and must be destroyed. In

[15] Mills goes on to say, “It is this imagination that sets off the social scientist from the mere technician. …there is an unexpected quality about it, perhaps because its essence is the combination of ideas that no one expected were combinable, say, a mess of ideas from German philosophy and British economics. There is a playfulness of mind back of such combining as well as a truly fierce drive to make sense of the world, which the technician as such usually lacks. Perhaps he is too well trained, too precisely trained. Since one can be *trained* only in what is already known, training sometimes incapacitates one from learning new ways; it makes one rebel against what is bound to be at first loose and even sloppy. But you must cling to such vague images and notions, if they are yours, and you must work them out. For it is in such forms that original ideas, if any, almost always first appear” [149, p. 211].

> just this way, *our knowing is also conditioned by something which is "already there,"* by realities or by inner laws. [218, p. 282, emphasis added]

Similarly Nietzsche states, "As soon as we see a new image, we immediately construct it with the aid of all our previous experiences. . . even in the realm of sense perception" [157, pp. 173–174]. We have already pointed out that knowledge is conditioned—Kant's a priori categories of cognition, Simmel's forms, and of course, conceptual frameworks. In this subsection, we examine the role that this conditioning plays in knowledge production, as well as its impact on thought. To begin the discussion, consider an observation from Edward Sapir's introduction to the study of speech:

> As soon as the word is at hand, we instinctively feel, with something of a sigh of relief, that the concept is ours for the handling. Not until we own the symbol do we feel that we hold a key to the immediate knowledge or understanding of the concept. . . *And the word, as we know, is not only a key; it may also be a fetter.* [184, p. 17, emphasis added]

In some sense, language can be viewed as an integral part and/or manifestation of conceptual frameworks—not the canonical languages, such as English, Russian, or French, but the individualized languages of use and familiarity, which derive from the official, but are intricately personalized. We refer again to the distinction between the individual scope of conceptual frameworks as opposed to the broader scope of conceptual boxes or paradigms. One's *functional* language is unique—at the very least a unique union of subsets, but arguably far more conceptually personalized still. As a partial manifestation of one's conceptual frameworks, functional language tells us something about the impact that conceptual frameworks have on thinking and knowing.

To be clear, we do not argue for an extreme interpretation of Benjamin Whorf's ideas [215] on language and cognition (often referred to as the Sapir-Whorf hypothesis). We do not believe that the limitations of one's official language, be it English, Spanish, French, or Eskimo, *determine* what one can and cannot think (nor do we believe Whorf intended such an extreme position). Of this point, Geoffrey Pullum's arguments with respect to "the great Eskimo vocabulary hoax" [172] are illustrative. The Eskimo language does not, he explains, necessarily have vastly more words for snow than does, say, English, and Eskimos are not necessarily aware of—or able to think about or "see"—more types of snow than are English speakers. Language does not strictly determine or constrain one's thoughts. Rather, one's unique *language of familiarity* (as opposed to one's official language) predisposes one's thinking (rather than determining it), and the underlying mechanism, which drives that predisposition, is the individual's conceptual framework. Thus a botanist easily sees a rich variety of plant species in what otherwise appears to the layperson to be simply a field. It is not that the layperson *cannot* see that there are many different plants in a field; s/he is simply not *predisposed* to differentiate them—a point which is even more poignant in the social world and of non-physical phenomena [43, p. 3].

Language is not the only example of ontological presuppositions that affect perception. In an experiment by Bruner and Postman [34][16] subjects were asked to identify a series of five playing cards. Each card was presented repeatedly (with gradually increasing exposure times from 10 to 1,000 ms) until correctly identified. Most of the subjects produced an initial guess for most of the cards on the earliest (and shortest) exposures; with only a slight increase in exposure time all of the subjects produced an initial guess for all of the cards. The trick, though, was that some of the cards were purposely made anomalous (e.g., a red six of spades or a black four of hearts). For normal cards, initial guesses were generally correct. For anomalous cards, however, the subjects almost universally misidentified them as legitimate playing cards—for instance, calling a *red* six of spades the six of either spades (as though it were black) or hearts, as though they had actually seen only normal cards.

The phenomenon of experience-configured perception is echoed throughout the theoretical and scientific literature. In commenting on the limits of hearing, Nietzsche acknowledges

[16] Also cited by Kuhn [129, pp. 62–64, 112–113].

this principle: “One hears only those questions for which one is able to find answers” [157, p. 206]. Similarly, hermeneutic philosophy stipulates that one must move into the tradition of the writer when interpreting texts, and when conducting ethnography, one must be socialized into the culture (e.g., participant observation) [55, pp. 104–145]. Without socialization, and especially without an awareness of the need to adjust their conceptual frameworks, the subjects of the card experiment unknowingly adapted their observations to an inaccurate ontology. Contrary to their knowledge, red sixes of spades could exist! The relationship between personal ontologies and perception may be why, as Kuhn pointed out [129], two people can look at the same phenomenon and literally see different things.

When an observation conflicts with one’s conceptual framework, as in the card experiment, there is a conflict between one’s *per*ception of reality and one’s *con*ception of reality. We generally do not tolerate such inconsistencies at a personal level [71, 91, 210]; the resulting cognitive dissonance presses us to adjust either the observation or the conceptual framework in order to bring the two into alignment. The trouble, though, is that “our psychic makeup is somehow adjusted to disregard whole realms of phenomena that are so all-pervasive as to be irrelevant to our daily lives and needs” [215, p. 210]. Thus according to Kuhn, the response in science to this type of cognitive dissonance necessarily begins with unconscious adjustments to the offending observations [129, p. 64]. Although the conceptual framework and the observation are probably both inaccurate to some degree (due to their interdependence), the tendency is to question the observation first because it is the new thing, external and not yet validated. To accept a conflicting observation, however, one must deal with the inevitable ripple of changes that occur in the already well-established conceptual framework.[17]

In Kuhn’s terms, only during a period of crisis are conceptual frameworks exposed sufficiently to be challenged directly. Again, in the card example, most of the subjects did

[17] In Piaget’s terms (see Footnote 14), accommodation is more difficult than assimilation; accommodation typically requires originality and often periods of trial and error [81, pp. 35–36, 45–46, 52, 56, 60–61]. Perhaps this is also why opponents of Popper claim that scientists never actually try to falsify their hypotheses [55, pp. 29–54].

eventually discover the anomalous cards, but that awareness came only after increased and *repeated* exposure to the anomaly—*that is, knowledge only came by repeated experience (or replication of experience)*—and even then only after undergoing a process of hesitation, confusion and crisis, and eventually an awakening. In most cases, once a subject correctly identified one or two of the anomalous cards, s/he was able to identify the others with little difficulty. In those cases, *repeated exposure* had induced an observational crisis, which effectively exposed the subjects' conceptual frameworks, making those frameworks available for updating. However, some subjects were never able to update their conceptual frameworks. Even after becoming aware of a problem and with forty times the average exposure necessary to identify a normal card, a few subjects were still unable to correctly identify the anomalous cards. One such subject exclaimed in confusion, "I can't make the suit out, whatever it is. It didn't even look like a card that time. I don't know what color it is now or whether it's a spade or a heart. I'm not even sure now what a spade looks like!" [34, p. 218]. Quite possibly, a prior awareness of conceptual frameworks would have helped this subject better cope with the cognitive dissonance.

As these examples show, certain distinctions are forced upon our future perceptions by previously acquired frameworks, distinctions that can be overcome, but only with *repeated* and concerted efforts, and only by creating still more imperfect distinctions. However, our conceptual frameworks are not exclusively limiting, they are also enabling. Returning to language, Christopher Alexander states:

> Every creative act relies on language. It is not only those creative acts which are part of a traditional society which rely on language: all creative acts rely on pattern languages: the fumbling inexperienced constructions of a novice are made within the language which he has. The works of idiosyncratic genius are also created within some part of language too. And the most ordinary roads and bridges are all built within a language too. [2, p. 208]

According to Alexander, language enables us to be creative by steering us away from meaningless arrangements, thus allowing us to focus our efforts on key aspects and refinements of the conceptual material with which we are working. Indeed, far more meaningless and unhelpful combinations of concepts exist than those which are useful. With respect to spoken language, if we had to search though all possible arrangements of words every time we wanted to speak, we would never be able to say anything. Thus with a prior framework in place, we can pour all our energy into the finer shades of meaning [2, pp. 206–207]. But of course, this directionality is a two-edged sword; it steers us away from useful constructions just as well as from the useless. Here again we see the tradeoff of all taxonomy.

Ultimately, Alexander's notion of the pattern language is a reformulation of the idea of conceptual frameworks, mixed with concepts from human language and applied to architecture. Consequently, the point he is making—that language enables humans to be creative—does not apply exclusively to spoken language. His argument is in fact far more expansive than the articulation of speech or even than the explicit, conscious thoughts from which that speech arises. The scope of his vision becomes clear when—after painstakingly articulating a process by which to create a pattern language—Alexander then instructs his readers that they must liberate themselves of that language; they must ultimately move beyond it because only through its transcendence can they achieve the final goal:

> So paradoxically you learn that you can only make a building live when you are free enough to reject even the very patterns which are helping you… At this final stage, the patterns are no longer important: the patterns have taught you to be receptive to what is real… And in this sense, *the language is the instrument which brings about the state of mind*… It is the gate which leads you to the state of mind… [2, pp. 542–547, emphasis added]

Thus a truly great architect, one who makes buildings and structures *live*, is one who has *optimized*—through *repeated experience*—his (or her) own frameworks to specifically facilitate that goal. The optimization process for Alexander can be guided by developing a pattern (or symbolic) language around the core objective (note the context sensitivity), but the final usable result is ultimately only achievable by a change in oneself on a deeper level than that of conscious thought or expression. Thus although science is the guardian of rational thought, its true objectives must necessarily run deeper than the explicit and articulable.

5.3.5 Summary of Concepts

The idea of conceptual frameworks (as we present it here) is not intended to be an argument about how the brain or mind represents information internally. Ultimately, we want to avoid making claims regarding the mind's deepest representations of information as continuous, discrete, or something in between. However, regarding the tangible expression of the mind through rational, conscious thought, it seems reasonable to conclude that once "thought stuff" is utilized or activated in the consciousness—that is, once rational thought gets hold of our conceptions of reality—by necessity of its purpose, rational thought must discretize that reality (if not already done at a deeper level). Certainly, the moment thought stuff hits the tongue it must be fit to a discrete set of words, partitioned and chunked, and then rewoven together as language. These processes always yield enhancements to some entities and relationships, while diminishing others, drawing attention to some aspects of existence at the expense of others, all the while leading to a loss of fidelity. This is the nature of taxonomies, of ontologies. The purpose of rational thought is thus to *direct* one's attention, in order to create meaning amidst a mass of otherwise indistinguishable stuff, a sea of stuff flowing into stuff, and this process is that which science is purposed to formalize. Consequently, discretization—which is the formation of conceptual frameworks—not only creates an imperfect representation of reality (selective representation), but simultaneously *enables* knowledge formalization, expression, transfer, communication, application, and so forth—it is fundamental to all processes of science. We are, therefore, inescapably bound to conceptual frameworks, which are necessary and prerequisite to all thought and experience. Moreover, conceptual frameworks are always incomplete models that fundamentally impact our perceptions and thinking *for better and* *for worse*. As with the gestalt effect, the act of moving into a framework creates meaning, though simultaneously, it necessarily obscures numerous other possible meanings. Thus the paradox of our knowing the world is that conceptual frameworks are *both necessary and always insufficient* for perceiving, thinking, and acting—and it is this last point which brings us to the heart of Simmel's concept of transcendency.

5.4 Simmel's Transcendency

To explain his theory on the transcendent character of life, Simmel spends considerable effort substantiating the fact that our lives are bounded in every way. First, he explains that we always stand or orient ourselves between two boundaries, a higher and a lower. Our lives in fact are divided into more and less, better and worse, wiser and more foolish (to use Simmel's examples [193, p. 353]). Everything we do is ascertained by us in terms of some measure of meaning, adequacy, morality—some placement on a scale to reflect how good things are, a classification or assignment—which both reflects things as they are, but is nevertheless oriented with respect to the arbitrary position of self. "The boundary, above and below, is our means for finding direction in the infinite space of our worlds," says Simmel [193, p. 353]. But the dichotomous boundaries are not the only ones that hold us together. Every concept delineates a boundary. Consider speed, for instance, and slowness: to paraphrase Simmel, we cannot actually conceive speed and slowness beyond a certain degree—the speed of light, the slowness with which a stalactite grows—"we cannot project ourselves into such tempi" [193, p. 355]. And the same limitations bind us for temperature, perception of light beyond the visible spectrum, the experience of time, and so forth. Ultimately, as Simmel explains, our boundaries encompass even the most abstract concepts. It is as if "[o]ur imagination and *primary* apprehension stake out areas from the infinite fullness of reality and the infinite modes of apprehending it" [193, p. 355]. Thus we live in our own self-defined boundaries, and in some sense we *are* boundaries because we have boundaries "everywhere and always" [193, p. 353]. The ways we think and the ways we act are defined by our boundaries.

What, then, are these boundaries, with respect to conceptual frameworks? Said Simmel, "For insofar as every content of life—every feeling, experience, deed, or thought—possesses a given intensity, a specific hue, a certain quantity, and a *certain position in some order of things*..." [193, p. 353, emphasis added]—which is to say that life, as an experience of some reality, is situated in conceptual frameworks (or Simmel's forms), which forms grant it a measure of *determinacy*, objectivity. Simmel continues, "This participation in realities, tendencies, and ideas... may well be obscure and fragmentary" [193, p. 354]—a reference again to Simmel's fragmented man; knowledge is the man-made fusion of fragmented images, themselves distorted in their apprehension by the presuppositions of that same fusion. Although life is situated in frameworks of our own construction, and although those frameworks are themselves fragmentary in origin, these frameworks give life the complementary (and contradictory) advantages of "richness and determinacy" [193, p. 354]—richness because our frameworks bring us closer to the objects of their focus, granting us perception and understanding, and *determinacy* because our frameworks also distance us from those objects, granting us sufficient objectivity to act. It is in the compromise between intimacy and objectivity, nearness and distance, that Simmel situates life, knowing, and we would add, knowledge.

Therefore conceptual frameworks are the boundaries of which Simmel speaks when he talks of the transcendent character of life. Simmel calls them "forms", but as with conceptual frameworks, forms simply represent the particular carving up and reforming of the world that we hold at any given moment. As boundaries they are always necessary and yet insufficient at the same time, and it is from this paradox, or contradiction of life and knowing, that Simmel draws his insights on transcendency. Of this Simmel states:

> A deep contradiction exists between continuity and form as ultimate world-shaping principles. Form means limits, contrasts against what is neighboring, cohesion of a boundary... form impresses on its bit of matter an individual shape... tears the bit of matter away from the continuity of the next-to-one-another and the after-

> one-another and gives it a meaning of its own, a meaning whose *determinateness is incompatible with the streaming of total being*, if the latter is truly not to be dammed. [193, pp. 365–366, emphasis added]

Thus via their *determinate* natures, conceptual frameworks enable thought, experience, and action, but simultaneously, as constraints they are limiting and never sufficient. Ultimately, conceptual frameworks can never grasp reality in a single unity because from the moment of their construction they always dam or block it up—or in other words, conceptual frameworks (which constitute knowledge) are always outmoded from the moment they are constructed.

Before moving on, we illustrate the two sides of this point with examples, one from the linguist Benjamin Whorf and a second from Friedrich Nietzsche. First, the statement from Whorf:

> Somewhat analogously, the mathematical formula that enables a physicist to adjust some coils of wire, tinfoil plates, diaphragms... into a configuration in which they can project music to a far country puts the physicist's consciousness on to a level strange to the untrained man, and makes feasible an adjustment of matter to a very strategic configuration, one which makes possible an unusual manifestation of force... We do not think of the designing of a radio station or a power plant as a linguistic process, but it is one nonetheless. The necessary mathematics is a linguistic apparatus, and, *without its correct specification of essential patterning*, the assembled gadgets would be out of proportion and adjustment, and would remain inert. [215, pp. 249–250, emphasis added]

Whorf's point makes clear that specific a priori patterns enable us to wield and control power. Whorf speaks in terms of languages because for him language is the physical manifestation of conceptual frameworks, or possibly the physical encoding (in some sense) of conceptual frameworks. But to abstract his concept, allowing conceptual frameworks to be something broader, which interacts fundamentally with language, but also with many other human mental processes, we see that conceptual frameworks are critical to what we can actually accomplish in the world of practice. In fact, it is not hard to see that nothing can be done or achieved at all, no action of even the smallest import can be executed without some conceptual framework catalyzing, guiding, and driving it to completion. Thus it is important that we know how to manage our conceptual frameworks, and from a developmental and analytical standpoint, that we do not get "stuck in a rut." This point relates directly to science, which is an institution purposed to generate *usable* knowledge, knowledge that can enable new action, new powers of movement, knowledge to enable us to move into new states of being and living. Thus science and the management of conceptual frameworks are quite similar. Science in some sense can be viewed as the management of conceptual frameworks, or at least as a superset of that function.

On the other hand, Nietzsche makes clear the extent to which conceptual frameworks also bound, or to use his connotation, cripple us:

> Almost always the books of scholars are somehow oppressive, oppressed; . . . every specialist has his hunched back. Every scholarly book also mirrors a soul that has become crooked; every craft makes crooked. [Specialists are grown] into their nook, crumpled beyond recognition, unfree, deprived of their balance, emaciated and angular all over except for one place where they are downright rotund. . . [157, p. 322]

In specializing, scholars distort and disfigure themselves, and this disfigurement is inescapable. "On this earth one pays dearly for every kind of *mastery*," says Nietzsche [157, pp. 322–323]. Every scientific field, though it may have a golden floor, has a leaden ceiling [157, p. 322]. It is the fundamental nature of all taxonomies, of all conceptual frameworks. All taxonomies accentuate some features and properties, while diminishing infinitely many more. Thus all specialties cripple us; we cannot maintain optimal distance and objectivity, while at the same time approaching the world with complete intimacy. To characterize the situation pessimistically, the crippling nature of conceptual frameworks is both necessary and inescapable; our frameworks are necessary for knowing (as well as experiencing, acting, etc.), and yet never sufficient to fully know the world. In Nietzsche's words, the "'polydexterous' man of letters...who really *is* nothing but 'represents' almost everything," is to be despised, whereas the scholar is to be praised for his hunched back [157, p. 323]. Possibly with this fact in mind, Nietzsche said elsewhere that a true search for knowledge requires, as it were, being "reborn in a hundred beings" [157, p. 215].

This brings us to the crux of Simmel's transcendency:

> Life [or we might substitute *knowledge*] is thus caught up in a contradiction, that it can only be accommodated in forms [or conceptual frameworks] and yet cannot be accommodated in forms, that it passes beyond and destroys everything which it has formed. This, of course, appears as a contradiction only in logical reflection, which *conceives the individual form as an intrinsically valid, real or ideal fixed structure*, discontinuous with other forms, and in logical contrast to movement, streaming, reaching further. Life as immediately experienced is precisely the unity of being formed and that reaching out beyond form which manifests itself at any single moment as destruction of the given current form. *Life is always more life than there is room for in the form allotted by and grown out of it.* [193, p. 370, emphases added]

This dual nature of life and of knowing is the meaning behind the quote at the beginning of this paper. To know something is to simultaneously limit one's knowing, in real and consequential ways. Thus the process of living—and so too of knowledge production—is one of constantly constructing new frameworks only to blast them down again. Life is both fixed and variable, and the essence of it is its capacity to overstep its own bounds [193, p. 364]. "For, although the boundary as such is necessary, every single determinate boundary can be stepped over, every enclosure can be blasted, and every such act, of course, finds or creates a new boundary" [193, p. 354]. Transcendence is, therefore, immanent in life [193, p. 363], and so too in knowledge production. Thus the "achievement of every structure is at once a signal to seek out another one" [193, p. 370].

Having described the particulars of Simmel's transcendency and having further related it to conceptual frameworks, we are finally prepared to address the question of relevance. Is the contradiction of our conceptual frameworks—that knowledge can only be accommodated in conceptual frameworks, and yet cannot be accommodated in conceptual frameworks—really important? Is it truly necessary that we are aware of our conceptual frameworks, or at least of their existence? Simmel in fact does argue that transcendence is immanent in life; in which case, we will surely get beyond our frameworks sooner or later. And indeed, it seems that we are constantly stepping over one boundary or another, so why should we as a scientific community take these ideas seriously?

Simply stated, we should care because science is the disciplined process of learning—or in other words, the *explicit* management of knowledge production. Life for Simmel may inevitably transcend itself by constantly and unconsciously overflowing its bounds at every opportunity, but science by definition is not to be left to the whims of fate. The whole point of science is to be systematic about things. Therefore, if life is defined by its boundaries—which are conceptual frameworks, or our knowing the world—and if science is the explicit management of that knowledge, then whenever we find ourselves struggling to build knowledge (in science), we must consider one question before all others: *By what mechanism are we and/or should we be managing the transcendence of our own conceptual frameworks?* This, of course, is a methodological question.

5.5 Transcending Our Conceptual Frameworks

In this section, we return to the central thesis of this paper—addressing the question of whether replication is truly essential to knowledge production in empirical software engineering. However, it will soon become clear that we are asking the reader to consider a broader definition of replication than typically discussed in the software engineering literature, one which is capable of encapsulating the entire knowledge-building process. We allow the boundaries of replication to extend considerably beyond the basic goal of validation, and even beyond what many would consider to be the limits of differentiated (or conceptual) replication for the purpose of generalization. However, as we show, replication (defined in terms of *repeated experience*) is a fundamental method of transcending our conceptual frameworks. Thus, we argue that far more may be gained by using it as a comprehensive structuring principle for knowledge production than is risked by expanding its meaning.

Simmel argues that life by its nature both creates and transcends boundaries; as he puts it, "transcendence is immanent in life" [193, p. 363]. Boundaries are necessary for life and experience to exist, but inevitably we transcend those boundaries, always creating new ones. Thus "the process of 'reaching beyond itself' [is] the primary phenomenon of life" [193, p. 364]. The only true, non-transcendable boundary is the fact that we cannot escape all boundaries. "It is as if the 'I' were always chasing after itself, without ever being able to overtake itself" [193, p. 364]. This state, however, is not necessarily unfortunate because to completely obliterate all boundaries would be to live in a world without meaning. Consequently, taxonomies are necessary to our cognitive existence, and conceptual frameworks are a fundamental part of our thinking, perceiving, and interacting with the world.

Given this state, what are we left with then?—nothing more or less than the management of our own transcendence. In other words, given that conceptual frameworks are always partial and fragmented, then it seems reasonable to conclude that *not* all *valid* ways of conceiving the world are most useful at a given time, within a specific context. As a formalization of the learning process, science is thus responsible for the controlled transcendence of conceptual frameworks—i.e., the effort of consciously and systematically manipulating and evolving frameworks by which to understand the world. Of course, recognizing the reality of our conceptual frameworks is itself an act of transcendence; to discern a wall is to perceive the other side [193, pp. 355–358]. Nevertheless, recognition is not enough. "For only whoever stands outside his boundary in some sense knows that he stands within it, that is, knows it as a boundary" [193, p. 355]—or in other words, *some things we cannot know about the world until we have discovered (or created) a way to represent them conceptually.* The question thus becomes, how do we step beyond our boundaries in order to see them from without?—or how do we locate ourselves in a world beyond that which we are currently capable of representing?

Recalling Bruner and Postman's card experiment [34], in which subjects were asked to identify playing cards given limited visual exposure, remember that it took many episodes before the subjects were able to correctly identify the anomalous cards. Once they were aware that anomalies could exist, most subjects were able to easily recognize and correctly describe all of the cards, even with the shortest exposure times. Thus we see that *repeated experience* facilitates the transcendence (and, presumably, the improvement) of conceptual frameworks. This effect, however, is not exclusive to individuals:

> It took many centuries of controversy about ideological and practical issues before some people realized that their ordinary ideas might not be accurate and hence were in need of logical ordering and empirical testing. *Even after* there arose a community of individuals dedicated to this purpose, much of the raw material of human illusion remained mixed in with the more solid part of sociological knowledge. [43, p. 2, emphasis added]

We find this description instructive on two accounts. First, as with the card experiment, overcoming illusion in sociology is an extended process, one requiring concerted and focused repetition of experience—in this case, through observation and empirical testing (not unlike software engineering research). Second, scientific disciplines rely on repeated experience to evolve conceptual frameworks on a cultural/social level, just as individuals do on the cognitive level. In the former case, however, we simply refer to the process by other terms, such as the evolution of theories, paradigms, or worldviews. Thus conceptual frameworks, which are defined on the level of individuals, also reflect macrosocial aspects of scientific development. Of course, the reverse is true as well. Thomas Kuhn's stage theory, including the development and succession of paradigms, applies in general form to the individual. In the card experiment, for instance, subjects experienced multiple stages of cognitive progression, beginning with unawareness, followed by confusion and crisis, accommodation, and finally (for most) resolution. These stages do not perfectly mirror Kuhn's description, but the general form of their progression is the same—*repeated experience drives a cycle of conceptual framework (or paradigm) formation, development, and replacement/evolution.*

We have thus far argued by example that repeated experience *can* facilitate knowledge production. However, conceptual frameworks imply a stronger assertion than that. According to the theory of conceptual frameworks, *there is no other way to build knowledge besides repeated experience*. First, recall from Simmel's transcendency that conceptual frameworks can never fully grasp reality in a single unity, and consequently, the "achievement of every structure [or knowledge] is at once a signal to seek out another one" [193, p. 370]. Given these constraints (which are deeply rooted in the nature of human cognition), science cannot conquer both absolute and universal truth once and for all. Instead, knowledge production always entails multiple perspectives (or conceptual frameworks) and an agile flowing between those frameworks. Remember that to fully know something requires at a minimum both nearness and distance (referring back to Simmel), and for humans these two perspectives can never be reconciled completely within a single representation (though improvements can be made with effort and new frameworks can be better than old ones). Consequently, Simmel's fragmented man insists that our perceptions and reconstructions are always partial, coarse-grained containers of experience, rather than copies of reality.

As such, category formation always leads to inflation of some distinctions, while artificially diminishing others. The specific relationships we think are so primary to the world are not necessarily so. Those relationships exist in a sea of relationships; we pick them out first because we are predisposed to do so (by culture, past experiences, instinct, etc.), and (hopefully) second because they are particularly meaningful to our current purposes. Nevertheless, latent relationships may exist that are more meaningful or useful than those which we currently know, and many new relationships will certainly become useful as our goals evolve. Ultimately, the conclusion—to know one thing always means to obscure something else—requires that *repeated experience (or its formal equivalent, replication)* must be a critical element in knowledge production; we cannot produce useful knowledge by any other mechanism.

5.6 Building Knowledge through Replication

The theory of conceptual frameworks highlights two general processes that must be facilitated in any science in order to produce useful knowledge. First, researchers must continually refine their existing frameworks to more effectively address current problems and goals; and second, researchers must regularly transcend their frameworks in order to discover fundamentally new perspectives, which perspectives are needed not just to address current problems and goals, but also to evolve those same goals. Although more difficult, the latter objective is critical to overcoming the persistent, seemingly unsolvable problems.

The theory of conceptual frameworks also indicates that multiple perspectives are required in order to capture a particular phenomenon. Consequently, if the observational process is to be at all systematic, then it must include an adequate variety of replications, sufficient to help (or enable) the researcher to find and transition into the various relevant conceptual frameworks necessary to grasp the phenomenon under study. Or in other words, *replication is a powerful mechanism that drives the refinement and evolution of conceptual frameworks.*

Note that the concept of replication suggested here is much broader than typically considered—i.e., that replication is, essentially, repeated experience. This point is important because it means that all aspects of science which concern the repetition of experience (or observation) should be conceptually grouped and coordinated with respect to the key knowledge-building processes (discovery, validation, and generalization)—i.e., replication should subsume all knowledge-production activities.

In the following subsections, we discuss the implications of expanding replication, propose a framework for constructing replication types based on the theory of conceptual frameworks, and explore ideas relating to the "goodness" of a coordinate axis in our proposed framework (including developing a disambiguated working model of replication).

5.6.1 Implications of Expanding Replication

Expanding the definition of replication leads to several benefits for knowledge production.

First, doing so forces the replication discussion to turn greater attention to *synthesis across studies* (as opposed to the conduct and interpretation of *isolated studies*). According to Karl Hunt (from psychology), "The knowledge of science is not its raw data but its theories" [92, pp. 588–589]—i.e., the human synthesis of its observations. Moreover, the theory of conceptual frameworks predicts that the most useful knowledge occurs at the intersection of a multiplicity of frameworks—that is, via synthesis.

Unfortunately, synthesis is currently a highly neglected area of research in empirical software engineering (surprisingly, this is even true among systematic literature reviews [48]). Many "new" studies are published each year having strong ties to past work, and yet their results are never reconciled with that work. In turn, labeling more studies as replications, and thereby showing more clearly and formally how those studies relate to past work, would encourage researchers to think more in terms of the synthesis problem (as well as provide better guidance to future researchers conducting systematic literature reviews).

Thus, broadening the scope of replication may help accomplish a transition in methodological focus, from thinking in terms of isolated studies to thinking in terms of collections of studies (or as Basili et al. put it [19], in terms of "families of experiments"). Although methods for conducting and interpreting individual replications are needed, in establishing those methods we must consider a broader scope than just that of the individual study. That which is best from the perspective of the individual study is not necessarily best from the perspective of synthesis across studies.

Another benefit of expanding replication is that doing so necessitates extending our taxonomies of replication, which means incorporating more replication types than previously considered. A key problem with replication currently is that it often yields unexpected results; e.g., we intended to run a strict replication, but ended up with a differentiated one [104]. In fact, our replications produce such a wide array of outcomes that most results appear, on the surface, incomprehensible and unusable. Actually, the problem is not that our replications produce varied results, but that the variability exceeds our capacity for recognition and interpretation. It is as though we are trying to put together a puzzle without having first learned to sort the pieces by color and shape. By increasing our sensitivity to the types of replication that occur in practice, we can better recognize those types as they actually occur, as well as develop better methods for interpreting and synthesizing their results.

5.6.2 A Framework for Constructing Replication Types

Under an expanded definition of replication, the primary task for replication methodology is to construct an adequate range of useful replication types. However, the theory of conceptual frameworks predicts an infinite variety of possible types (since each type is actually a finite point within a continuous space). Thus, no static set of types will be fully adequate always and forever—at least not a set that is sufficiently minimal so as to be practically usable. Therefore, the system developed must allow for flexibility in addition to providing structure.

One solution is to develop an axis-based framework in which types can be identified as needed. Such a framework would act as a coordinate system, and in so doing, would naturally facilitate the systematic relatability of types (a critical necessity in order to generate knowledge from a set of variably-typed replications). To develop replication types via an axis-based framework would require two steps: 1) to identify an adequate range of useful coordinate axes, and 2) to identify productive intersections between those axes.

To date, numerous replication taxonomies have been proposed in various fields (for examples, see Gómez et al. [83, 84]). In all cases, the taxonomies consist of types identified and demarcated via conceptual discussion—i.e., they are not grounded in a coordinate system. Thus, although relatable within a given taxonomy, the types are difficult to systematically reconcile across taxonomies. Conversely, developing a set of replication axes, rather than types, and then subsequently deriving the types (as intersections between axes), yields not only a set of types with systematically definable relationships, but an expandable multidimensional coordinate system in which otherwise disparate taxonomies may be reconciled. In other words, the process allows new types to be integrated by, at the very least, identifying new intersections between axes and, in the worst case, by adding to or modifying the axes.

Such a framework reduces the need to draw fixed boundaries around replication definitions and methodologies—i.e., we can allow new replication types to be identified and adopted on an ad hoc basis. Given an axis-based framework, researchers can dynamically carve out structure as such structure proves itself to be applicable in practice. If, as with the grammar of a language, the underlying axes are standardized, then the subsequent pantheon of replication types will remain self-consistent and relatable—despite being the product of a distributed and dynamic decision making process.[18]

The notion of standardizing a set of axes, however, leads to yet another question, that of how to assess axis quality. Fortunately, the criticality of the axis-based selection process is not as high as the type-based selection process—primarily because axes are more flexible than types, given that they define gradients. A reasonable set of axes should allow for considerable flexibility (i.e., representational power) in identifying/defining practically relevant types. That said, the selection of axes should still be considered with care, since, as we show, certain axes lend themselves more readily to being mapped into a knowledge-production framework.

[18] One possible operationalization of this idea is Bahr et al.'s five-axis taxonomy: time, place, subjects, methods, and investigator's purpose. "These five [axes] (considered as dichotomies) generate a 32-cell property space that portrays the possible types of replication in more detail than [other] typologies" [11, p. 251]. Of course, this is just one possible instantiation, and one based on dichotomies. Further work is required to assess the fundamentality and applicability of the axes proposed by Bahr et al.

5.6.3 Initial Exploration of Axis Quality

To begin understanding what it means for an axis to be "good", we compare two specific possible axes: 1) *study similarity* [19, 84, 102]; and 2) *knowledge strategy* [55]. The study-similarity axis is well known in empirical software engineering and refers to the similarity between a replication study and its corresponding original (or reference) study. The knowledge-strategy axis, which we take from the social sciences, is similar in many respects to study similarity, but with a more direct focus on the knowledge production process. Below, we describe each axis in detail and then discuss their overlap.[19]

Example Axis—Study Similarity

Study similarity is the most common conception we find in any research field for how to differentiate replications [83, 84]. In software engineering, study similarity is most commonly represented as a dichotomy[20] of *strict* (or exact, similar, close) replication versus *differentiated* (or non-exact, conceptual, operational) replication. The dichotomy derives from two fundamental goals of replication: 1) "Testing that a given result or observation is reproducible" (i.e., the *reliability* test) and 2) "Understanding the sources of variability that influence a given result" (i.e., the *generality* test) [11, p. 250], [92, p. 589], [190, pp. 212–213]. Strict replication addresses the first of these two goals, whereas differentiated replication addresses the second.

A strict replication attempts to repeat a prior study's stated protocol as precisely (or strictly) as possible. A strict replication tests whether the stated protocol generates the stated results. If the results of the prior study are confirmed, we increase our *confidence*

[19] We discuss these two axes not because they are necessarily better than other possibilities, but because they represent a particular contrast on which we can develop ideas about axis quality. Further work is needed to devise specific criteria for axis quality and then to select an ideal set of axes for representing replication types.

[20] In recent work by Gómez et al. [85], as well as in work from other fields (e.g., sociology [11]), study similarity is divided into a more complex set of interrelated axes, which expand on the notion of what it means for a replication to be "differentiated". For the preliminary analysis in this paper, however, we limit study similarity to a single dichotomous axis.

that all necessary variables and methods have been identified and adequately understood to reproduce the phenomenon on demand. Note that the type of knowledge we build through strict replication is confidence, not certitude. The primary goal of strict replication is to hedge our future research investments against the risk of bad assumptions. We validate past studies so that we can, with some degree of confidence, proceed with the construction of higher-order knowledge.

In contrast, a differentiated replication intentionally alters aspects of a prior study in order to test the limits of the relationship between the stated protocol and the stated results. The researcher is interested in extending the prior study's conclusions by either broadening or narrowing generalizability. For example, reproducing a prior result via a replication in which the sample population was altered provides support for broadening generalizability. Conversely, failing to reproduce the prior result indicates a need to narrow generalizability—i.e., the conclusions need to be qualified by an additional variable.

Example Axis—Knowledge Strategy

"Knowledge strategy" is the name we give to an otherwise unnamed replication axis described by Paul Diesing in his book, *How Does Social Science Work? Reflections on Practice* [55]. In Diesing's words:

> [T]he rule to replicate exactly is a mistake. A replication is a test; but it is also part of a larger search and discovery process. *We cannot test one hypothesis* about flatworms, or students, or therapy patients, or small groups, until we have learned many things about the interacting factors affecting their behavior in an experimental situation... Until we have accumulated some plausible knowledge about many contextual factors, we cannot even specify what an exact replication is... For testing, a replication should be the same; but for search it should be different. Both are necessary. In the early stages of a project, sameness is not even possible, or it occurs by accident; later, both search and testing can be

> more systematic. In the early stages, failure to get the expected results is not a falsification but a step in the discovery of some limiting or interfering factor or a sign of experimenter error. Falsification, in the later stages, consists of finding that some other combination of factors was producing the predicted outcome. [55, pp. 337–338, emphasis in original]

The basic elements of Diesing's axis are *test* and *search*. According to Diesing, achieving a stable result (which is demonstrated via *testing*) is indicative of the fact that one has identified the key variables (including methods) on which the phenomenon pivots. Conversely, as long as the result slides around from replication to replication, then the key variables are not fully known or sufficiently understood. In this latter case, the task facing the researcher is to *search* out new variables and/or to clarify/validate the salience of known variables. Thus for Diesing, test essentially means validation and search essentially means exploration.

Comparison of Axes

As with the study-similarity axis, the elements of test and search also correspond to the two general goals of replication (reliability testing and generality testing). In fact, a test-oriented replication could be viewed as roughly synonymous with a strict replication, whereas a search-oriented replication could be viewed as roughly synonymous with a differentiated replication. However, the two axes are not identical, at the very least because the study-similarity axis characterizes the process of replication (i.e., how a replication is to be executed), whereas the knowledge-strategy axis focuses more on the process of building knowledge (i.e., how a replication and its results are to be used).[21]

The study-similarity axis seems on the surface to be a fairly objective classification scheme—meaning, any given study can be clearly identified as belonging to one category or the other (strict versus differentiated). However, in practice, many (or most) strict replications

[21] The fact that neither axis tells the full story highlights the importance of a multi-axis approach to defining replication. Or in terms of the theory of conceptual frameworks, it demonstrates the necessity of maintaining multiple perspectives.

end up "feeling", after the fact, like they should be classified as differentiated (in part because their results diverge from those of the reference study, but also because the more we look, the more execution-related differences we find).[22] Similarly, search and test appear, on the surface, to be fairly objective categories. Search is the primary mechanism for exploring the domain of a problem, whereas test is an assessment step for determining the completeness with which the search problem has been solved. However, upon deeper reflection, we realize that the above definitions fail to clearly delineate the two categories—that is, we cannot tell precisely where test ends and search begins.

As for the study-similarity axis, the classification difficulty may be due, in part, to the fact that the differentiated category covers more ground than does the strict category. For a study to be strict it must fit a fairly narrow set of parameters (i.e., it must follow precisely the protocol of the reference study). However, a replication may be classified as differentiated due to a myriad of possible variations. Therefore, in a sense, it is easier to implement a differentiated replication than it is to implement a strict one; and thus, it is no surprise that many replications intended as strict actually end up differentiated in the end. We could of course, make a similar argument for the knowledge-strategy axis.

Although mildly satisfying, the above explanation does not actually solve the classification problem. For example, in software engineering, we have for years worked to refine our techniques so as to produce replications we feel are truly "strict". However, despite these efforts, strict replications remain both problematic to classify and largely unproductive [31, 32, 40, 49, 70, 105, 137, 181, 189, 190, 211]. Ultimately, to resolve the classification problem requires isolating the ambiguities inherent in the classification scheme itself. To do this, we must first 1) formalize several replication and knowledge-production concepts, 2) define a working model of replication, and 3) resolve the *infinity problem*. We deal with each

[22] For a related example from physics, see Mulkay [154, pp. 75–77]: Early gravitational-wave research had the trappings of strict replication—scientists striving to validate each other's observational reports—but in actuality, it functioned much more like Diesing's search process; strict replication was hardly possible until the theoretical space was well defined. Of this Mulkay states, "as researchers. . . negotiated agreement about which experiments were to be regarded as competent and equivalent, they were defining the nature of their problematic empirical phenomenon and creating a distinctive region of scientific culture" [154, p. 77].

of these issues in turn, after which we redefine the two example axes and form conclusions about axis quality.

Replication Concepts

- *Protocol* – A set of formally articulated variables and methods which define the conduct of a study.
- *Protocol intent* – The protocol intended by the researcher in planning a replication.
- *Protocol reality* – The protocol that actually occurred when the replication was executed.
- *Protocol description* – An imperfect representation of the protocol reality, as constituted by publications, white papers, lab packages, etc.
- *Execution reality* – The precise state of the universe at the time a replication was executed (i.e., every imaginable variable, even seemingly unrelated variables, such as the phase of the moon).
- *Execution description* – An imperfect (and necessarily incomplete) representation of the execution reality, as constituted by publications, white papers, lab packages, etc.
- *Irrelevant factors* – Elements of the execution reality not included in the protocol description (i.e., factors thought to be unrelated to the results).
- *Protocol proper* – A minimal set of variables sufficient to produce the stated results on demand (i.e., what we want to know).
- *Replication outcome* – The fact of whether a replication did or did not obtain the same result as the reference study.

Knowledge-Production Concepts

- *Salient variables* – Variables (and methods) necessary to produce the stated results on demand (i.e., elements of protocol proper).

- *Inclusion objective* – Determining whether all salient variables have been identified.
- *Exclusion objective* – Determining whether all identified variables are truly salient.
- *Details objective* – Determining whether identified/salient variables are sufficiently understood/documented.

Notes on the above Concepts

We list two concepts above (protocol intent and protocol reality) that we actually do not need for the present discussion. We include those concepts, however, as a way of communicating more clearly what we mean by delineating it from that which we do not mean. In particular,

- We are not interested in protocol intent because intent has only indirect bearing on a replication's results (as opposed to protocol reality). Protocol intent may need to be discussed in a replication report, but conscientious researchers should strive to clearly delineate it from their description of protocol reality.
- Protocol reality is meaningful, and would be preferable over protocol description, if it were knowable. However, since it is not knowable, and so cannot be used to guide the knowledge-building process, we ignore it in preference for protocol description.

One of the primary objectives of replication is to formulate a protocol description that approximates the (albeit unknowable) protocol proper. However, the variables initially included in a protocol description may not be sufficiently comprehensive or may not all be salient. In other words, some elements of the execution reality may, in fact, be critical to producing the stated results, but are not yet known. Such variables must be identified and added to the protocol description (the inclusion objective). Additionally, some variables in the protocol description may, in reality, have little or no effect on the stated results. These variables must also be identified, but in this case, they need to be removed from (rather than added to) the protocol description (the exclusion objective).

Since, due to its vastness, the execution reality is not fully knowable or reportable, we include "execution description" as a distinct concept. Unfortunately, most published reports do not currently distinguish between a protocol description and an execution description. Rather, they mix elements from both sets to produce a single undifferentiated discussion.

A Working Model of Replication

Based on the above concepts, we can describe the replication/knowledge-production process as follows:

- *Original Study* – First, a researcher conducts an original study. Based on that study, s/he reports a protocol description, an execution description, and a set of results/conclusions. Recognizing that the execution description cannot be truly comprehensive, the researcher includes as much detail as possible about any variable that can be reasonably argued to influence the results (and possibly even some variables thought to be irrelevant, if space allows). The task here is clearly ambiguous, subjective, and theory driven. In some sense, there can be no right answer, but the researcher makes his/her best effort. The purpose of the execution description is to supply data for later stages of the knowledge-building process (specifically, to help in synthesizing across studies). Thus, any data, as long as it is reasonably accurate, is better than no data. As with the execution description, selection of information for the protocol description is also ambiguous, subjective, and theory driven (at least at first). In this case, the researcher makes a best guess at what s/he believes to be the minimal and truly essential variables necessary to produce the stated results.
- *Replication, Series 1 (Inclusion Objective)* – When the time comes for a first series of replications, the researcher focuses on the inclusion objective. To tackle that objective, the researcher replicates the protocol description as precisely as possible. Initially, the researcher may also try to closely match the execution description in order to confirm that the reported results are not inaccurate due to some random, unexplainable

circumstance. But ultimately, the process at this stage calls for varying irrelevant factors in order to determine if any essential variables have been mistakenly excluded from the protocol description. Some of this variance will occur naturally during the course of conducting a replication, but some may need to be conscientiously effected. This is where the execution description comes into play. The execution description can be used not only to identify irrelevant factors to intentionally vary, but can also be used to assess the degree to which irrelevant factors have indeed been varied, and thereby shown to be truly irrelevant. In other words, the execution description is the primary artifact by which completion of the inclusion objective can be assessed. For varied factors, a confirmatory outcome signals that the factor is truly irrelevant; otherwise, the factor must be considered for inclusion in the protocol description.

- *Replication, Series 2 (Exclusion Objective)* – At some point, the inclusion objective is shown to be reasonably satisfied, after which the researcher turns his/her focus to the exclusion objective. The exclusion objective is, in a sense, less important than the inclusion objective for two reasons. First, it is less important because, in forming the initial protocol description, the researcher selected only those variables believed to be truly important; thus, the protocol description is already minimized to some degree. Second, for the protocol description to yield usable (i.e., transferable) knowledge, it does not necessarily need to be optimally minimal. Thus, the researcher may decide to end the investigation after the first series of replications. However, if the protocol description is bloated, due to an excessive number of variables (which necessitate an unwieldy number of caveats in the conclusions), then the researcher may need to execute a second series of replications. In this series, the researcher systematically varies elements of the protocol description to determine whether specific variables are truly necessary. As long as the replication outcome is confirmatory, the varied factors can be dropped from the protocol description (or at least further generalized).

Note that the details objective cannot be directly addressed, since the protocol and execution realities are unknowable. However, consistent failure to produce confirmatory results during Series 1 replications is a good indication that the conception and/or articulation of variables is inadequate. Conversely, successfully traversing the inclusion objective provides considerable confidence that the details objective has been sufficiently accomplished.

Finally, note that the above process is idealized. For instance, in practice, researchers may have to iterate between Series 1 and Series 2 replications due to the difficulty of identifying when the inclusion objective is truly complete. In other cases, researchers may have to find ways to pursue both objectives within the same replication, thus meshing the two series together. That said, the above description is useful for addressing the problem of classification with respect to our two example axes.

The Infinity Problem

The primary reason why replication, both in general and as a classification problem, is so difficult in practice is because it involves an infinite search space (i.e., the infinity problem). For example, in the naive sense of the term, a replication can fail to be *strict* in an infinite number of ways. The infinity problem is precisely the problem characterized by the theory of conceptual frameworks; it is the reason why conceptual frameworks can only selectively represent reality, as well as why transcending our frameworks is a difficult and always somewhat ad hoc process. Since replication is essentially a process for developing conceptual frameworks, it too must deal with the infinity problem. To resolve the infinity problem with respect to replication, we briefly consider how the problem is managed in the theory of conceptual frameworks.

According to the theory of conceptual frameworks, knowledge (or conceptual frameworks) represents a fundamental tension between two extremes: complexity and generality. *Ultimate complexity* is to model reality in all its infinite detail; in a sense, it is to have a map of the world, actual size.[23] Conversely, *ultimate generality* is to abstract away all detail, to produce a single, formless mass. Neither extreme is of much practical value; and ultimate complexity, in particular, is not actually possible given the infinite nature of reality (as opposed to the finite tools of language and rational thought). Conceptual frameworks, therefore, represent a balancing point between complexity and generality—which for human cognition is achieved via processes such as assimilation and accommodation (see Section 5.3.3, including Footnote 14). Thus, to resolve the infinity problem with respect to replication requires defining a balancing point between complexity and generality, which concept can then serve as a reference or anchor around which to classify replication.

Considering our dissection of replication above, candidate concepts that could represent a balancing point between complexity and generality include: protocol intent, protocol reality, protocol description, execution reality, execution description, irrelevant factors, and protocol proper. Of these, we exclude protocol intent and protocol reality for the same reasons previously discussed. We also discard execution reality because it is essentially a snapshot of ultimate complexity and, therefore, not a balancing point, but an extreme. Similarly, we discard irrelevant factors because it represents an infinite domain. Execution description is finite, and thus a reasonable candidate. However, we discard it because, by definition, it is intended to be an unfettered approximation of execution reality (i.e., it should always be biased toward too much complexity). This leaves two remaining candidates, protocol description and protocol proper, both of which are finite. Since protocol proper is unknowable, we select protocol description to represent our balancing point for replication. In that case, protocol proper is the theoretically ideal balancing point, whereas protocol description is the heuristically-determined, potentially ever-evolving, approximation of that ideal.

Based on this setup, several important points become clear:

[23] Reference to a joke by comedian Steven Wright: “I have a map of the United States... Actual size. It says, ‘Scale: 1 mile = 1 mile.’ I spent last summer folding it. I also have a full-size map of the world. I hardly ever unroll it. People ask me where I live, and I say, ‘E6’.”

- Too little or too much complexity/generality yields knowledge that is not practically usable (i.e., transferable). With too much complexity, the knowledge will never be applicable outside the setting of its observation. Conversely, with too much generality, the knowledge will apply everywhere and always, but with an intolerably large margin of error.

- Series 1 (of the replication process described above) aims to ensure sufficient complexity in the protocol description (the inclusion objective), whereas Series 2 aims to ensure sufficient generality (the exclusion objective).

- Series 1 (of the replication process described above) is more difficult to accomplish than Series 2 because Series 1 involves heuristically searching an infinite domain (execution reality), whereas Series 2 requires searching a finite domain (protocol description).[24]

- The success of any replication process, as with the success of conceptual frameworks, is dependent first and foremost on effectively managing the infinity problem. All other concerns are secondary. If the infinity problem is not adequately addressed, then the process is guaranteed to fail.

- Truly exact replication (i.e., to precisely replicate execution reality) is not only impossible, but also pointless because it would reveal no information about protocol proper. Or in other words, it would tell us nothing about the balancing point we need between complexity and generality, and therefore, it would produce no new knowledge. Accordingly, "strict" replication should not be made synonymous with "truly exact" replication—that is, it should not be defined by the fidelity with which it mirrors execution reality.

[24] It could be argued that Series 2 also involves an infinite search space, in that each of the finite number of variables in the protocol description can be conceptualized via an infinite variety of embodiments. However, we anticipate that most such variables would not, in practice, need to be re-conceptualized in any substantial way. Thus, in most cases the search space would be effectively finite. At the very least, we can conclude that Series 1 deals with an infinite domain of higher order than does Series 2.

The Study-Similarity Axis (redefined)

Having selected a balancing point for replication, we can now define the study-similarity axis in such a way as to remove ambiguity, while at the same time preserving the axis's value as a tool for knowledge production. We define it as follows:

- *Strict replication* – The study precisely replicates the *protocol description* of a reference study (i.e., the study is a Series 1 replication).
- *Differentiated replication* – The study varies targeted aspects of the *protocol description* of a reference study (i.e., the study is a Series 2 replication).

Concerning the above definitions, note the following important points:

- For the definitions to be usable, researchers must begin formulating protocol descriptions as separate from execution descriptions. That said, an original study which fails to do so is not necessarily unusable; in that case, the replicating researchers can draft a protocol description based on their reading of the original study. The point is simply to initiate (and then maintain) a working copy of the knowledge (i.e., the balancing point) under construction. Precisely who does the initiating is not necessarily critical. In some cases, the protocol description may actually be more accurate if initiated by an external researcher.
- The definitions do not fully solve the problem of replication, in that a protocol description can still involve considerable complexity (especially considering the potential for interactions between variables). Thus, further refinement of the overall framework is needed. However, the definitions and ideas discussed above do represent a workable infrastructure in which to reason about replication—i.e., a good starting point to which further refinements can be made.[25]

[25] For example, the Gómez taxonomy (mentioned previously in Footnote 20) could be applied to further subdivide the category of differentiated replication. Alternatively, the Gómez taxonomy could be used to structure Series 1 replications by highlighting likely-important variables from the set of irrelevant factors for consideration in successive rounds of testing.

- The key to the definitions is the protocol description, not the replication outcome. This is an important distinction. The replication outcome is only meaningful when interpreted within the context of the study (not vice versa), which context the replication types are meant to represent. To determine the types (or the context) based on the outcome renders both the types and the outcome as useless for knowledge production.
- The protocol description may need to include multiple components, along the lines described by Diesing (from Sociology):

  > The end result of such a research program is not a single generalization but a cluster of generalizations. Some state the limits of validity of a major generalization; others state facilitating or interfering factors. Some state conditions for variations in a process; others describe the variations. [55, p. 338]

  For instance, a protocol description could specify a range of values for each explanatory variable, with mappings for the various values (and their interactions) to the results that would be expected if those particular values were implemented in a replication. Such a mapping could be termed a *theory (or predictive) component*. Additionally, each protocol description could specify a precise value for each explanatory variable, thus documenting the actual protocol implemented in the given replication. Such a description could be termed an *instance component*. The instance component would represent the state of an individual replication, whereas the theory component, being a synthesis of results across studies, would act as a persistent knowledge store. Via a replication's instance component, its results could be mapped onto (and thus used to refine) the theory component, which would carry over from replication to replication.

Given the above definitions for *strict* and *differentiated*, we can now unambiguously classify replications in terms of study similarity. First, the definitions depend on a finite set of variables, those of the protocol description, which means classifying study similarity is no longer subject to the infinity problem; the protocol description supplies a complete spec against which to compare a replication for classification purposes. Second, the definitions clearly separate a replication's type from its outcome, thus removing confusion between the two; moreover, the working model of replication (described above) shows, for each replication type, how to interpret a given replication outcome; thereby each replication type is tied into a knowledge-production framework. Finally, by separating the classification of strict replication from the infinite domain of execution reality (via the protocol description), we have made it realistically possible to execute a strict replication in practice.

The Knowledge-Strategy Axis (redefined)

Having clarified the study-similarity axis, we now need to better define the knowledge-strategy axis, as well as discuss the relationship between the two axes. Concerning the knowledge-strategy axis, recall from Diesing's description that *test* is essentially validation, whereas *search* is exploration.

Since it is impossible (and also pointless) to fully replicate an execution reality, we cannot define test as simply, "to repeat a study exactly." In choosing a better definition, it is worthwhile to consider the term's usage in practice. One common usage of the term *test* is roughly as follows:

> A replication that tries to mimic a prior study as closely as possible in order to assess whether the prior study's results could have occurred due to random chance (e.g., due to sampling error).

Note from this description that a test replication differs from the reference study in two respects: first, in an infinity of unavoidable, and hopefully innocuous ways; and second, in a specific and targeted way, without which the results would be meaningless (e.g., to test for sampling error, a replication must intentionally draw a new random sample). Thus, when we say test, we do not actually mean that nothing has changed. Rather, many things have changed; we simply believe we have kept the changes within a specified scope.

Interestingly, the above conclusion (that a test is a replication involving only changes within a specific scope) sounds very much like our definition for strict replication. Thus, we could define test as synonymous with strict. After all, strict replication does include an element of test in that it "tests" whether the protocol description is sufficient (i.e., whether any variables are missing or are inadequately articulated). However, based on the description of Series 1 replications above, it also seems clear that strict replication includes an element of search—in that we are "searching" for important variables, without which the protocol description is inadequate. Thus, when we execute a strict replication we are (or should be) operating in both test and search-oriented mindsets.

We can also view differentiated replication in terms of both test and search. We can view it in terms of search inasmuch as the primary activity is to "search out" or identify variables for elimination (by varying them). Alternatively, we can view it as test in that we are "testing" specific factors to see if they should remain in the protocol description. Ultimately, a Series 2 (or differentiated) replication differs from a Series 1 (or strict) replication only by which variables are included in the protocol description—i.e., precisely the same assessment is being made in both cases, but from different reference points.

Thus, every replication, strict or differentiated, is both a test and a search activity. The testing activity is defined by the variables from the protocol description that remain unchanged, whereas the searching activity is defined by the variables (either in or out of the protocol description) that are modified. Thus, as far as the knowledge-strategy axis is concerned, two replications differ from one another only by the particular mix of test and search that they utilize. As a corollary, to maximize knowledge production, researchers should take advantage of both test and search, to some degree, in every replication.

Benefits of Overlapping the Axes

The benefit of overlapping the study-similarity and knowledge-strategy axes is that, collectively, they form a knowledge-production pipeline, mapping the physical execution of a replication (strict versus differentiated) into specific knowledge-acquisition activities (test and search). Based on such a framework, practical methods (parameterized by study similarity and replication outcome) can be attached to the activities of test and search, such that cross-study knowledge can be persisted and evolved in a coordinated manner (via the protocol description).

Additionally, adopting the knowledge-strategy axis in addition to the study-similarity axis reminds us as researchers of the fundamental importance of viewing all replications as both test and search activities. In particular, we often view replication as simply a mechanical, repetitive procedure (i.e., a test), and thus we overlook the importance and opportunity of search. For example, from the perspective of replication as a test, post-hoc methods seem like cheating; but from the perspective of search, the value of such methods is clearly evident.

Insights on Axis Quality

Concerning axis quality, the above discussion highlights several key points.

First, in judging axis quality we must take into account how an axis is to be applied in practice. Likely all axes we could devise could be applied in a myriad of ways, not all of which are going to be equally valuable or sufficiently disambiguated. Further, since the quality of an axis is tied to the mode of its application, then for each candidate axis, we must consider the conceptual range of modes in which it could be applied before we can assess its overall utility. Also, upon selecting an axis for inclusion in a standardized taxonomy, the mode of its application must also be standardized (or confusion will likely result).

Second, in assessing axis quality, we must also take into account other axes which may be included in the taxonomy. Clearly, the utility of adopting a particular axis is determined in part by its conceptual orthogonality to the axes already selected. But on a more subtle level, the value of a particular axis may also be dependent on an interaction between it and some other axis. For instance, the knowledge-strategy axis can only tell us something meaningful when considered within the context of study similarity.

Third, it appears that some axes are more tightly coupled with the knowledge-production process than with the physical process of replication. For example, unlike the study-similarity axis, the knowledge-strategy axis says little about how a replication was or should be implemented. Conversely, the knowledge-strategy axis provides clear guidance concerning how to build knowledge given a particular replication outcome. For example, if a strict replication fails to reproduce a result, then the replication becomes a search problem (i.e., identify the variable(s) lacking in the protocol description). Concerning axis quality, neither type of axis is more important (i.e., those oriented toward the knowledge-production process or those oriented toward the replication process); rather, both types are needed.

Fourth, since axes vary in terms of their conceptual distance from the replication and knowledge-production processes (as just described), each axis creates a certain field of awareness for the researcher. Consequently, the particular mix of axes can guide a research investigation more or less effectively by influencing how the researcher thinks about problems. For example, the knowledge-strategy axis is valuable in part because it reminds the researcher of the criticality of the search activity (which reminder is particularly important in software engineering given that the current, dominant view of replication is that of a perfunctory validation mechanism).

5.7 Additional Ideas for Consideration

In this section, we outline several ideas related to the above discussion, but which need further development. We include these ideas as a point of reference for future work.

- *Originality versus innovation and creativity.* Bahr et al. lament, “Social scientists in all fields take great pains, which we must regard as perverse, to demonstrate their originality by not following too closely in the footsteps of their predecessors… For example, at some American universities…the eager student is urged to devise improvements that will make it impossible to compare his results with those of earlier studies in any meticulous fashion” [11, p. 261]. This “itch for originality,” as the authors call it, is a pervasive misconception, as though to be creative means by definition that one's work must be disconnected from others. Conversely, the theory of conceptual frameworks proposes that truly innovative research should typically occur in the intersection of ideas—and thus the harder a researcher works to reconcile his/her work with that of others, the more likely s/he will be to uncover a profound insight.

- *Qualitative paradigms.* The theory of conceptual frameworks indicates that knowledge is not constructed in a linear fashion, one study at a time, with each study looking back only one degree. Instead knowledge production requires, ideally, reassessment of the entire collection of studies each time a new study is added. Viewed in this way, the knowledge production process appears *hermeneutic.*[26] The process is definitely similar to the way in which grounded theory synthesizes multiple data sources via the practice of constant comparison. In fact, many qualitative techniques are based on a cycle of evidence reconsideration, as new evidence is incrementally integrated. The hermeneutic circle (as well as other qualitative paradigms) is, in a sense, reflective of the replication process, and may even be viewed as a pattern for structuring that process.
- *The primacy of search (as opposed to test).* As long as people are the driving function of software creation, it will remain a complex and multifaceted process. Consequently, a diverse set of conceptual frameworks is necessary to obtain a comprehensive understanding of the process. For instance, a recent paper comparing inspection and unit testing via a set of three replications found the effect of interest to be "overshadowed by complex differences in the tasks and experiments" [181, p. 6]. Similarly, in design pattern studies (e.g., [103, 126, 127, 155, 167]), secondary factors have been found to play a considerable role in the outcomes of the main effects. Thus, for software testing, as well as for design patterns and most other areas of software engineering research, the search problem is far from solved. In fact, according to the theory of conceptual

[26] For information on hermeneutics, see Diesing [55, pp. 104–145], especially his description of the hermeneutic circle [55, p. 109].

frameworks, knowledge construction requires constant immersion in the search process, even when the intent of a study is only to test a prior result.

- *The positive side of complexity.* Bahr et al. observe, “The facts of social change in a community are much less manageable intellectually than the myths we make about it when our imaginations are not checked by hard data; but the tangled patterns traced by empirical data are more interesting, and eventually more useful” [11, pp. 249–250].
- *The difficulty of publishing replications.* An oft-repeated concern in software engineering regarding replication is that such studies are hard to publish. The theory of conceptual frameworks, however, predicts that just about any replication—even a strict replication—has the potential to produce a wealth of insight. For example, the Middletown III study (a strict replication) has generated more than 70 publications to date—including numerous academic papers and book chapters, as well as two books [11, p. 246], [171] (and the dataset is still actively downloaded [11, p. 250], [39, 94]). Clearly, not every study is worthy of publication, nor does every study represent the same magnitude of contribution. That said, the theory of conceptual frameworks indicates that with sufficient effort, especially with a mindset toward synthesis, most reasonably-conducted replications should be capable of producing more than enough insight to merit publication. In that case, if replication studies are truly more difficult to publish than original studies, then likely we, as authors, are underutilizing our opportunities to build knowledge.
- *The balance of formal/informal replication.* Bahr et al. note “that many replications are not recognized as replications because they represent continuities in research, and particularly continuities in theory, rather than narrow duplication of prior work” [11, p. 250]. This “informal” type of replication may in fact be the more dynamic and more powerful of the two variants. Being free of systematic constraint, the informal variant may naturally explore a wider range of replication options, conforming more readily to practical circumstances. Then again, an ideal balance may also (and likely does) exist between the two variants, which would provide a degree of scientific control over the

replication process, and yet not excessively constrain that process. The possibility also exists that, as a research field, we are already operating within the optimal balance.

- *The need for synthesis-oriented reporting guidelines.* In 2010 Carver published a preliminary set of guidelines for reporting replications. The guidelines included four major categories of information: 1) background on the original (or reference) study, 2) background on the replication, 3) comparison of the results, and 4) cross-study conclusions. The last two categories clearly suggest some type of synthesis. However, in the detailed discussion of the report, synthesis is limited to comparing a replication's results with those of a *single* reference study. The guidelines do briefly mention families of studies—stating, "some special guidelines are in order for a replication that is part of a family" [40, p. 3]. However, the guidelines provide no more than a brief sketch of elements that ought to be reported in such cases. Furthermore, when the guidelines were the subject of an ISERN[27] session in 2011, the notion of "families of studies" was dropped entirely.

5.8 Conclusions

In *Cultural Boundaries of Science* [80], Thomas Gieryn quotes [80, pp. vii–viii] a scene from Mark Twain's 1894 novel, *Tom Sawyer Abroad* [208], in which Tom is floating in a balloon with Huck Finn across the Midwestern countryside. At one point, Huck questions their speed, concluding that they must not be traveling as fast as they thought or they would have crossed Illinois by now. When Tom asks Huck how he knows they have not yet crossed Illinois, Huck responds by showing on a map that Illinois is green and Indiana is pink—"You show me any pink down there if you can," challenges Huck, "No, sir, its green" [208, pp. 17–18]. The point of the story is that models (such as maps) are abstractions of reality, which only represent targeted features. Similarly, Gieryn argues, "no…scientific theory mimes reality literally, without mediation or translation or interpretation or contextualization" [80, p. viii].

[27] International Software Engineering Research Network

Gieryn is correct that science is “boundary-work” [80, p. 4–5]—both its construction and its consumption. However, boundary-work is not limited solely to formal theories and academic disciplines. As Simmel shows, all human knowledge is boundary-work.

Gieryn is also correct that science is not and cannot be about absolutely mirroring reality. First, we can only selectively represent reality, and second, although some representations are obviously better than others, the goodness of a representation is not solely a function of its correspondence with reality. It should correspond along some set of parameters, but inasmuch as it represents a specific perspective on reality, it should also align with our needs as knowledge consumers. Thus the quality of a conceptual framework is determined not only by its correspondence with reality, but also by the degree to which it facilitates our present goals. Since goals change over time, we must continually evolve our frameworks.

Based on these circumstances, the task for science is three-fold: 1) to help us develop our individual conceptual frameworks; 2) to catalogue and then filter away unhelpful frameworks from the ever growing set of candidates; and 3) to refine/project useful frameworks into the collective consciousness. Consequently, the primary objective of science is not to create a single ultimate framework, but to help us navigate a sea of frameworks. Along the way we must constantly remind ourselves that all activities in science—from creating metrics and designing experiments to interpreting data and forming conclusions—require first moving into a specific framework. As a result, we must be flexible (or creative) enough to switch between different frameworks, iteratively trying on various classification schemes as we search for useful knowledge. As a society we rely on many conceptual frameworks to achieve our goals; science is the systematic discovery, creation, and management of those frameworks.

Obtaining an awareness of our personal conceptual frameworks is important because our internal frameworks critically affect the type, quality, and applicability of the external/explicit knowledge we construct. According to Diesing, “Awareness that we have a conceptual scheme should presumably help us deal with troubles more effectively; we can suspect that the source of the trouble might be our own preconceptions, rather than unknown interfering forces, chance, or poor data collecting techniques. We might also become more tolerant or at least more understanding of the weird theorizing and tricky data manipulation of other schools" [55, p. 62]. Further, as with sociology, the history of software engineering research is "a progressive sophistication about our own thought, uncovering sources of bias that we did not know existed" [43, p. 4]. Once we bring our biases into the open, we can then consciously assess them with respect to our knowledge-production goals. In other words, we are never without theory;[28] to recognize our conceptual frameworks is only to make theory more explicit in science.

Further, the theory of conceptual frameworks teaches us that repetition of experience is the fundamental mechanism by which we validate and extend our knowledge. To transcend our current boundaries, we require continual repetition of experience, and to mitigate the limitations of human knowledge, we must integrate numerous perspectives. Since *replication* is the formalized equivalent of *repeated experience* in science, most studies are actually a replication of some type, and all replications can and should be deep knowledge-acquisition experiences. Furthermore, it is through the juxtaposition of studies that we can discover the deepest, most meaningful insights and create the most useful knowledge.

Thus, in this paper we are proposing a very different view of replication from that of traditional notions. Replication is not merely a mechanical, repetitive, and unimaginative procedure—one that is, no doubt, important, but nevertheless devoid of novelty and insight. Quite the opposite, Simmel's observations about the self-transcending character of life highlight that replication is, in fact, the bedrock of learning; and as such, it is intensely creative. Thus, if approached with the right mindset, replication does not have to be a sterile procedure; rather it can be a creative process that leads the scientist to his or her most novel insights.

[28] Machine learning [150, pp. 39–45] demonstrates that all conclusions require some type of inductive bias (Occam's razor, for example), otherwise an infinite number of hypotheses always exist to explain any empirical data or observation. Similarly, the philosopher Nelson Goodman [55, pp. 16–17] [88] shows that any observation is relevant to any hypothesis, and consequently, the confirmation power of any observation is zero—that is, true validation is impossible. Precisely for this reason, Kitchenham argues, "without some strong theories about the mechanisms underlying empirically observed phenomena we cannot use experiments to assist our understanding of software engineering phenomena" [108, p. 220].

Further, if science is the systematic management of our own conceptual frameworks, and if replication is key to that management, then without replication we really have no science. Replication is, therefore, inescapable if we are to build knowledge.

That said, in this paper we have by no means solved the problem of replication. If anything, we have only uncovered a vast set of new questions. Our findings do suggest a framework for developing a unified taxonomy of replication types, but establishing the axes in that framework is by no means a straightforward task. Further work is needed to test out and further expand the ideas in this paper, as well as to apply the core theory more systematically to the empirical software engineering literature.[29]

5.9 Acknowledgements

We are grateful to the following individuals for reviewing manuscripts of this paper: Howard Bahr (Dept. of Sociology, Brigham Young University), David Grandy (Dept. of Philosophy, Brigham Young University), and Dag Sjøberg (Dept. of Informatics, University of Oslo).

[29] The search space is vast and sometimes feels daunting. We resonate with the anonymous author who wrote the following inspired summary:

> We have not succeeded in answering all our problems. The answers we have found only serve to raise a whole set of new questions. In some ways we feel we are as confused as ever, but we believe we are confused on a higher level, and about more important things.

Chapter 6

Future Work

In this section, we outline several directions for future work. (For a summary of the overall contributions of the dissertation, i.e., the "conclusion-like" material, see Section 1.8.)[1] We divide this section into subsections reflective of the major topics of the dissertation. Note that the items below are highlights only. For specific details, see the individual papers.[2]

6.1 Conway's Law

With respect to Conway's Law, the most pressing research need is to simply run more empirical studies, particularly controlled experiments. Given that the phenomenon is not yet well documented, most studies at this point should probably be viewed as some type of replication—because only once the breadth of Conway's Law is well understood can we begin to conceptually separate studies without stymieing knowledge production. Thus, the primary objective at this point is to catalog variations in how the phenomenon operates in practice, after which niches can be carved out for isolated study. Additionally, in cataloging the phenomenon, it may be worthwhile to survey the literature for empirical evidence of Conway's Law. In making this recommendation, we recognize that our own survey of the literature does not specialize in empirical studies, nor does it consider any materials older than 2003.

[1] Given that the dissertation is structured as a series of papers, placing the contribution summary (i.e., the "conclusion-like" material) in the introduction (rather than here at the end) allows it to serve as a guide to orient the reader and to tie the chapters together.

[2] Additionally, note that this section—which describes general ideas for future work relative to the overall dissertation—is not the same as Appendix W, which provides detailed information on future work specific to Chapter 3.

6.2 Highly Differentiated Replication

As with Conway's Law, highly differentiated replication is in need of more empirical studies. However, in this case, our primary interest is not the topic of the study per se, but the method. In particular, we need to better formalize what we mean by *highly differentiated*. So far we have described it in terms of replicating a prior study's theory, but we have not been clear about what we mean by *theory*. Some amount of formalism could be accomplished without running additional empirical studies, but ultimately (as the theory of conceptual frameworks demonstrates), defining replication types can only be done effectively by first clearly understanding how those types connect into the overall knowledge-building process. Thus, we recommend gaining practical experience with highly differentiated replication before attempting to settle on a formalism. As part of that experience, we recommend exploring different ways to make use of the results of highly differentiated replications, particularly with respect to theory development.

6.3 Design Patterns

Concerning design patterns, the most obvious item for future work is to further replicate the PatMain experiment with additional controls and measurements for moderators. In that case, we recommend focusing on *developer experience*, *pattern knowledge*, and *motivation*. However, we also recommend taking careful notes on other factors which may be influencing the outcome of the experiment. These additional factors can then be cross-referenced against the list we include in this dissertation from our own replication of PatMain (see Section 3.6.2 and Appendices B, C, and Q). Further, it may be worthwhile to explore relationships between moderators. For instance, it would be useful to know whether developer experience can compensate for a lack of pattern knowledge (or vice versa).

Additionally, due to the fact that measurement operationalizations tend to vary across software engineering experiments, controlling variables within studies will not, by itself, solve the problem of generalizability. To fully solve the problem, we also need to develop methods for mapping moderators across studies. To accomplish this, we recommend investigating best methods for assessing common context variables (such as developer experience), and then formulating standardized assessments for those variables.

In addition to further replicating the PatMain experiment, it would be interesting to review the design pattern literature for data on potential moderators. Many studies likely contain at least some traces of information on moderators, the synthesis of which may reveal significant insights. The results of such a literature review could be used to corroborate (or even generalize and extend) the findings reported in Chapter 3.

6.4 The TCA Method

The TCA method could be profitably adapted to other settings. For example, it would be interesting to apply post-hoc moderator analysis with Bayesian models to existing replication data (i.e., without joint replication). Ideally, with further development, components of the method could be made to work on historical data, thus allowing us to reclaim replications which have previously been written off as “failures.” Additionally, TCA focuses on a single type of Bayesian model (an additive-effects model). The model’s relative simplicity, as well as its avoidance of linearity assumptions, makes it robust and broadly applicable. However, other models exist, including a variety of non-linear models, many of which may be preferable in certain specific circumstances.

Another area for improvement concerns the process by which the output of the TCA method is applied across a set of replications in order to generalize the results. The process we document in Chapter 4 essentially requires matching interaction relationships with observed divergences. Further work is needed to develop statistically-based methods (if possible) for applying moderator results to the generalization problem. At the very least, it would be nice to be able to compute a confidence statistic relative to a given generalization hypothesis.

6.5 Replication Theory

Recall that the theoretical component consists of two parts: 1) a distillation of external theory and 2) a preliminary application of that theory to empirical software engineering. Concerning the first part, Chapter 5 is extremely comprehensive. Nevertheless, it pales in comparison to the magnitude of the ideas not yet broached. Moreover, additional qualitative studies could investigate the very same sources and yield completely different, yet equally valid, results. Ultimately, the theory of conceptual frameworks is just a theoretical lens through which to interpret replication problems—it is not the final word on replication theory.[3] Accordingly, we encourage other researchers to bring their own unique perspectives to the table by exploring replication theory in similar ways as we have done.

Concerning the application of theory to empirical software engineering, several items of additional work are needed. First, further work is needed to apply the theory of conceptual frameworks more systematically to the empirical software engineering literature. Thus far, we have mapped the theory onto several replication problems which are pertinent to software engineering. However, we have not systematically reconciled the theory against methodology discussions already published in the literature. To help accomplish this, we recommend, as a good place to start, da Silva et al.'s literature reviews [49, 53], which cover both replication studies and methodology papers. Second, the application section of Chapter 5 (Section 5.6) proposes several key ideas to address replication problems, which ideas need to be tested via practical replications.

[3] Interestingly, the theory predicts this of itself. Or in other words, the theory cannot be both true and absolutely comprehensive since it dictates that all theories must necessarily be selective and positioned.

Appendix A

Replication Web Portal[1]

The replication web portal embodies the materials, instructions, and data collection infrastructure for the PatMain experiment. The web portal allows the researcher to do the following actions:

1. Register and create an instance of the experiment.
2. Create batches of participant IDs.
3. Print little chits providing the experiment URL and login ID for each participant.
4. Monitor experiment progress.
5. Close the experiment and download the results.

The web portal supports administering the experiment in full with all four programs, or administering an abridged version with only two of the four programs (either the pair CO/GR or the pair ST/BO).

Participation via the portal involves answering questionnaires, downloading programs, and uploading solutions. The portal guides the participant through the following steps:

1. Log in to the application using an assigned ID.
2. Choose a language: C++, C#, or Java.
3. Complete two short questionnaires assessing development experience (10 questions) and design pattern knowledge (19 questions).

[1] Cited in Chapter 3.

4. Perform maintenance tasks on two programs (one PAT and one ALT variant), including two tasks for each program (one coding and one comprehension). Coding tasks require the download and upload of source code. Comprehension tasks consist of 1–2 short answer questions.

5. At the end of each program, complete a performance self-evaluation (6 questions per program).

6. Submit final comments—in particular, listing any interruptions.

The total time required for the abridged experiment is approximately 2–3 hours. The portal records the participants' answers, uploads, and timings at each step. By assigning IDs in contiguous sequence, participants are evenly bucketed into the experiment groups. All participants see the same instruction text for a given program regardless of the assigned treatment group.

The portal's source code (written by Martin Liesenberg) is provided in the lab package as a ZIP file (jointrep_experiment_webapplication.zip, packaged by Lutz Prechelt). The ZIP file contains a README.txt and the directory layout of a Java EE WAR file (for Tomcat and MySQL). Portal screenshots and the original experiment programs are also provided in the lab package.

Appendix B

Information on Sites[1]

Table B.1 provides an overview of the experiment execution and participants at each of the four sites. Note that the three-day experiment timeframe mentioned for UA includes only the official PatMain experiment protocols. All additional UA protocols (described below), were administered in a separate one-week period.

B.1 Additional Protocols

The UA replication added protocols beyond those defined by PatMain. All additions were administered to participants as a separate "pre-experiment," which the participants completed prior to the PatMain experiment. Although the two experiments were administered at different times, they involved the same participants and programs. Consequently, UA's results could be systematically different from BYU, FUB, and UPM.

The pre-experiment required participants to view UML diagrams of the programs (not program source code) and to answer questions. Thus UA participants likely performed better on the PatMain tasks—i.e. lower times and higher correctness scores—than they otherwise would have. However, the main effect (program *variant*) most likely remains intact. In general, previewing UML diagrams would tend to raise the overall level of program comprehension. Thus if any impact did occur, the most likely effect would simply be to reduce (but not reverse) the experiment effect among UA participants. Accordingly, we believe the UA data

[1] Cited in Chapter 3.

Table B.1: General information about the experiment and participants at each of the four sites.

	BYU	FUB	UA	UPM	All
Experiment Timeframe	3 weeks	approx. 1 week	3 days	4 days	-
Experiment Execution	asynchronous*	asynchronous*	asynchronous*	asynchronous*	-
Programming Language	Java	Java	Java	Java	-
Language Required	yes	no[†]	yes	no[†]	-
Participant Type	students	students	students	students	-
Solicitation Venue	software engineering course	software project course	software engineering course	research group[‡]	-
Participation	voluntary	voluntary	assignment	voluntary	-
Total Participants	22	13	18	8	61
Undergraduate Students	20	9	0	0	29
Graduate Students	2	4	18	8	32
Data Discarded as Invalid	1 undergrad	1 undergrad	4 grad	2 grad	8
Socioeconomic Background	mostly middle-class	broad	unknown	unknown	-
Nationality	mostly North American	mostly German	10 American 8 other	all South American	-

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid

* Participants took the experiment at a time of their own choosing.

[†] The participants were allowed to select their preferred language (C++, C#, or Java), but all chose Java.

[‡] Software engineering research group of the UPM joint replicators.

can be used in the joint analysis. See the E_joint lab package for copies of UA's pre-experiment artifacts.

Additionally, note that UA tested its participants on all four programs (CO, GR, ST, and BO), rather than on just CO and GR. However, UA issued two web portal IDs to each participant, one for the CO/GR programs and one for the ST/BO programs (rather than running all four programs in the same session). Thus, UA's experiment groups for the CO/GR programs are consistent with those of BYU, FUB, and UPM.

B.2 Design Pattern Education

At BYU, design patterns are taught by two professors (neither of which is affiliated with this research study). The first professor teaches classic GOF patterns via both the Gamma et al. book [77] and via a modestly-sized programming project (approx. 3–5 KLOC). He does not explicitly teach patterns in a generalizable way, but when asked, commented, “it’s probably some of both,” meaning general and specific (interview, Oct. 30, 2012). The second BYU professor also teaches GOF patterns via a modestly-sized programming project (approx. 3–5 KLOC), but does not teach patterns from the Gamma et al. book [77]. Instead, he uses in-class programming examples. Via these examples he first shows “bad ways of solving a problem,” after which he then interactively helps the students to improve those bad solutions by incorporating design patterns (interview, Oct. 30, 2012).

In general, the teaching style at BYU stresses applying design patterns within specific implementations. Of this, the latter professor commented, “On average, I do not think most of the students could generalize the patterns well, given their limited experience in the course” (interview, Oct. 30, 2012). Nearly all of the BYU participants should have previously received training from one of these two professors. Thus both types of education are represented among BYU participants.

At FUB, all participants were recruited from a project course involving Eclipse plugin programming. In that context, they each definitely had practical contact with the Observer pattern, as well as possibly Composite, Strategy, and others. Further, most of the FUB participants had previously taken Lutz Prechelt’s basic software engineering course. That course dedicates two 90-minute lectures to design patterns—specifically, composite, adapter, bridge, facade, observer, strategy, abstract factory, and builder. Later in the course, two further 90-minute lectures on software reuse cover other types of patterns—including, analysis and usability patterns (for requirements), usability architecture patterns (for design), software process patterns, and anti-patterns. Although the design patterns course does not require students to implement patterns, all of the FUB participants had practical experience with patterns in the aforementioned project course.

In general, Lutz explains, “I teach patterns as a form of packaged, reusable, and reconfigurable design ideas where the specific form of their implementation is definitely not a key element of the pattern as such. Ideally, my students should come out with a rather flexible, adaptable idea of what the use of a specific pattern looks like” (email, Oct. 9, 2012).

UA participants were all recruited from Jeffrey Carver’s software engineering course. Jeff reports, “In our course, the students learn design patterns from a book. We do not do any special exercises on patterns. The students in the course each prepare a lecture on one or more patterns to teach to the class. I add information that they miss” (email, Oct. 30, 2012).

UPM participants were all graduate students, recruited from the software engineering research group. According to Natalia Juristo, “We did not teach the students about patterns. Our experimental participants were master and PhD students who said they knew design patterns, which does not mean they really do. None of the students completed their undergraduate degrees at UPM. They studied computing in other Spanish or Latin-American universities. So I do not know how many previously received training on patterns. I tend to think they have learned patterns as part of their work, but cannot be sure” (email, Oct. 30, 2012).

Appendix C

Participant Demographics per Site[1]

Table B.1 provides general information about the participants at each site. Tables C.1 and C.2 summarize the participants' self-reported developer experience and pattern knowledge. For each variable in Tables C.1 and C.2, we statistically compare sites using the Kruskal-Wallis rank sum test (which procedure compares medians, rather than means). We use a nonparametric (distribution-free) test because, in many cases, the data are not normally distributed. Histograms of the data are provided in the lab package.

C.1 Developer Experience (Table C.1)

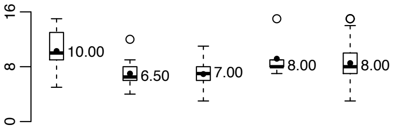
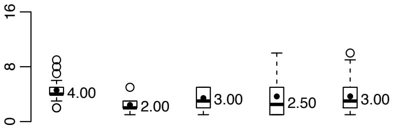
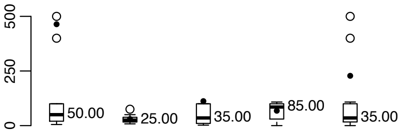
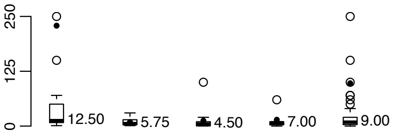
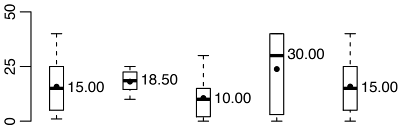
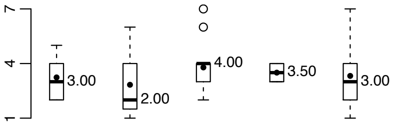
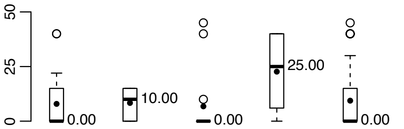
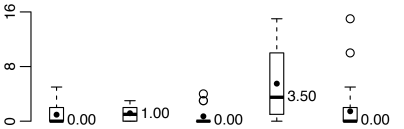
The most noticeable differences across sites (in terms of means) involve the lines of code (LOC) and professional experience questions. Statistically, the sites also differ in terms of the languages-used questions. We discuss each in turn.

Concerning LOC questions (i.e., LOC-lifetime and LOC-Java), four of the American participants (3 at BYU, 1 at UA) report unrealistically high values.[2] For example, one at BYU reports 8 million lifetime lines of code. Conversely, none of the participants at the European universities (FUB and UPM) list such extreme values. Once the unrealistic outliers are dropped, the four sites no longer statistically differ on the LOC questions (updated p-value for LOC-Java = 0.053).

[1] Cited in Chapter 3.

[2] For students reporting at most 5 years professional experience, we consider responses of 150 KLOC or more to be unreasonable. Some of the extreme outliers may be due to typing errors, but some respondents may simply have exaggerated, possibly due to disinterest with the questions. All other data for these participants appear reasonable, so we do not exclude their data from analysis.

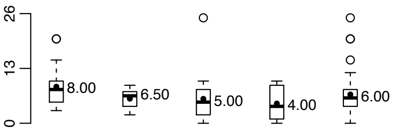
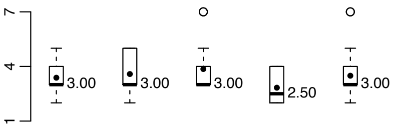
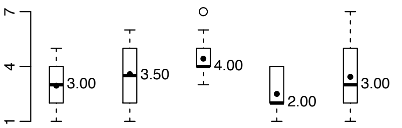
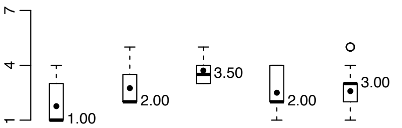
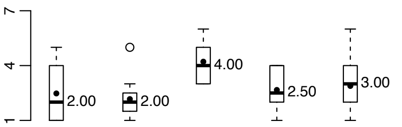
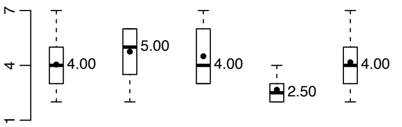
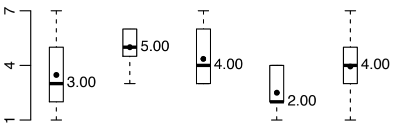
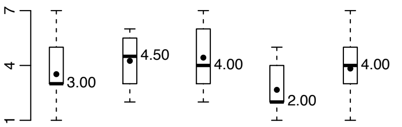
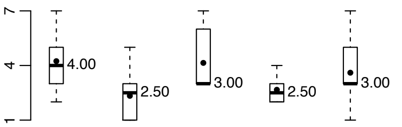
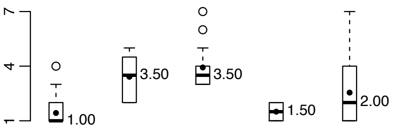
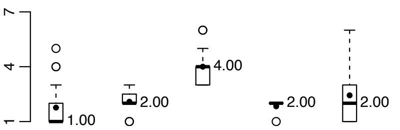
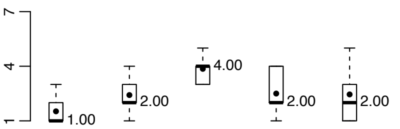
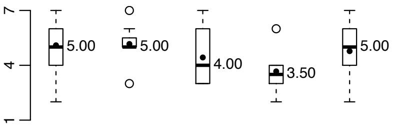
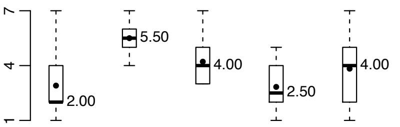
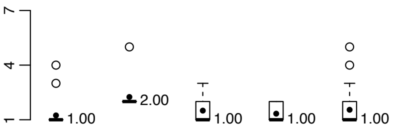
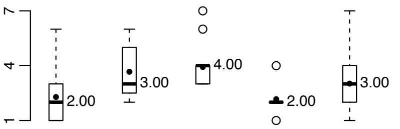
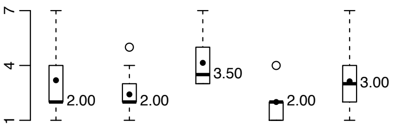
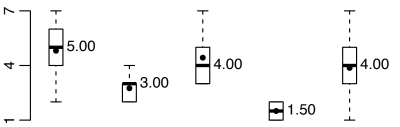
Table C.1: Summary statistics for the developer experience pre-questionnaire. For a description of each variable, identified by field number, see Appendix F. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

	BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Languages Used Lifetime* (field 7)	 10.00	6.50	7.00	8.00	8.00	**0.001**
Languages Used Often (field 9)	 4.00	2.00	3.00	2.50	3.00	**0.007**
Lines of Code Lifetime*† (KLOC) (field 10)	 50.00	25.00	35.00	85.00	35.00	0.329
Lines of Code Java‡ (KLOC) (field 11)	 12.50	5.75	4.50	7.00	9.00	**0.010**
Programming Hours Per Week (field 12)	 15.00	18.50	10.00	30.00	15.00	0.131
Programming Skill (scale 1–7)§ (field 13)	 3.00	2.00	4.00	3.50	3.00	0.261
Working Hours Per Week (field 15)	 0.00	10.00	0.00	25.00	0.00	0.054
Years Professional Experience (field 16)	 0.00	1.00	0.00	3.50	0.00	**0.029**

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)
* "Lifetime" means the number of languages (or LOC) the participant reports having *ever* used (or written).

† Two outliers not shown: BYU=8000 and UA=1000.
‡ One outlier not shown: BYU=4000.
§ 1=high skill, 7=low.

Table C.2: Summary statistics for the design pattern knowledge pre-questionnaire. For a description of each variable, identified by field number, see Appendix F. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).[†]

		BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Patterns Used Lifetime* (field 19)		8.00	6.50	5.00	4.00	6.00	0.077
Abstract Factory‡ (field 20)		3.00	3.00	3.00	2.50	3.00	0.454
Adapter (field 21)		3.00	3.50	4.00	2.00	3.00	**0.010**
Bridge (field 22)		1.00	2.00	3.50	2.00	3.00	<**0.001**
Chain of Responsibility (field 23)		2.00	2.00	4.00	2.50	3.00	**0.002**
Command (field 24)		4.00	5.00	4.00	2.50	4.00	**0.032**
Composite‡ (field 25)		3.00	5.00	4.00	2.00	4.00	**0.008**
Decorator‡ (field 26)		3.00	4.50	4.00	2.00	4.00	0.086
Factory Method (field 27)		4.00	2.50	3.00	2.50	3.00	**0.002**
Flyweight (field 28)		1.00	3.50	3.50	1.50	2.00	<**0.001**
Mediator (field 29)		1.00	2.00	4.00	2.00	2.00	<**0.001**
Memento (field 30)		1.00	2.00	4.00	2.00	2.00	<**0.001**
Observer (field 32)		5.00	5.00	4.00	3.50	5.00	0.104
Proxy (field 33)		2.00	5.50	4.00	2.50	4.00	<**0.001**
Reactor (field 34)		1.00	2.00	1.00	1.00	1.00	<**0.001**
Strategy (field 35)		2.00	3.00	4.00	2.00	3.00	**0.002**
Template Method (field 36)		2.00	2.00	3.50	2.00	3.00	**0.009**
Visitor (field 37)		5.00	3.00	4.00	1.50	4.00	<**0.001**

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)
* "Lifetime" means the number of patterns the participant reports having *ever* used.

[†] Individual patterns (fields 20–37) are self-assessed on an ordinal scale (1–7): 1=*never heard of it*, 2=*have only heard of it*, 3=*understand it roughly*, 4=*understand it well*, 5=*understand it well and have worked with it once*, 6=*understand it well and have worked with it two or three times*, 7=*understand it well and have worked with it many times.* The differentiation between levels 2–4 is subjective and may be sensitive to cultural influence. Since these values are frequent in the responses, cross-site comparisons may be problematic.

[‡] The design pattern is included in E_joint.

As for professional experience (i.e., working-hours-per-week and years-professional-experience), two of the UPM participants report outlier values (10 and 15 years experience, working 40 hours per week). Since UPM only contributes 6 participants, its median responses are much higher than those of the other three sites. Statistically, UPM differs from the other sites for years-professional-experience, but not for working-hours-per-week. However, with the two outliers excluded, the p-value for years-professional-experience becomes 0.203, indicating that the UPM outliers differ from all other participants, even those at UPM.

Thus, outliers account for most of the statistically significant differences in developer experience that we observe across sites. Excluding unrealistic outliers, only two significant cross-site differences remain: 1) Two of the UPM participants report significantly more professional experience than any other participant in the study, and 2) BYU participants report using more programming languages than participants at the other three sites (see languages-used-lifetime and languages-used-often). The BYU participants' advantage in language experience may be a product of their university's teaching style, or it may reflect a cultural tendency to exaggerate on survey questions. Coincidentally, BYU participants are also the primary contributor of unrealistic responses to the LOC questions.

C.2 Pattern Knowledge (Table C.2)

Statistically, the four sites differ on 14 of the 18 patterns surveyed. However, the patterns-used-lifetime variable, which describes *actual pattern use*, is not statistically significant. Additionally, only four of the statistically significant patterns involve a median exceeding 4.0 (i.e., "understand it well"), and the largest median among all patterns is only 5.5 (i.e., between "understand it well and have worked with it once" versus "understand it well and have worked with it two or three times"). Thus, cross-site differences in pattern knowledge almost exclusively concern exposure to design pattern concepts rather than experience implementing patterns.

Many of the individual patterns display similar box plot arrangements—for instance, Command, Composite, and Decorator. With a little effort, we can identify several potentially meaningful arrangements. However, since we measured 17 patterns, these groupings may simply be due to random chance. To test the possibility, we count the number of unique arrangements. Ordering sites by median, allowing means to break ties, we find 10 unique arrangements among the 17 patterns. If the pattern variables share dependencies, we would expect fewer unique arrangements than predicted by random chance. With 24 possible permutations, the probability of obtaining 10 or fewer by random chance is 0.085. Thus we cannot reject the null hypothesis that the observed groupings are due to random chance, though the p-value is suggestive.

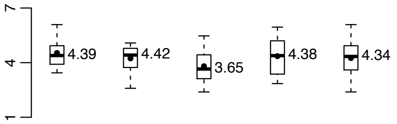
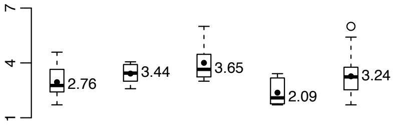
We also examine the order of sites overall, to see which ones tend to report higher pattern knowledge. We assign points to each site based on its rank for each of the 17 pattern knowledge variables—where rank 1 (i.e., highest pattern knowledge) is worth 3 points, rank 2 is worth 2 points, and so forth. As before, rank is determined by median, allowing means to break ties. For example, the rank order for the Command pattern is FUB, UA, BYU, UPM. Given this scheme, UA scores 41% of the total possible points, followed by FUB with 32%, then BYU with 17%, and UPM with 10%. Thus for an overwhelming number of patterns, UA and FUB report greater pattern knowledge than BYU and UPM. Incidentally, this effect explains why we find fewer unique permutations than expected.

Appendix D

Summary Statistics for Key Variables[1]

Tables D.1 and D.2 summarize key experiment variables. For each variable, we statistically compare sites using the Kruskal-Wallis rank sum test (which procedure compares medians, rather than means). We use a nonparametric (distribution-free) test because, in several cases, the data are not normally distributed. Histograms of the data are provided in the lab package.

Table D.1: Summary statistics for key explanatory variables (see Section 3.4 for a description of each variable). The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

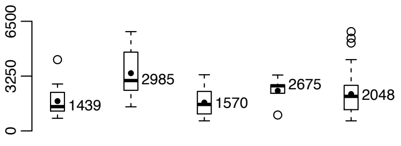
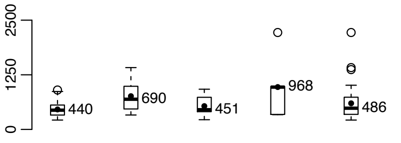
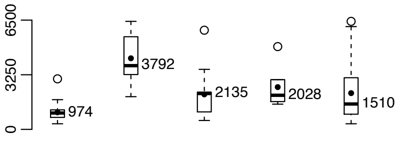
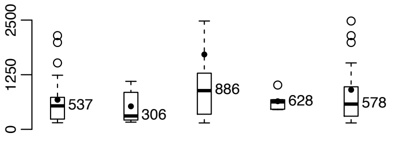
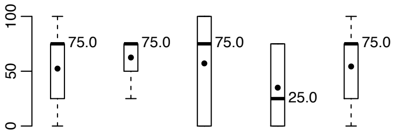
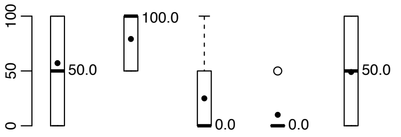
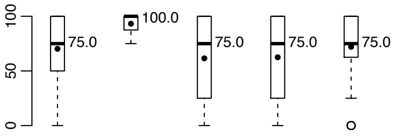
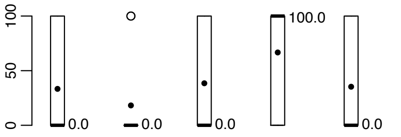
		BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Developer Experience (scale 1–7)*		4.39	4.42	3.65	4.38	4.34	0.158
Pattern Knowledge (scale 1–7)*		2.76	3.44	3.65	2.09	3.24	**0.001**

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)

* Aggregate metric. See Section 3.4 for additional scale information.

[1] Cited in Chapter 3.

Table D.2: Summary statistics for response variables. For a description of each variable, identified by field number, see Section 3.4 and Appendix F. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

	BYU (21) FUB (12) UA (14) UPM (6) All (53)	*sig.*
CO Task 1 Time (seconds) (field 66)		**0.002**
CO Task 2 Time (seconds) (field 69)		**0.050**
GR Task 1 Time (seconds) (field 75)		<**0.001**
GR Task 2 Time* (seconds) (field 78)		0.278
CO Task 1 Correctness (0–100%) (field 81)		0.608
CO Task 2 Correctness (0–100%) (field 87)		**0.001**
GR Task 1 Correctness (0–100%) (field 89)		0.147
GR Task 2 Correctness (0–100%) (field 91)		0.261

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)

*Two outliers not shown: UA=7239 and UA=5842.

Appendix E

Unusable Data[1]

In 8 cases, we exclude all of a participant's data from analysis as unusable. These data still appear in the data file provided in the lab package (with appropriate annotations), but are completely ignored for analysis.

- *10354*: The participant left both CO task 2 and 3 blank; the timings for those tasks are also both zero seconds, which requires clicking passed the tasks without even reading them; additionally, the participant reports "many interruptions" during the experiment, but does not provide sufficient information for us to correct his or her timings.
- *11088*: The participant quit the experiment before completing any program tasks.
- *31563*: The participant held a false assumption that the experiment was to be no more than 2 hours, as stated in the participant's comments.
- *36737*: The participant reported having to repeat the experiment due to website problems; many of his or her timings are extremely short (less than 10 seconds); presumably, the participant resubmitted previously competed solutions.
- *61820*: Instead of listing programming languages when requested, the participant responded as though s/he did not know any languages or completely misunderstood the question—e.g., "I don't know" and "None that I know of"; also his or her CO/GR task 1 timings are both extreme outliers, totalling 7.75 hours, which implies long, unrecorded breaks; further, the participant completed the 19-question pattern knowledge survey

[1] Cited in Chapter 3.

in only 13 seconds; given that his or her responses covered an array of categorical selections, such a low time suggests random choices.

- *63358*: Instead of listing programming languages when requested, the participant responded as though s/he did not know any languages—e.g., “Yes, I did. But not that much I’ve experience on this as I never worked in a software company” (sic) and “I worked in the telecom company and their I had experience with them very little” (sic); the participant also left CO task 3 completely blank, got all tasks completely wrong, and listed almost no developer experience on the pre-questionnaire; the participant does not appear to meet the minimum participation guidelines.
- *91072*: The participant’s CO task 1 timing is unreasonably low (76 seconds), whereas his or her CO task 2 timing is exceptionally high; conversely, the participant achieved a correctness of 50% on task 1, but got task 2 completely wrong; the participant appears to have used the browser back button to confuse the web portal’s timing mechanism.
- *94345*: The participant did not complete any of the program tasks.

We also partially exclude data for the following 3 participants. In these cases, we are missing data for either the CO or GR program:

- *15350*: The participant submitted GR source code for CO task 1, so we exclude all CO program data.
- *92863*: The participant submitted CO source code for GR task 1, so we exclude all GR program data.
- *95105*: The participant completed the questionnaires and the CO program tasks, but quit before completing the GR program tasks, so we exclude all GR program data.

Appendix F

Dataset Schema Definitions[1]

In this section, we describe the schema used for the E_joint dataset, which is provided in the lab package. The dataset includes 91 fields.

F.1 Experiment Metadata

These fields describe contextual or administrative elements of the experiment, such as research lab and treatment group assignments.

1. `id`: Unique participant id.
2. `group_idMOD4`: Treatment group (computed as *participant ID* % 4); corresponds to the alternation of program *order* (CO/GR) and program *variant* (PAT/ALT), as described in Section 3.3.3.
3. `researchLab`: The research team providing the given participant's data, where:

    - BYU=*Brigham Young University.*
    - FUB=*Freie Universität Berlin.*
    - UA=*The University of Alabama.*
    - UPM=*Universidad Politécnica de Madrid.*

4. `experimentLang`: The programming language in which the given participant completed the experiment, corresponding to one of three options: C++, C#, or Java. In our data,

[1] Cited in Chapter 3.

this value is always Java. BYU and UA required Java; the FUB and UPM participants all chose Java.

5. `experimentLangRequired`: Identifies whether the particular research team required their participants to use a specific language for the experiment (corresponding to "TRUE"), or whether the participants were allowed to choose their preferred language (corresponding to "FALSE").

F.2 Pre-Questionnaire 1—Developer Experience

Each field (except for 7, 9, and 18) represents one question on the developer experience survey. Exact questions are provided in the lab package. All scores are self-assessments. For summary statistics, see Appendix C.

6. `langsUsedLifetime`: A comma-separated list of programming languages used at least once in the participant's lifetime (listed in the order given by the participant). Spelling, capitalization, and punctuation have been standardized.
7. `langsUsedLifetime_count`: A count of the number of languages listed in field 6.
8. `langsUsedOften`: A comma-separated list of programming languages known well by the participant and worked with several times (listed in the order given by the participant). Spelling, capitalization, and punctuation have been standardized.
9. `langsUsedOften_count`: A count of the number of languages listed in field 8.
10. `locLifetime`: The total number of lines of code that the participant has ever written in any programming language. The value in this field should be greater than or equal to that of field 11.
11. `locJava`: The total number of lines of code that the participant has ever written in Java. The value in this field should be less than or equal to that of field 10.
12. `progHoursPerWeek`: Hours per week in which the participant reads, writes, or modifies code.

13. `progSkill`: The participant's self-assessed programming skill, relative to all other programmers in the world, where: 1=*top 10%*, 2=*top 25%*, 3=*top 40%*, 4=*average*, 5=*bottom 40%*, 6=*bottom 25%*, 7=*bottom 10%.*

14. `studentStatus`: The participant's current student status, where: 1=*undergraduate*, 2=*graduate*, 3=*postgraduate*, 4=*non-student.*

15. `workHoursPerWeek`: Hours per week spent by the participant working as a "professional software developer." This question referred to the same work as that of field 16.

16. `yearsProfExp`: Years spent by the participant working as a "professional software developer." This question referred to the same work as that of field 15.

17. `major`: Main course of study; this question was stated such that it was to be answered only if the participant is currently a student. Thus a non-null answer should indicate that the participant is currently a student and should correspond with a value of 1, 2, or 3 for field 14.

18. `major_translated`: Translation of field 17 from German/Spanish into English.

F.3 Pre-Questionnaire 2—Design Pattern Knowledge

Each field represents one question on the pattern knowledge survey. Exact questions are provided in the lab package. All scores are self-assessments. Fields 20–37 represent individual patterns. All but two of the individual patterns were originally defined by Gamma et al. [77]. Field 31, the Multistructor pattern, does not actually exist and was included as a sanity check. Field 34, the Reactor pattern, was originally defined by Schmidt et al. [185, pp. 179–214]. Scores for the individual patterns are based on a 7-point ordinal scale: 1=*never heard of it*, 2=*have only heard of it*, 3=*understand it roughly*, 4=*understand it well*, 5=*understand it well and have worked with it once*, 6=*understand it well and have worked with it two or three times*, 7=*understand it well and have worked with it many times*. For summary statistics, see Appendix C.

19. `patternsUsedLifetime`: The number of software design patterns with which the participant has ever worked.
20. `abstractFactory`: Abstract Factory pattern.
21. `adapter`: Adapter pattern.
22. `bridge`: Bridge pattern.
23. `chainOfResponsibility`: Chain of Responsibility pattern.
24. `command`: Command pattern.
25. `composite`: Composite pattern.
26. `decorator`: Decorator pattern.
27. `factoryMethod`: Factory Method pattern.
28. `flyweight`: Flyweight pattern.
29. `mediator`: Mediator pattern.
30. `memento`: Memento pattern.
31. `multistructor`: Multistructor is *not* an actual pattern. This field was included as a sanity check. Most participants answered 1 (*never heard of it*). A few answered as high as 3 (*understand it roughly*), which is not too surprising out of 61 participants, since the term "multistructor" could reasonably be confused with other patterns. Most importantly, no one answered 4 (*understand it well*) or above. Also, of those participants who answered 2 or 3, all appear (based on their other responses) to have conscientiously completed the experiment.
32. `observer`: Observer pattern.
33. `proxy`: Proxy pattern.
34. `reactor`: Reactor pattern.
35. `strategy`: Strategy pattern.

36. `templateMethod`: Template Method pattern.

37. `visitor`: Visitor pattern.

F.4 Task Responses

These fields include responses submitted for short-answer tasks, as well as responses to post-task questionnaires. Note that the coding tasks (CO/GR task 1s) involved submitting source code; thus they do not appear here. Instead, the participants' source code solutions are included in the lab package.

Concerning fields 47–48 and 56–57, the participants appear to have interpreted these questions in two different ways. Most participants responded with values less than 100%—e.g., 50%, presumably meaning 50% less time, or half the time it would have taken. A few participants responded with values exceeding 100%—e.g., 300%, presumably meaning 300% faster.

38. `CO_task2`: The participant's short-answer response to CO task 2.

39. `CO_task2_translated`: Translation of field 38 from German/Spanish into English.

40. `CO_task3`: The participant's short-answer response to CO task 3.

41. `CO_task3_translated`: Translation of field 40 from German/Spanish into English.

42. `CO_patternsNoticed`: A comma-separated list of design patterns (listed in the order given by the participant), which the participant reports having noticed in the CO program. Spelling, capitalization, and punctuation have been standardized.

43. `CO_difficulty`: The participant's assessment of combined difficulty for all three CO program tasks, where: 1=*quite easy*, 2=*reasonably easy*, 3=*neither easy nor difficult*, 4=*reasonably difficult*, 5=*quite difficult*.

44. `CO_confidence`: The participant's self-reported confidence (as a percentage) that s/he has correctly solved the three tasks for the CO program.

45. `CO_difficultAspects`: Aspects of the CO program tasks which the participant found most difficult.

46. `CO_difficultAspects_translated`: Translation of field 45 from German/Spanish into English.

47. `CO_patKnowHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the CO program tasks due to his/her personal "knowledge of the design patterns used in the program."

48. `CO_docHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the CO program tasks due to "the explicit documentation of the design patterns used in the program."

49. `GR_task2`: The participant's short-answer response to GR task 2.

50. `GR_task2_translated`: Translation of field 49 from German/Spanish into English.

51. `GR_patternsNoticed`: A comma-separated list of design patterns (listed in the order given by the participant), which the participant reports having noticed in the GR program. Spelling, capitalization, and punctuation have been standardized.

52. `GR_difficulty`: The participant's assessment of combined difficulty for the two GR program tasks, where: 1=*quite easy*, 2=*reasonably easy*, 3=*neither easy nor difficult*, 4=*reasonably difficult*, 5=*quite difficult*.

53. `GR_confidence`: The participant's self-reported confidence (as a percentage) that s/he has correctly solved the two tasks for the GR program.

54. `GR_difficultAspects`: Aspects of the GR program tasks which the participant found most difficult.

55. `GR_difficultAspects_translated`: Translation of field 54 from German/Spanish into English.

56. `GR_patKnowHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the GR program tasks due to his/her personal "knowledge of the design patterns used in the program."

57. `GR_docHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the GR program tasks due to "the explicit documentation of the design patterns used in the program."

F.5 Final Comments

These fields document all comments provided by participants at the end of the experiment. Note that participants also provided useful comments in fields 45 and 54.

58. `finalComments`: Any final comments the participant submitted after completing the experiment.

59. `finalComments_translated`: Translation of field 58 from German/Spanish into English.

F.6 Survey and Task Times

These fields record web page timings. All timings represent the time spent (in seconds) on the associated web portal page, which is *not* necessarily equivalent to the time spent working—e.g., the participant may have taken a break or been interrupted. Note in this regard that some participants mention in their final comments having taken breaks during the experiment. In most cases, we apply time adjustments to correct for interruptions (see fields 64, 68, 73, and 77). In one case, due to insufficient information, we can only record the problem (see fields 70 and 79) and exclude the data during analysis.

60. `devExpSurveyTime`: The time spent by the participant on the development experience pre-questionnaire page.

61. `patKnowledgeSurveyTime`: The time spent by the participant on the pattern knowledge pre-questionnaire page.

62. `CO_task1DownloadTime`: The time spent by the participant on the source code download page for CO task 1 (note that this time may include working time—e.g., the participant may have begun reading code before moving on to the task description page).

63. `CO_task1WorkTime`: The time spent by the participant on the description page for CO task 1.

64. `CO_task1WorkTimeCorr`: Time that is to be added to CO task 1 to correct for breaks reported by the participant in his/her final comments.

65. `CO_task1UploadTime`: The time spent by the participant on the solution upload page for CO task 1 (note that this time may include working time—e.g., the participant may have proceeded to this page after reading the task description, but before having worked on the task).

66. `CO_task1TotalTime`: The total time (with corrections) spent by the participant on the download, description, and upload pages for CO task 1 (i.e., the sum of fields 62–65). This is the time recommended for analysis.

67. `CO_tasks2-3WorkTime`: The time spent on the task page for CO tasks 2 and 3. These tasks were presented on the same page and timed together. Tasks 2 and 3 were short-answer questions that did not require the download, modification, or upload of source code.

68. `CO_tasks2-3WorkTimeCorr`: Time that is to be added to CO tasks 2 and 3 to correct for breaks reported by the participant in his/her final comments.

69. `CO_tasks2-3TotalTime`: The total time (with corrections) spent by the participant on the task page for CO tasks 2 and 3 (i.e., the sum of fields 67 and 68). This is the time recommended for analysis.

70. `CO_dataValid`: Specifies whether the data for the CO program tasks should be considered valid for the given participant. "FALSE" indicates that the participant's data are either known to be invalid or are so anomalous (e.g., impossibly small timings) that they

cannot reasonably be considered as valid. “TRUE” indicates that the data appear to be valid. A parenthetical note indicates that the data are suspect—i.e., the researcher should consider them with caution. Parenthetical notes include a list of anomalous data fields.

71. `GR_task1DownloadTime`: The time spent by the participant on the source code download page for GR task 1 (note that this time may include working time—e.g., the participant may have begun reading code before moving on to the task description page).

72. `GR_task1WorkTime`: The time spent by the participant on the description page for GR task 1.

73. `GR_task1WorkTimeCorr`: Time that is to be added to GR task 1 to correct for breaks reported by the participant in his/her final comments.

74. `GR_task1UploadTime`: The time spent by the participant on the solution upload page for GR task 1 (note that this time may include working time—e.g., the participant may have proceeded to this page after reading the task description, but before having worked on the task).

75. `GR_task1TotalTime`: The total time (with corrections) spent by the participant on the download, description, and upload pages for GR task 1 (i.e., the sum of fields 71–74). This is the time recommended for analysis.

76. `GR_task2WorkTime`: The time spent on the task page for GR task 2. Task 2 was a short-answer question that did not require the download, modification, or upload of source code.

77. `GR_task2WorkTimeCorr`: Time that is to be added to GR task 2 to correct for breaks reported by the participant in his/her final comments.

78. `GR_task2TotalTime`: The total time (with corrections) spent by the participant on the task page for GR task 2 (i.e., the sum of fields 76 and 77). This is the time recommended for analysis.

79. `GR_dataValid`: Specifies whether the data for the GR program tasks should be considered valid for the given participant. "FALSE" indicates that the participant's data are either known to be invalid or are so anomalous (e.g., impossibly small timings) that they cannot reasonably be considered as valid. "TRUE" indicates that the data appear to be valid. A parenthetical note indicates that the data are suspect—i.e., the researcher should consider them with caution. Parenthetical notes include a list of anomalous data fields.

F.7 Task Correctness Scores

These fields list the correctness scores assigned to participant solutions and short-answer responses. All scores are percentages (0–100%). A suffix of "`LabGrade`" indicates that the scores were assigned by the research team specified in field 3—i.e., BYU, FUB, UA, or UPM. A suffix of "`BYUGrade`" indicates that the scores were assigned by the BYU research team, which regraded all solutions to ensure consistency across sites.

80. `CO_task1LabGrade`: CO task 1 correctness, assessed by the individual research team.
81. `CO_task1BYUGrade`: CO task 1 correctness, assessed by the BYU research team.
82. `CO_task2LabGrade`: CO task 2 correctness, assessed by the individual research team.
83. `CO_task2BYUGrade`: CO task 2 correctness, assessed by the BYU research team.
84. `CO_task3LabGrade`: CO task 3 correctness, assessed by the individual research team.
85. `CO_task3BYUGrade`: CO task 3 correctness, assessed by the BYU research team.
86. `CO_tasks2-3LabGrade`: Average of fields 82 and 84. This field corresponds with field 69.
87. `CO_tasks2-3BYUGrade`: Average of fields 83 and 85. This field corresponds with field 69.
88. `GR_task1LabGrade`: GR task 1 correctness, assessed by the individual research team.
89. `GR_task1BYUGrade`: GR task 1 correctness, assessed by the BYU research team.
90. `GR_task2LabGrade`: GR task 2 correctness, assessed by the individual research team.
91. `GR_task2BYUGrade`: GR task 2 correctness, assessed by the BYU research team.

Appendix G

Data Preparation Process[1]

Producing the final dataset involved three steps:

1. *Merging individual datasets from the four research teams:* Given that the web portal provides all data in a fixed schema, this step was trivial.
2. *Unifying terminology and format:* This step involved correcting spelling errors, as well as matching capitalization and punctuation across columns to facilitate readability. We modified only the four columns representing lists of either programming languages or design patterns (e.g., changing "Decoratr" and "decorator pattern" both to read as "Decorator"). These four fields were provided by the participants as free-form text. We did not alter the list orderings, nor modify any other columns. To ensure consistency, we made these changes using spreadsheet tools (i.e., spell checking and search/replace tools).
3. *Annotating the data:* This step involved three sub-processes: 1) a column search for data errors (none were found); 2) a column search for outliers; and 3) a row search for participants who deviated from the instructions. All anomalies are recorded in the data file as annotations. The data file is provided in the lab package. We have added two columns to the data file to describe data validity: `CO_dataValid` and `GR_dataValid`. For an explanation of how to read these columns, see fields 70 and 79 in Appendix F. We have also added columns for recording time corrections (fields 64, 68, 73, and 77 in Appendix F), which we inferred from the participants' final comments.

[1] Cited in Chapter 3.

The data preparation process involved only minor syntactic corrections for readability, as described in Step 2 above. All other anomalies were simply annotated. In cases where correct values are known, those values are recorded in the annotations. It is left up to the analyst to deal with anomalous data as s/he sees fit, based on the annotations provided.

Appendix H

Variables Excluded from Analysis[1]

In this section, we list the fields we exclude from statistical analysis, with an explanation for each. Future work may find some of these fields useful. Field numbers correspond to those shown in Appendix F.

- `group_idMOD4` (field 2): Group is a surrogate for the four combinations of program *order* and *variant*, so it is redundant.
- `experimentLang` (field 4): All participants used the same programming language (Java), so this field does not differentiate participants.
- `experimentLangRequired` (field 5): This field conveys little information beyond that already provided by field 3, `researchLab`.
- `langsUsedLifetime` (field 6): This field is replaced by field 7, `langsUsedLifetime_count`.
- `langsUsedOften` (field 8): This field is replaced by field 9, `langsUsedOften_count`.
- `studentStatus` (field 14): This field can be viewed as a high-level indicator of developer experience. However, it does not necessarily represent developer experience. For instance, in our data, student status does not correlate with `progSkill` or `progHoursPerWeek` (0.02 and 0.06, respectively)—both of which seem directly related to developer experience.[2] Although student status could represent some other important concept, we

[1] Cited in Chapter 3.

[2] Pearson product-moment correlation coefficients (calculated with R 2.15.2); two-sided p-values = 0.89 and 0.65, respectively.

exclude it from analysis because, within the context of the PatMain study, it only seems relevant as a measure of developer experience.

- `workHoursPerWeek` and `yearsProfExp` (fields 15 and 16): We ignore working-hours-per-week and years-professional-experience for two reasons. First, both questions ask about "professional" work. Second, in hindsight, the term *professional* seems ambiguous for students. Should students count part-time work, or do these questions refer only to full-time work? Many participants (18) do, in fact, report part-time hours. However, more than half of the participants (29 of 53) report zero hours and zero years experience, at least some of whom may be working in professional software companies, but do not consider their jobs to be "professional" because the work is not full time.
- `major` (field 17): All participants are essentially the same major, so this field conveys little information.
- `patternsUsedLifetime` (field 19): This field includes several unreasonable outliers. We could deal with these outliers by bucketing or otherwise transforming the data. However, estimating the number of patterns one has ever used seems more difficult than estimating one's knowledge of specific patterns. Thus we prefer to use the individual pattern knowledge assessments instead (fields 20–37).
- `multistructor` (field 31): This field was included as a sanity check. No such design pattern actually exists.
- `CO_task2`, `CO_task3`, and `GR_task2` (fields 38, 40, and 49): These fields are replaced by correctness scores (fields 87 and 91).
- `devExpSurveyTime` (field 60): Survey times are unrelated to the experiment hypotheses.
- `patKnowledgeSurveyTime` (field 61): Survey times are unrelated to the experiment hypotheses.
- Component task times (including download, work, upload, and corrections; fields 62–65, 67–68, 71–74, and 76–77): We use the total times instead (sum of the component times)

because of uncertainty in how the participants navigated the task download, description, and upload pages.

- `CO_dataValid` and `GR_dataValid` (fields 70 and 79): These fields apply only to pre-analysis decisions concerning which data entries are valid.
- "`LabGrade`" correctness scores (fields 80, 82, 84, 86, 88, and 90): For consistency across sites, we use the centrally-graded BYU scores instead.
- `CO_task2BYUGrade` and `CO_task3BYUGrade` (fields 83 and 85): Like E_repl, we combine CO tasks 2 and 3. Thus these fields are replaced by field 87, `CO_tasks2-3BYUGrade`.
- Post-questionnaire data and final comments (fields 42–48, 51–57, and 58): Where applicable, we apply these responses qualitatively to help interpret the statistical results.

Appendix I

Derived Metrics[1]

In this appendix, we describe the derivation of several metrics used in the statistical analysis. Metrics not mentioned here are described sufficiently in the main paper.

I.1 Developer Experience

To compose the developer experience metric, we transform, scale, and average 6 component metrics (field numbers correspond to those shown in Appendix F):

1. `langsUsedLifetime` (field 6)
2. `langsUsedOften` (field 8)
3. `locLifetime` (field 10)
4. `locJava` (field 11)
5. `progHoursPerWeek` (field 12)
6. `progSkill` (field 13)

For a description of each component, see Appendix F. For summary statistics, see Appendix C. The aggregation process is accomplished in 6 steps:

1. Replace each response for `langsUsedLifetime` and `langsUsedOften` with its cardinality.
2. Apply a natural log transformation to each response for `langsUsedOften`, `locLifetime`, and `locJava` to correct for skew and outliers.

[1] Cited in Chapter 3.

3. Scale all component variables (`langsUsedLifetime`, `langsUsedOften`, `locLifetime`, `locJava`, `progHoursPerWeek`, and `progSkill`) to a range of 1–7, such that zero maps to 1 and the variable's maximum value maps to 7. Note that `progSkill` is already on a 1–7 scale. However, for that variable 7 initially represents low skill; thus the scale must be reversed.
4. Average the components `langsUsedLifetime` and `langsUsedOften` to create a single variable representing the general concept, `langsUsed`.
5. Average the components `locLifetime` and `locJava` to create a single variable representing the general concept, `locWritten`.
6. Finally, average the four variables `langsUsed`, `locWritten`, `progHoursPerWeek`, and `progSkill` to produce the final aggregate developer experience metric.

The final metric is a continuous variable ranging from 1 to 7 (scaled to match the range of the pattern knowledge metric), where 7 represents high experience. The metric is an average of four core components—languages used, LOC written, programming hours per week, and self-assessed programming skill—with each component receiving a weight of 25% in the average. The four components (after pre-averaging) only moderately correlate (from 0.2 to 0.4), which is ideal for creating an effective aggregate metric. The components measure related concepts, yet each component incorporates unique information.

Additional notes:

- Log transformation impacts the Pearson product-moment correlation, but not the rank correlation. Scaling impacts neither.
- We pre-average some variables (steps 4 and 5) in order to reduce their collective impact on the final average. In each case, the variables are highly correlated—0.87 in the case of the LOC metrics and 0.63 in the case of the languages used metrics.[2] These correlations

[2] All correlations in this section are Pearson product-moment correlation coefficients (calculated with R 2.15.2). Parametric tests are appropriate for these data because they have been previously normalized via log transformation.

are significantly higher than those for any other pair of variables, with the next highest being only 0.43. Thus we pre-average in cases where multiple variables measure a similar concept to prevent that concept from having excessive influence on the final metric.

- A few entries in the dataset are clearly erroneous. In most cases, no correction is possible because we have no idea what the true responses should have been. However, in two cases we do adjust the data prior to computing the aggregate metric. For participants 38048 and 92689, we append a copy of their `langsUsedOften` response to the end of their `langsUsedLifetime` response. In both cases, the participants report disjoint sets for these variables. Conversely, all other participants report `langsUsedOften` as a subset of `langsUsedLifetime`.

I.2 Java Familiarity

The java familiarity metric was the only metric we tested in the statistical models that turned out to be completely unhelpful. The metric is derived from the languages-used-often variable, based on the following ordinal scale: 1=*the participant does not list any object oriented languages*, 2=*the participant lists only non-Java object oriented languages*, 3=*the participant lists Java*. Most participants (47/53) list Java, so this variable has little statistical impact. Thus, we ignore it in the discussion, other than to mention that we explored it.

Appendix J

Statistical Model Assumptions[1]

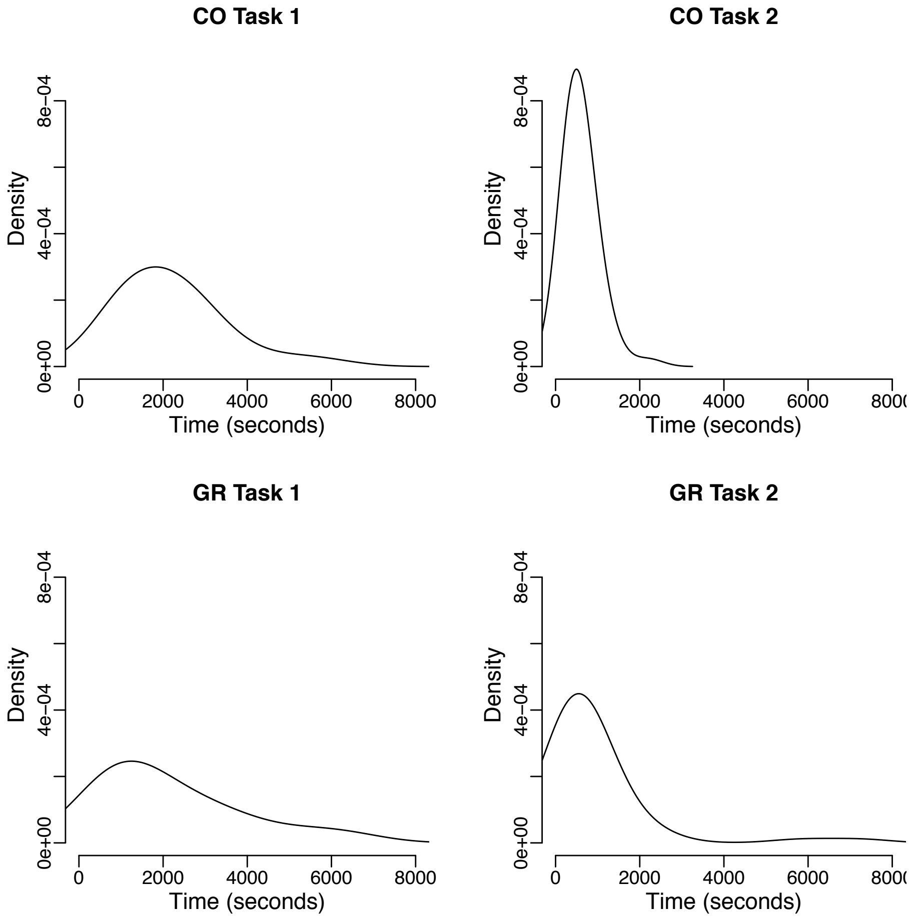
The frequentist models depend on several assumptions, including: 1) response variables should be normally distributed; 2) explanatory variables should *not* be significantly correlated; and 3) explanatory variables should be homoscedastic (i.e., of constant variance, as opposed to heteroscedastic). Given our setup, assumptions 2 and 3 also apply to the Bayesian models.

J.1 Normality

The *time* data are skewed with several significant outliers. For the frequentist analysis, we normalize *time* by log-transformation (see Figures J.1–J.2 and Table J.1). However, for the Bayesian analysis, we model *time* using a gamma distribution, so normality is not an issue in that case. Concerning *correctness*, the range is discretized into only five buckets, thus precluding the possibility of gross outliers (a primary threat to model validity and a principle reason for concern with normality). Also, with such a limited range, *correctness* cannot take on much of a skew. Thus, we do not apply a log transformation to *correctness*. That said, with only five buckets, the *correctness* variable is unlikely to fit a smooth normal distribution. Thus, although we assume normality for the frequentist analysis, we avoid that assumption entirely in the Bayesian analysis by modeling *correctness* with a beta distribution.

[1] Cited in Chapter 3.

Figure J.1: Density plots for the *time* response variable *before* log transformation. Compare to Figure J.2 and Table J.1.

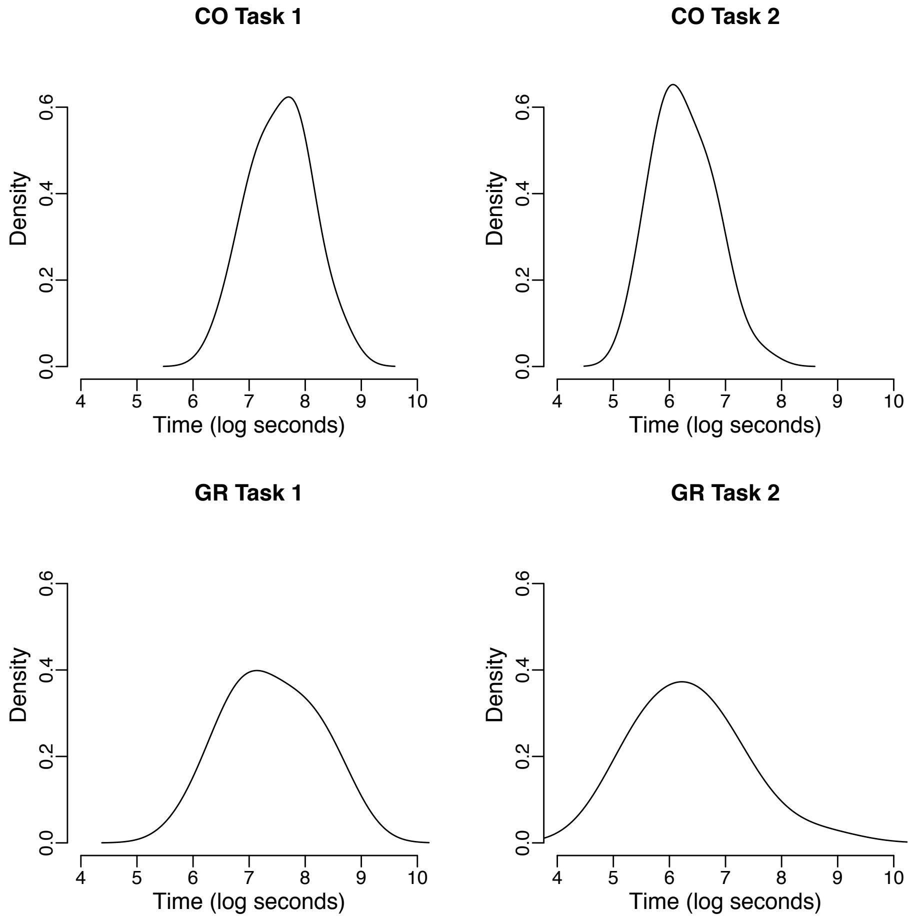


J.2 Multicollinearity

Multicollinearity occurs when two or more explanatory variables in a statistical model are significantly correlated. In the presence of multicollinearity a model's predictive power and overall reliability are not impacted. However, parameter estimates for the collinear variables themselves may be inaccurate and can change erratically with only small changes to the data. For instance, two highly significant, but collinear variables can both appear insignificant when included in the model together.

All correlations between explanatory variables for the models considered in this paper are low. Generally, multicollinearity is not a problem for pairwise correlation magnitudes

Figure J.2: Density plots for the *time* response variable *after* log transformation. Compare to Figure J.1 and Table J.1.



below 0.7 [57],[2] and the highest magnitude among our explanatory variables is only 0.4 (between *devExp* and *time_ln* on GR task 2).[3] Thus multicollinearity is not a concern. A complete list of all correlations is included in the lab package.

J.3 Heteroscedasticity

Heteroscedasticity occurs when the variance in a dataset differs across sub-populations (e.g., treatment groups). This condition can be seen by simply plotting each explanatory variable

[2] Most texts cite correlation thresholds in the range 0.5 to 0.9 as indicating potential multicollinearity [57, 68]. Also, note that correlation magnitudes are not a direct measure of collinearity, and they can fail to detect the condition in some cases. However, *all* collinearity detection methods are subject to some error, and rule-of-thumb correlation thresholds have been shown to perform at least as well as the more complicated methods [57].

[3] We use Pearson product-moment correlation coefficients (calculated with R 2.15.2). Parametric tests are appropriate in this case because the data have been normalized via log transformation.

Table J.1: Results for the Shapiro-Wilk test of normality for the *time* response variable before/after log transformation (calculated with R 2.15.2). p-values represent the probability of obtaining a given sample, assuming a normally distributed population—i.e., low p-values indicate risk of non-normality. Compare to Figures J.1 and J.2.

	Shapiro-Wilk p-value *Before* Log Transformation	Shapiro-Wilk p-value *After* Log Transformation
CO Task 1	2.16×10^{-04}	0.755
CO Task 2	9.17×10^{-07}	0.275
GR Task 1	1.86×10^{-05}	0.286
GR Task 2	1.52×10^{-11}	0.044*

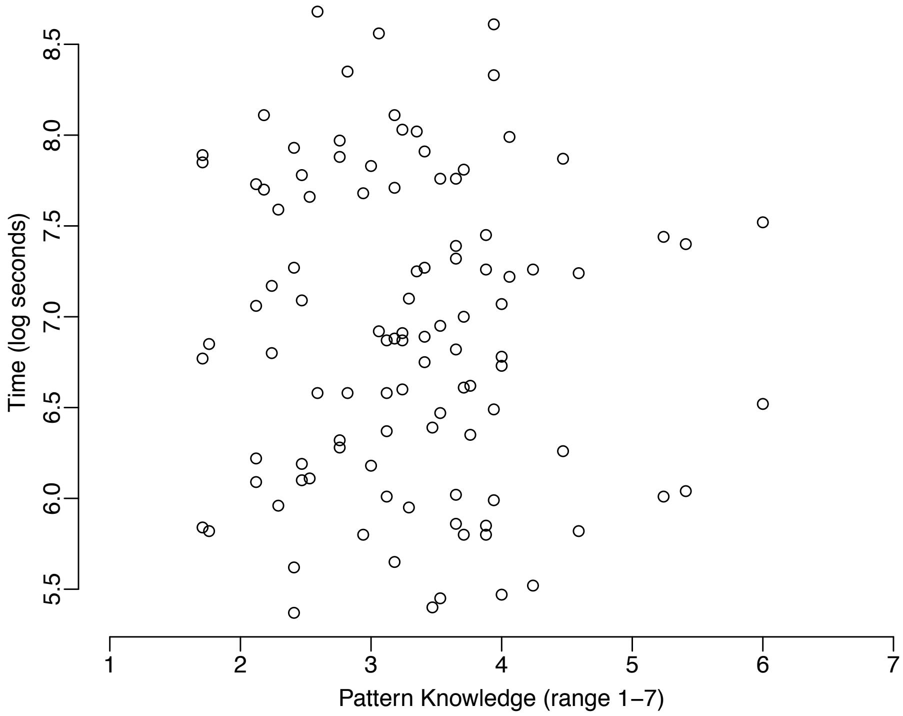
*This p-value indicates possible non-normality. However, that conclusion depends on a single extreme outlier, participant 90620, without which the p-value becomes 0.211. Also, the log-time density plot for GR Task 2 appears roughly normal even with the outlier (see Figure J.2), and our final results ultimately exclude participant 90620 (as described in Section 3.6.1). Thus, we are not concerned about normality in this case.

against the response variable. If any expanding, shrinking, or multi-modal variance patterns are visible across the range of the explanatory variable, then heteroscedasticity is a concern. We provide scedasticity plots for all explanatory variables in the lab package.

For our dataset, scedasticity plots indicate concern in only one case—*patKnow*. For *patKnow*, only 13% of participants score above 4.0, such that the variance appears lower in the upper range (see Figure J.3). Our statistical models assume constant variance, which may be true, but the data are simply too sparse in the upper range to know for sure.

If *patKnow* is heteroscedastic, the statistical results would only be minimally impacted. First, *heteroscedasticity does not bias least squares coefficient estimates, so none of our parameter estimates in the frequentist analysis would be affected.* Second, heteroscedasticity does impact variance estimates, which in turn can bias p-values. However, such a bias would primarily affect only the upper range of *patKnow*, and it would most likely mean inflated p-values. Specifically, since most of the data reside in the lower range, variance for the upper range (if it is inaccurate) is likely overestimated by the common variance term. Consequently, if we were to allow our models to estimate separate variances for high and low *patKnow*,

Figure J.3: Scedasticity plot for *patKnow* (CO_time model). Data sparseness in the upper range (>4.0) indicates the possibility of heteroscedasticity.



and if the data sparseness problem were resolved, the resulting p-values would more likely decrease or remain unchanged than to increase. *In other words, if our analysis is biased with respect to pattern knowledge, it is most likely biased toward type 2 errors—failure to reject the null hypothesis; furthermore, any such bias would apply only to the upper range.*

Appendix K

Tuning Frequentist Models[1]

We tune all frequentist models using a standard covariate pruning technique [174, p. 345]. The technique is essentially a modified form of backward stepwise regression. We avoid basic stepwise regression because it effectively constitutes data dredging (i.e., fishing for significance), which can seriously bias the results [174, pp. 353–354]. The primary difference between the modified form (which makes it appropriate) and the basic form is that we drop all main effects from the model before performing the elimination procedure. This way, tuning does not manipulate the final results. The main effects are only added back to the model once tuning is complete. Discarded covariates are still adjusted for in the final models, since they had a chance to be included prior to adding the main effects back in [174, pp. 345–347]. Conceptually, the tuning process acts as a high-level filter that removes any covariates unrelated to the response variable.

We use p-values for the elimination criterion. The process requires iteratively removing the least significant covariate until all remaining covariates are at least moderately significant (p-values $\lesssim 0.1$). For all models, we treat program *variant* and all interactions as main effects. After returning the main effects to the model, the final step is to drop any non-significant interactions. In general, we never include an interaction without including its lower order terms. Thus we always drop high-order interactions first. Pruning interactions does reintroduce the concern of data dredging. However, given the nearly complete lack of significance that the interactions have in all models, removing them is appropriate and the threat of data dredging is minimal.

[1] Cited in Chapter 3.

Appendix L

Discretization of Bayesian Variables[1]

As explained in Section 3.5, for the Bayesian models, we discretize all continuous explanatory variables into *low* and *high* categories. We divide variables based on a visual inspection of clustering and/or by conceptually interpreting the variable's scale. The resulting partitions for each variable are follows:

- *devExp*: 2 buckets, representing high and low developer experience (high = scores of 4.5–7.0 inclusive, matching 22 of 53 participants).
- *patKnow*: 2 buckets, representing high and low pattern knowledge (high = scores of 3.5–7.0 inclusive, matching 21 of 53 participants).
- *time* (when used as a covariate for *correctness*): 2 buckets, representing high and low work times. High times are determined on a per-task basis:
  - CO task 1: $\geq$2000 sec. (26 of 52 observations)
  - CO task 2: $\geq$500 sec. (25 of 52 observations)
  - GR task 1: $\geq$1900 sec. (22 of 51 observations)
  - GR task 2: $\geq$700 sec. (18 of 51 observations)
- *correctness* (when used as a covariate for *time*): 2 buckets, representing high and low solution correctness (high = scores of 75–100 inclusive, matching 103 of 206 observations).

[1] Cited in Chapter 3.

Appendix M

Bayesian Priors[1]

To avoid biasing the Bayesian analysis, we enlisted an external researcher to provide estimates for all priors. Our helper—who had 5 years of experience managing professional developers, as well as 10 years of experience teaching undergraduate and graduate computer science students—is an expert in the area of Bayesian statistics. To inform our helper, we gave him mean and variance data for all CO and GR tasks from E_orig. We did not give him any data from E_joint. The priors he selected are shown in Table M.1.

We gave our helper only two constraints in selecting the priors (both suggestions of Felt [69]): First, we instructed him to center all priors—with the exception of the variances and base offset—at zero, thus assuming no effect by default (i.e., the null hypothesis). Second, we instructed him to select broad priors. Doing so allows the posterior distributions to move more easily in response to the data, thus minimizing the weight of our biases in the analysis. In general, choosing broad priors leads to broader posteriors, but for a post-hoc analysis, sacrificing some precision is an acceptable tradeoff in order to focus the analysis more on the data. After all, the purpose of a post-hoc analysis is to formulate data-driven conjectures.

In selecting the priors, our helper assumed that the student participants would take longer and score lower (on average) than the professionals from E_orig. He also assumed that they would display greater variance than the professionals. Our helper chose normal distributions for most parameters primarily because we have no reason to believe anything

[1] Cited in Chapter 3.

Table M.1: Prior distributions for all Bayesian model parameters.

Response Variable	Models	CO Task 1 Variance	CO Task 2 Variance	GR Task 1 Variance	GR Task 2 Variance	Base Offset	All Other Parameters
time	T1–T6	$\Gamma\left(3, 480000\right)$	$\Gamma\left(3, 83333\right)$	$\Gamma\left(3, 403333\right)$	$\Gamma\left(3, 163333\right)$	$\mathrm{N}\left(1900, 500^2\right)$ $3\sigma = 25\,\mathrm{min.}$	$\mathrm{N}\left(0, 1000^2\right)$ $3\sigma = 50\,\mathrm{min.}$
correctness	C1–C6	$\Gamma\left(2, 0.07\right)$	$\Gamma\left(2, 0.08\right)$	$\Gamma\left(2, 0.055\right)$	$\Gamma\left(2, 0.12\right)$	$\mathrm{N}\left(0.5, (0.4/3)^2\right)$ $3\sigma = 40\,\mathrm{pts.}$	$\mathrm{N}\left(0, 0.3^2\right)$ $3\sigma = 90\,\mathrm{pts.}$

$\Gamma(k, \theta)$ = gamma distribution, where k and θ represent shape and scale.
$\mathrm{N}(\mu, \sigma^2)$ = normal distribution, where μ and σ^2 represent mean and variance.
3σ = the approximate practical range of a normal distribution on either side of the mean.

other than symmetric noise. He chose gamma distributions for the variances because the support for gamma is limited to values greater than zero.[2]

[2] Inverse gamma is a common choice for variance, but given our use of Gibbs sampling, conjugacy was not necessary.

Appendix N

Notes on Observation Filtering[1]

E_repl used observation filtering when modeling the *time* response variable. As Vokáč et al. explain, “Since completion times have little meaning for solutions with low correctness, only those solutions achieving correctness score 4 (‘almost correct’) or 5 (‘correct’) were used in [the *time* analysis]” [212, p. 158]. Scores of 4 and 5 in E_repl’s data correspond to scores of 75% and 100%, respectively, in our data. Although the authors do not explain why low-scoring solutions are problematic, one concern is that the associated timings may be *censored* (i.e., artificially capped). For example, some participants may have submitted an incomplete solution simply because they were tired of the task.

Observation filtering can mitigate the problem of censored data, but it also limits the generality of the results. Further, from a statistical standpoint, observation filtering actually addresses two separate issues: 1) it accounts for variance due to an interaction between *correctness* and *variant*; and 2) it accounts for variance due to a relationship between *correctness* and *time* (i.e., participants may score higher simply by working longer). Thus, to improve on the use of observation filtering, we address these issues separately.

N.1 *correctness* × *variant* Interaction

Many conditions could induce an interaction between *correctness* and *variant*. For instance, added noise from data censoring could mask the effect of *variant* at low levels of *correctness*. We test for a *correctness* × *variant* interaction in the Bayesian models. For those models, we

[1] Cited in Chapter 3.

Table N.1: Unfiltered Bayesian results showing the *correctness* × *variant* interaction and the marginalized effect of *variant* for each of the four tasks (uT4:46–54,387–406). Probabilities exceeding a significance (*sig.*) of 0.75 are bolded. Insignificant probabilities are those near 0.5.

program	*task*	*correctness*	ALT−PAT	p(ALT>PAT)	*sig.*
CO	1	low	120	0.61	0.61
CO	1	high	−373	**0.17**	**0.83**
CO	1	*marg.*	−126	0.39	0.61
CO	2	low	−69	0.40	0.60
CO	2	high	135	0.67	0.67
CO	2	*marg.*	−46	0.51	0.51
GR	1	low	−409	**0.23**	**0.77**
GR	1	high	317	**0.78**	**0.78**
GR	1	*marg.*	33	0.53	0.53
GR	2	low	−28	0.47	0.53
GR	2	high	−408	**0.17**	**0.83**
GR	2	*marg.*	−218	0.32	0.68

GALT−PAT = the difference between variants (in seconds)—i.e., the difference between the posterior distribution means.
p(ALT>PAT) = posterior probability that the ALT variant takes longer than the PAT variant.
sig. = significance of *variant* (i.e., max of p and $1-p$, where p is the posterior probability).
marg. = marginal posterior probability for *variant*, factoring out its interaction with *correctness*.

divide *correctness* into low and high buckets, based on the same threshold used by E_repl (as shown in Appendix L).

Table N.1 shows the *correctness* × *variant* interaction and the marginalized effect of *variant* for each task. The table indicates a relatively strong interaction between *correctness* and *variant*. First, *variant* is more significant within the interaction than as a standalone variable. Second, for all tasks, the effect of *variant* (denoted ALT−PAT in the table) varies considerably across *correctness* levels (low, high). For instance, on GR task 1, PAT is estimated to take 409 seconds *longer* than ALT when correctness is low. However, for high correctness, PAT requires 317 seconds *less* than ALT—a difference of 12.1 minutes. For tasks requiring 20–30 minutes to complete, even 5 minute differences can be significant.

Thus, we do find an interaction between *correctness* and *variant*. However, that interaction is inconsistent across tasks. Therefore, observation filtering is inadvisable. Excluding low-scoring data is simply too blunt a method to account for the complex *correctness* × *variant* relationship we observe across tasks.

We confirm this conclusion by applying observation filtering to the frequentist models. After filtering, the effect of *variant* is completely lost. Thus the significance of *variant* depends on both low and high correctness scores, and consequently, we cannot filter individual observations based solely on correctness. For additional discussion of the *correctness* × *variant* interaction, see the analysis of moderators in Section 3.6.2.

N.2 *correctness-time* Relationship

We test for a relationship between *correctness* and *time* by including *correctness* as a covariate in the *time* models (and vice versa). The Bayesian models (see Table N.2) indicate that *correctness* and *time* are likely related in E_joint's data. The frequentist models further reveal that the relationship varies by program, being highly significant for the GR program, but less so for CO.

The *correctness-time* correlation is positive in all models—i.e., the participants are likely achieving higher scores at the expense of time. The magnitude of the effect is modest. For example, in the GR_time model, a 10-point increase in correctness corresponds with a 5.9% increase in time. The other three frequentist models show similar results (see Tables X.1, X.5, and X.13 in Appendix X). According to the Bayesian models, achieving a high score (75% or 100%) is associated with a 2–4 minute increase in time (which could be significant relative to 20–30 minute tasks).

E_repl also included *correctness* as a covariate in the *time* models, but found it to be insignificant. However, E_repl only tested *correctness* after having filtered the data. If a relationship did exist, it was likely lost due to the filtering. As a test, we applied observation filtering to our own data, exactly as described by Vokáč et al., and reran the frequentist

Table N.2: Unfiltered frequentist and Bayesian results showing the significance of *correctness* as a covariate in the *time* models (and vice versa). p-values less than or equal to 0.05 and posterior probabilities exceeding 0.75 are bolded. All p-values are two-sided. All posterior probabilities describe the probability of a positive correlation between *correctness* and *time*.

			Posterior
Model	Covariate	p-value	Probability[*]
CO_time	correctness	0.071	-
CO_correctness	time_ln	0.110	-
GR_time	correctness	<**0.001**	-
GR_correctness	time_ln	**0.003**	-
T1	correctness	-	**0.99**
T2	correctness	-	**0.97**
T3	correctness	-	**0.91**
T4	correctness	-	0.68[†]
T5	correctness	-	**0.95**
T6	correctness	-	**0.90**
C1	time	-	**0.85**
C2	time	-	**0.84**
C3	time	-	**0.82**
C4	time	-	0.63[†]
C5	time	-	**0.81**
C6	time	-	**0.88**

[*] Source: uT1–T6,C1–C6:61–63.

[†] For these models only, the covariate is interacted with other variables. In both cases, the interaction requires estimating 16 parameters, rather than 2. The increase in parameters dampens statistical significance [174, p. 347].

models. Similar to E_repl, *time* and *correctness* both became insignificant in all models after the filtering (p-values ≥ 0.23). Thus we do find a significant, though modest, relationship between *correctness* and *time*, and observation filtering does account for that relationship.

N.3 Summary

We find significant evidence for an interaction between *correctness* and *variant*, but that interaction is inconsistent across tasks. Thus we cannot exclude low-scoring observations without eliminating the main effect. Further, we find evidence for a small positive correlation between *correctness* and *time*, which is fully accounted for by observation filtering. However, since observation filtering is not acceptable, we must account for the *correctness-time* relationship by including *correctness* as a covariate in the *time* models (and vice versa), rather than by filtering.

We conclude that observation filtering, as implemented by E_repl, does not reduce cross-site variance for E_joint. As a method for reducing irrelevant variance, it is simply too inefficient. We also propose that the analysis of E_repl could be improved by addressing the *correctness* × *variant* interaction and the *correctness-time* relationship separately, via statistical modeling, rather than through observation filtering.

Appendix O

Notes on Participant Filtering[1]

In this section, we provide additional information on the participant filtering discussed in Section 3.6.1.

O.1 Filtering by Correctness

Figures O.1, O.2, and O.3 present different views of the E_joint participants, plotted by average task time and correctness. Figure O.1 was the plot used by the four independent reviewers to decide the filtering threshold. Figure O.2 shows the participants categorized by site. Figure O.3 adds ID labels and the filter threshold. Notice that the American universities (BYU and UA) account for most of the participants filtered.

Figure O.4 is a post-hoc validation of the participant filtering—i.e., we generated it only *after* choosing the filter threshold. Figure O.4 shows the distribution of participants from E_joint whose data were identified during the annotation process as questionable. For example, participant 15350 reported having previously written zero lines of code in any language. For a complete list of such concerns, see the data file in the lab package. Figure O.4 also shows the distribution of the professionals from E_orig. E_orig's participants completed tasks on paper, rather than on a computer, so comparisons to E_joint are only tentative. That said, notice that the questionable data mostly fall to the left of the filter, whereas all of E_orig's participants (the professionals) fall to the right.

[1] Cited in Chapter 3.

Figure O.1: E_joint participants plotted by average task time and correctness. Plot used by the four independent reviewers to select the filter threshold.

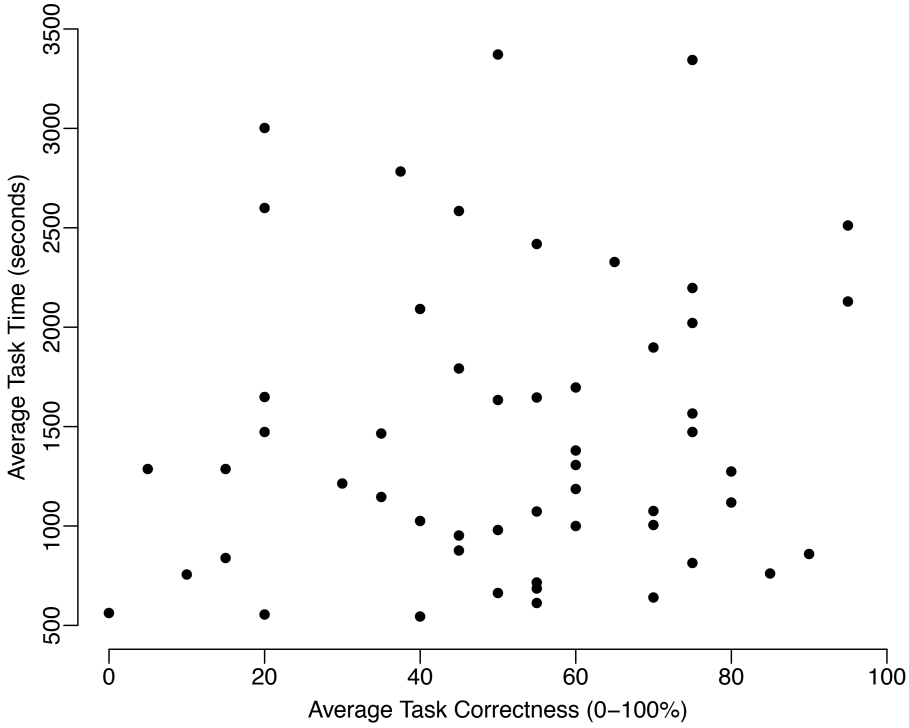


Figure O.2: E_joint participants plotted by average task time and correctness. Same as Figure O.1, but with participants categorized by site.

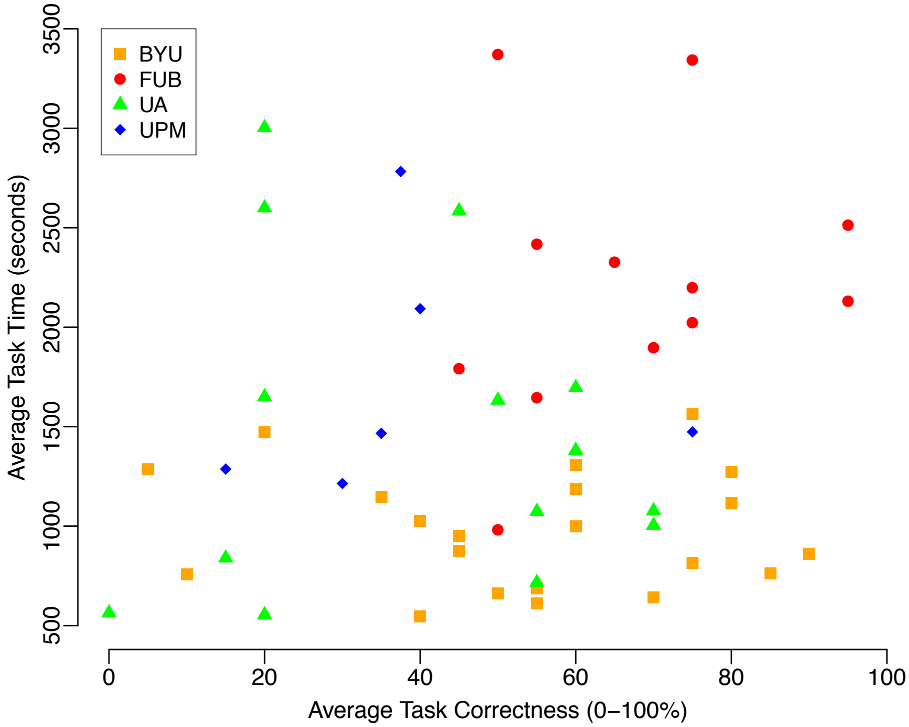


Figure O.3: E_joint participants plotted by average task time and correctness. Same as Figure O.2, but including participant IDs and filter threshold.

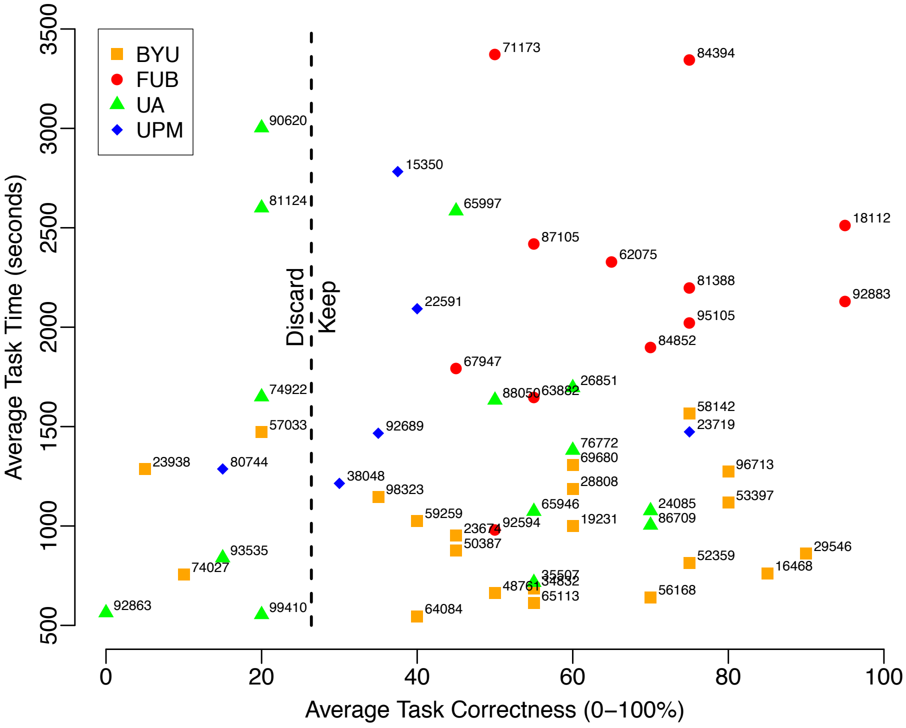
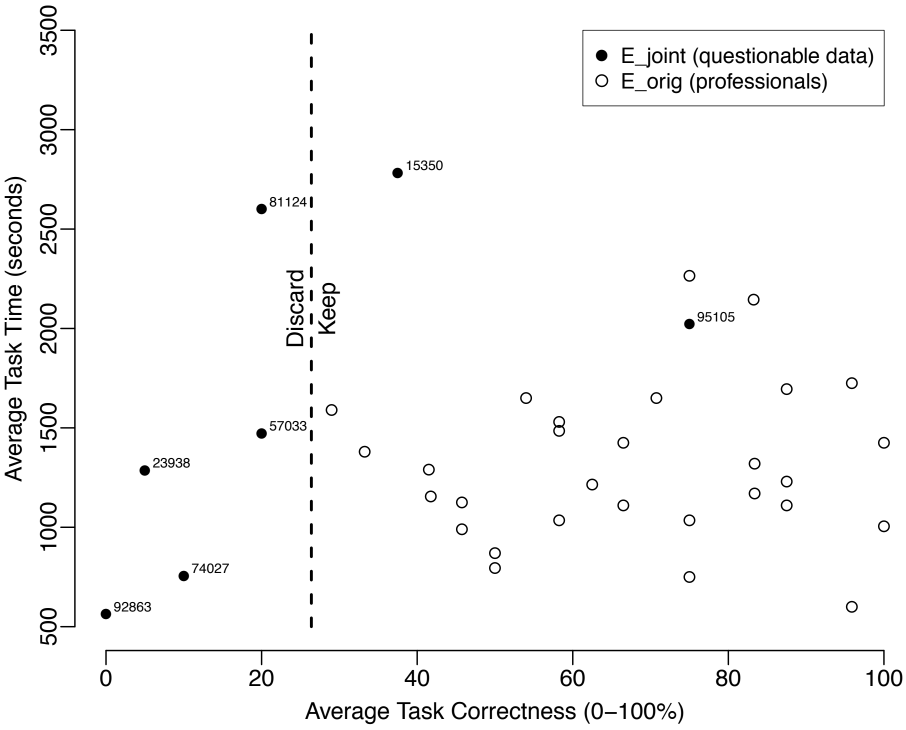


Figure O.4: Post-hoc validation of the filter threshold, showing questionable data from E_joint, as well as professionals from E_orig.



Figures O.5 and O.6 show *time* and *correctness* displayed according to the *site* × *variant* interaction. The plots depict the data before and after participant filtering. We include the figures for two reasons: 1) to show that the main effect, program *variant*, significantly varies across the four sites; and 2) to show that participant filtering only marginally reduces that variance. See Sections 3.6.1–3.6.2 for further discussion.

O.2 Filtering by Time

In addition to filtering by *correctness*, we also asked the reviewers to select a threshold for filtering by *time* (again, based on Figure O.1). Possibly, high times indicate underqualified participants, similar to low correctness. Or more likely, high times indicate technical difficulties or the taking of unrecorded breaks. For example, at the outset of the analysis we discarded all data for participant 57033 because s/he reported having spent more than an hour setting up an IDE.

Each reviewer selected a different threshold (approximately 2250, 2550, 2650, and 3200 seconds). We tested each threshold and found that none substantially affect the results when applied in addition to the *correctness* filter (discussed above). If anything, filtering by *time* slightly reduces the significance of *variant*—likely due to the loss of good data. Therefore, like E_repl, we do not exclude participants based on work times.

Figure O.5: Time data, showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.

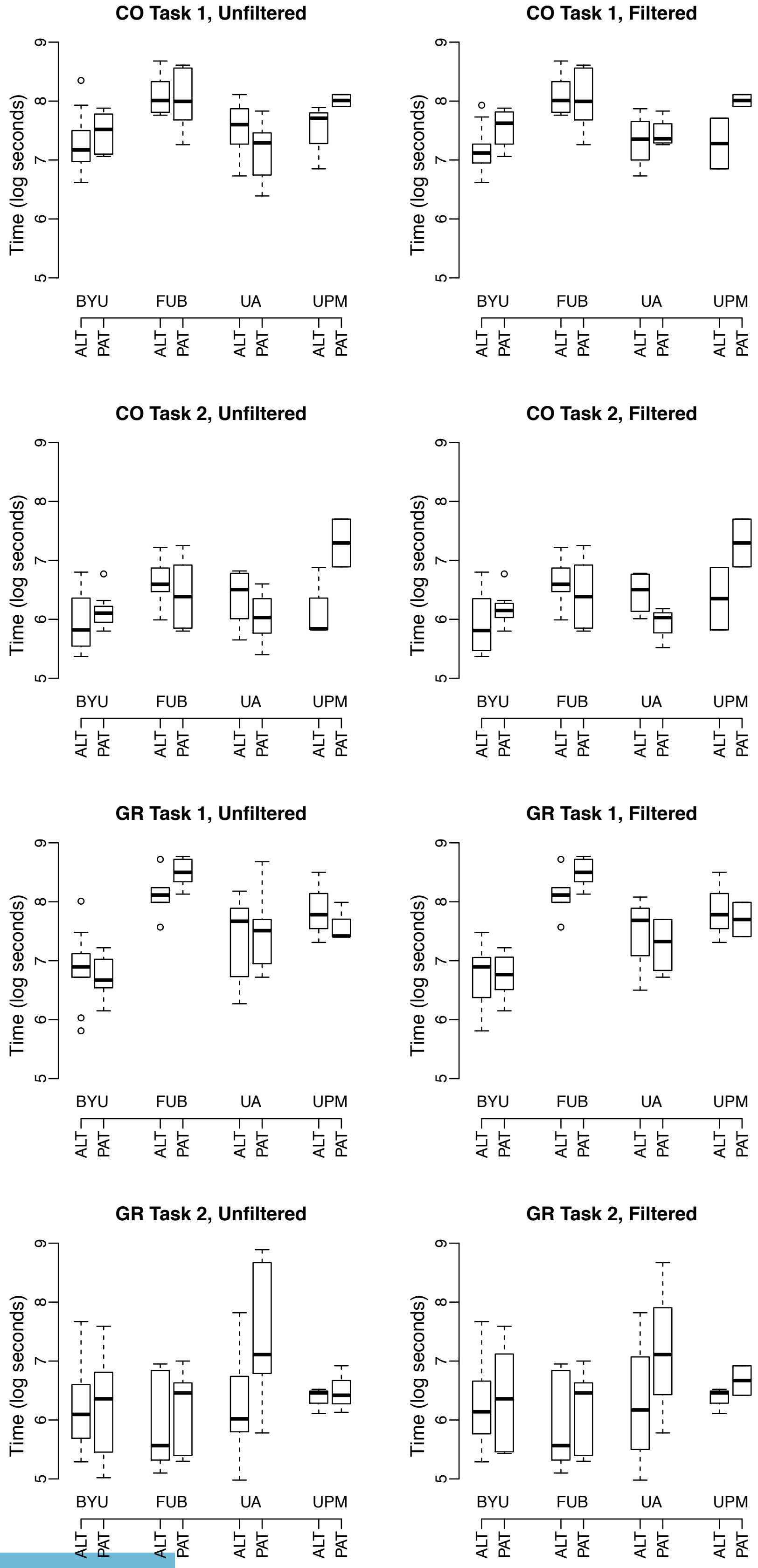
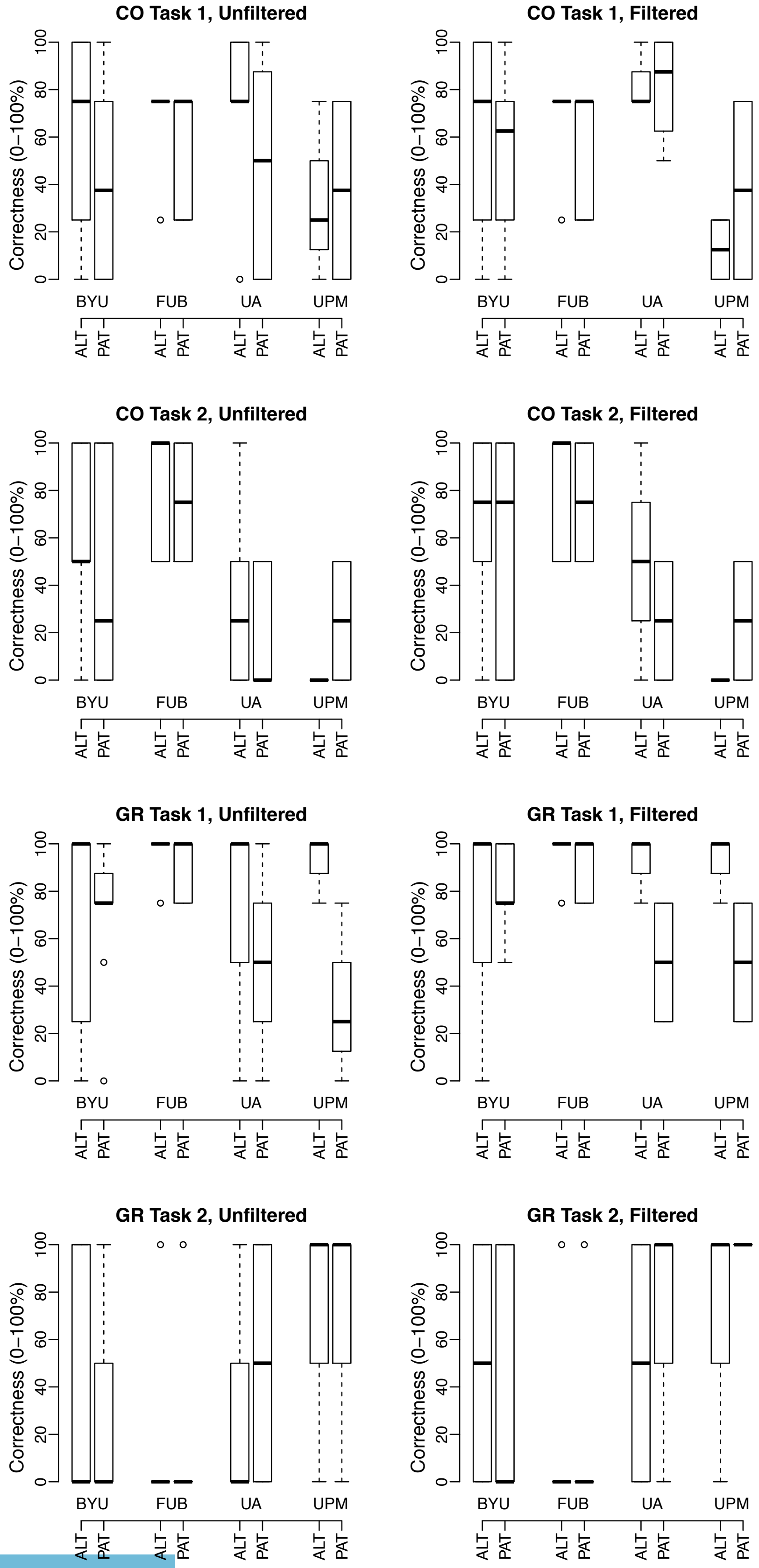


Figure O.6: Correctness data, showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.



CO Task 1, Unfiltered

CO Task 1, Filtered

CO Task 2, Unfiltered

CO Task 2, Filtered

GR Task 1, Unfiltered

GR Task 1, Filtered

GR Task 2, Unfiltered

GR Task 2, Filtered

Appendix P

Visualization of Moderator Variables[1]

In this section, we provide an extended version of Table 3.5 from Section 3.6.2. Table P.1 extends Table 3.5 in three ways: 1) it shows probabilities for additional interactions; 2) it visualizes the probabilities via box plots; and 3) it provides back references to the Bayesian results tables, mapping the box plots to the posterior probabilities on which they are based. Below we explain how to read Table P.1. The explanation assumes familiarity with Table 3.5.

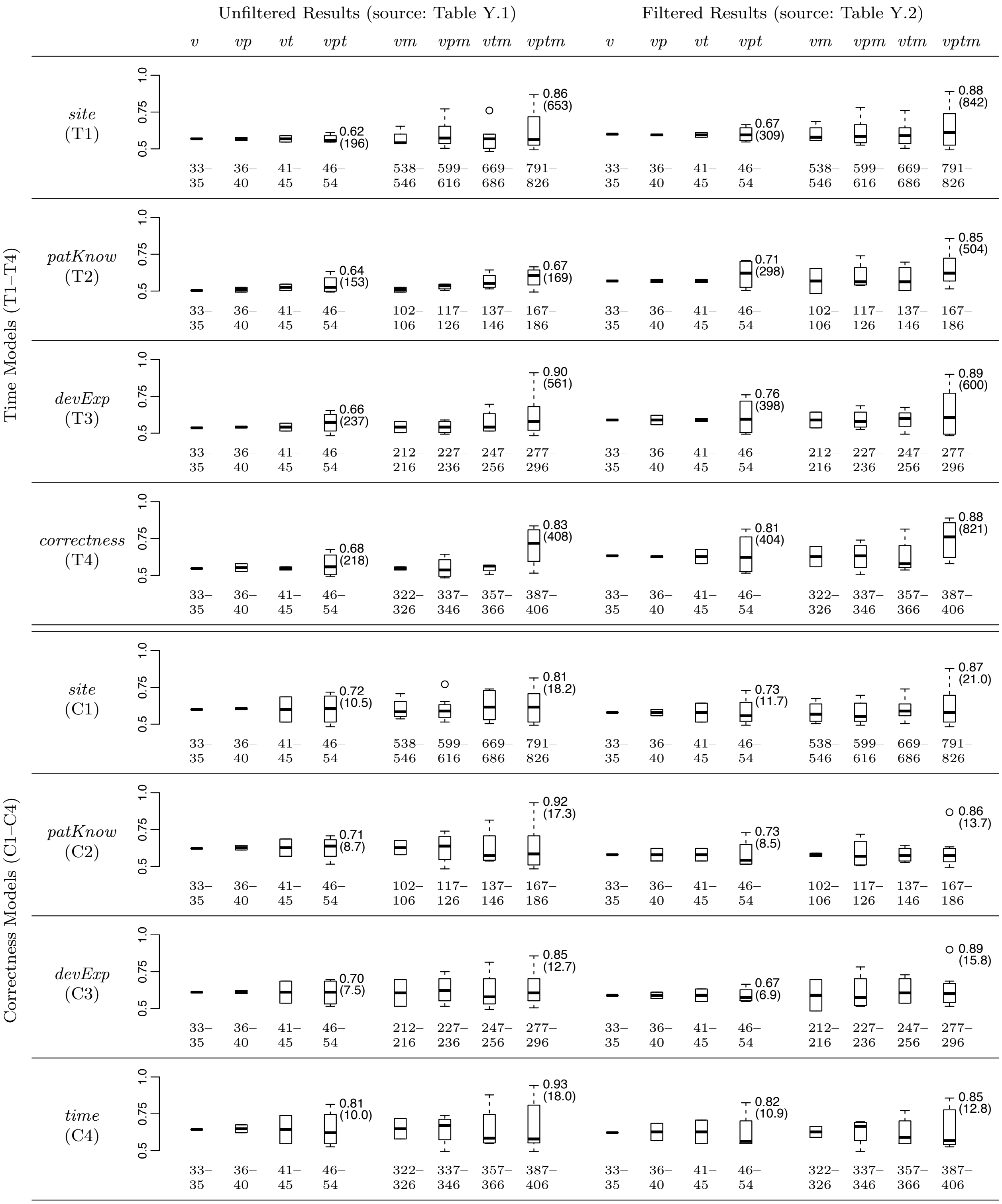
In Table P.1, we replace Table 3.5's column labels, $\neg m$ and m, with the labels *vpt* and *vptm* (where $v = variant$, $p = program$, $t = task$, and $m = mod$). We then add six additional interaction columns: *v*, *vp*, *vt*, *vm*, *vpm*, and *vtm*. We include the additional columns to show that, in most cases, the significance of *variant* depends not only on the given moderator, but also on *program* and *task*—i.e., *variant* is most significant in the four-way interactions (labeled *vptm*). Thus, we focus in Chapter 3 on the interactions *vpt* and *vptm*.

In Table P.1, instead of listing only the max significance for each interaction, we use box plots to show the full range of significances. Viewing the full range is helpful in order to see how the spread of probabilities changes across the interactions. For the *vpt* and *vptm* interactions, we label the max significance values and (in parentheses) the corresponding effect estimates. The labeled values match the numbers shown in Table 3.5.

Table P.1 also includes back references to the Bayesian results tables. The back references are important as an audit trail for the analysis. They also allow the reader to compare interactions in more detail, if desired. To locate the source data for a given box plot, note the table identified in the header (i.e., Table Y.1 or Y.2 from Appendix Y), the Bayesian

[1] Cited in Chapter 3.

Table P.1: Bayesian interaction results—moderator assessment. Expanded version of Table 3.5 from Section 3.6.2. See Appendix P for a description of how to read and interpret this table.



model listed on the left (i.e., one of T1–T4 or C1–C4), and the row numbers provided directly below the box plot. For example, source data for the top left box plot can be found in Table Y.1, column T1, lines 33–35 (uT1:33–35).

Appendix Q

Moderator Variables Continued[1]

This material is a continuation of Section 3.6.2.

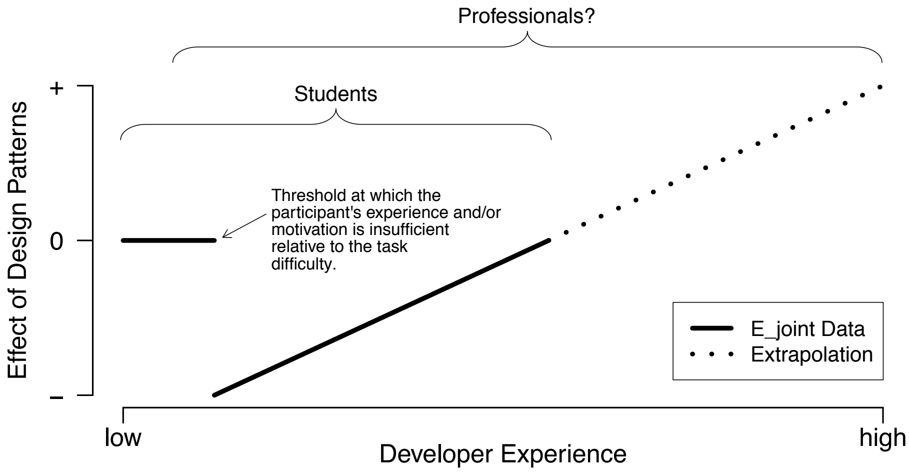
Q.1 Task Difficulty

Nearly twice as many E_joint participants complained in their post-questionnaire comments about the difficulty of the CO tasks, as compared to GR (13 versus 7). The participants also assessed the CO tasks as being slightly more difficult, and they reported feeling slightly less confident in their CO solutions. Correspondingly, *variant* is less significant in the CO_time model than in the GR_time model (before filtering, CO_time p-value = 0.925, GR_time p-value = 0.016). Also, filtering low-scoring participants strongly increases the significance of *variant* in the CO_time model, but has little impact on the GR_time model (CO_time p-value shifted from 0.925 to 0.019, GR_time p-value shifted from 0.016 to 0.025).

If the CO tasks were more difficult than the GR tasks, then presumably the low-scoring (i.e., underqualified/undermotivated) participants failed so badly that they masked the main effect in the unfiltered CO_time model. In the case of GR_time, however, those same participants did not struggle as much, and so the main effect is detectable without filtering. Thus, *the data suggest that a threshold of experience/motivation exists, which is dependent on task difficulty, and below which a developer will fail so badly that design patterns have no measurable impact.* We depict this threshold in Figure Q.1, incorporated into the moderating effect of developer experience.

[1] Cited in Chapter 3.

Figure Q.1: Relative impact that developer experience has on the effect of design patterns, taking into account the co-moderating influence of task difficulty. Since the graph depends on a specific level of task difficulty, the axes are depicted as relative scales. A positive effect for design patterns (+) means that the patterns lead to lower work times and higher quality solutions.



Interestingly, E_orig and E_repl both found *variant* to be more significant for the CO program than for GR. For example, E_orig obtained p-values < 0.001 for CO, but all p-values for GR were in the range 0.02–0.17. Thus, the filtering—which is explained in part by task difficulty—brings E_joint's results into greater alignment with the prior two PatMain studies.

Q.2 *correctness* and *time*

Table Q.1 shows the frequentst results for *correctness* and *time*.[2] As covariates for one another, both *correctness* and *time* are clearly meaningful irrespective of filtering, especially for the GR program. The *correctness-time* correlation is estimated to be positive in all models—i.e., the participants are achieving higher scores at the expense of time. For example, the unfiltered GR_time model indicates that a 10-point increase in correctness corresponds with a 5.9% increase in time (80% CI: 4–8). The other frequentist models show similar results (see Appendix X).

The most likely variable to account for a *positive* correlation between *correctness* and *time* is *motivation*. As previously discussed, some participants were significantly more

[2] The relationship between *correctness* and *time*, as well as the *correctness* × *variant* interaction, are also discussed in Appendix N.

Table Q.1: Frequentist model p-values for *correctness* and *time*. p-values less than or equal to 0.05 are bolded.

Model	Covariate	Unfiltered	Filtered
CO_time	correctness	0.071	0.104
CO_correctness	time	0.110	NS
GR_time	correctness	<**0.001**	<**0.001**
GR_correctness	time	**0.003**	<**0.001**

NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix K.

Table Q.2: Bayesian interaction results—moderator assessment. Excerpt from Table 3.5 in Section 3.6.2, showing models T4 and C4. Included here for convenience.

time	Unfiltered		Filtered		*correctness*	Unfiltered		Filtered	
Models	$\neg m$	m	$\neg m$	m	Models	$\neg m$	m	$\neg m$	m
correctness	0.68	**0.83**	**0.81**	**0.88**	*time*	**0.81**	**0.93**	**0.82**	**0.85**
(T4)	218	408	404	821	(C4)	10.0	18.0	10.9	12.8

motivated than others. Particularly at FUB, the participants appear to have been willing to take longer in order to do a better job (see also the discussion of "perceived time limits" below). Thus, the frequentist results for *correctness* and *time* reinforce the importance of analyzing motivation as a potential moderator in future design pattern studies.

Table Q.2 shows the Bayesian interaction results for *correctness* and *time*. Prior to filtering, the *correctness* and *time* interactions appear significant, especially in the *time* models. After filtering, however, the interactions are largely eliminated. Since filtering mitigates the interactions, we anticipate that the interactions were primarily due to the fact that *variant* has little impact on underqualified and/or undermotivated participants (as discussed in the prior subsection on "task difficulty"). Thus, upon removing those participants, the interaction disappears and general significance for the main effect increases. Overall, these results support the need to filter underqualified and undermotivated participants, as was done in E_repl. They also represent indirect evidence that motivation could moderate the effect of *variant*, since the filtering targeted (in part) undermotivated participants.

Table Q.3: Frequentist and Bayesian results showing the significance and effect of program *order*. p-values less than or equal to 0.05 and posterior probabilities exceeding 0.75 are bolded. All p-values are two-sided.

	Unfiltered		Filtered	
Model	*sig.*	1st−2nd	*sig.*	1st−2nd
CO_time	**0.007**	296	**0.004**	306
CO_correctness	NS	-	NS	-
GR_time	**0.002**	452	**0.038**	325
GR_correctness	NS	-	NS	-
T1	**0.994**	189	**0.964**	141
T2	**0.998**	228	**0.978**	175
T3	**0.998**	223	**0.981**	180
T4	**0.996**	214	**0.970**	164
T5	**0.996**	210	**0.970**	157
T6	**0.995**	189	**0.967**	142
C1	**0.865**	−3.4	**0.874**	−3.8
C2	**0.827**	−3.1	**0.850**	−3.6
C3	**0.852**	−3.5	**0.883**	−4.5
C4	**0.781**	−2.3	**0.852**	−3.6
C5	**0.799**	−2.6	**0.898**	−4.3
C6	**0.803**	−2.7	**0.812**	−3.1

sig. = significance of *order*—i.e. p-values for the frequentist models (first four rows), posterior probabilities for the Bayesian models (source: *u*T1–T6,C1–C6:11–13 and *f*T1–T6,C1–C6:11–13).
1st−2nd = estimated difference between program orderings (in seconds or percentage points).
NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix K.

Q.3 Program Order

Table Q.3 shows the effect of program *order* in the frequentist and Bayesian models. For the *time* models, *order* is statistically significant, with effect estimates in the range 2.4–7.5 minutes. An effect size of 5 minutes could be meaningful given that each task required only 20–30 minutes to complete. For the *correctness* models, *order* is less significant, and its effect is fairly small (2.3–4.5 percentage points). Based on these results, we conclude that *the participants spent significantly less time, and possibly scored slightly higher, on whichever* *program was second*. Since performance tended to improve on the second program, the most likely explanation is a learning effect.

Interestingly, both E_orig and E_repl found *order* to be *in*significant. Since E_joint used the same basic design, the experiment proper likely did not cause the learning effect. Instead, we believe the effect is due to the initial setting up of development environments. E_joint's participants were given only two things: task instructions and source code. They had to set up their own development environments, including importing the source code. The effort required to setup IDEs could certainly account for extra time on the first program—especially since E_joint's participants were students, who may not have known prior to the experiment how to import existing code into an IDE.

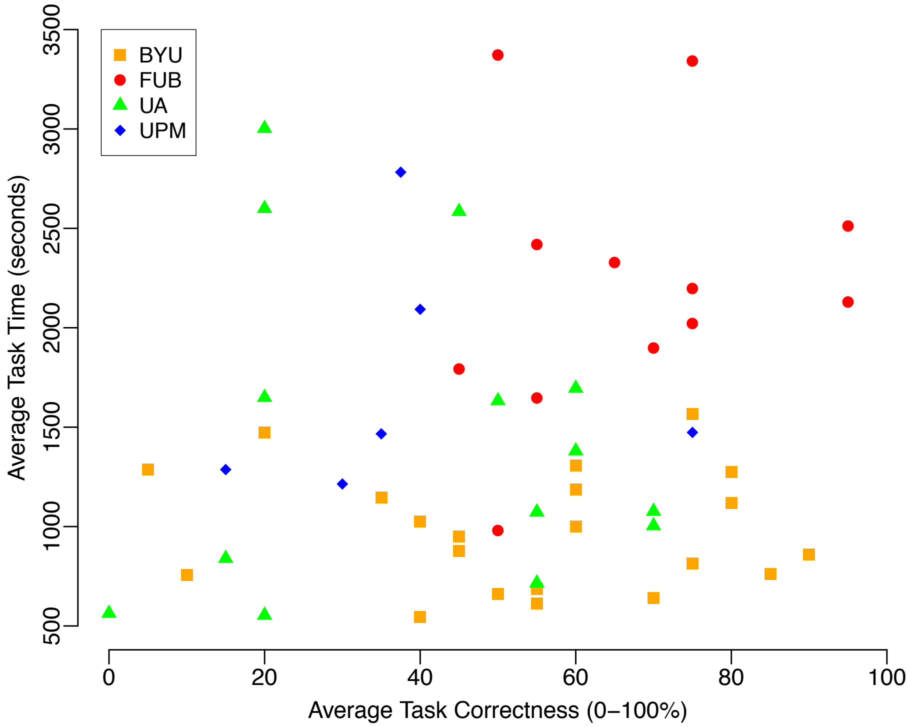
Conversely, E_orig's participants completed tasks on paper, and E_repl's participants were given a standardized environment with the programs already set up as development projects. Moreover, "[a]ll [E_repl] subjects performed an initial, familiarization task in order to try out the programming environment and the user interface" [212, p. 188]. Thus, environment setup was not a significant factor for either E_orig or E_repl.

In general, program *order* is not directly relevant to design patterns, nor to industrial software development. To eliminate it in future studies, we recommend administering a pre-task, as was done in E_repl. In the case of E_joint, we statistically correct for the effect by including *order* as a covariate in all models.

Q.4 Perceived Time Limits

Notice in Figure Q.2 that the BYU participants appear constrained on the time they were willing or able to spend on the experiment, relative to the participants at the other three sites. Also, the FUB participants took longer (on average) than the participants at any other site—in particular, more than twice as long as the BYU participants (see Table Q.4). These trends could be due to variations in the participants' perception of time requirements.

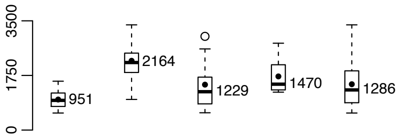
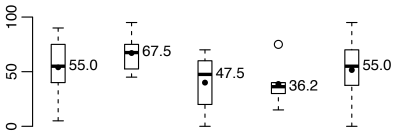
Figure Q.2: Participants plotted by average task time and correctness, categorized by site. Repeat of Figure O.2 from Appendix O. Included here for convenience.



The BYU participants, though volunteers, were primarily motivated by course credit. As students, they were required to complete 7 hours per week of software engineering work. The students were permitted to count any time spent on the experiment toward their weekly 7 hours. However, a former teaching assistant indicated that students typically struggle to fill the 7 hours. In fact, the class from which the participants were solicited reported an average of only 6.6 hours per week. If BYU participants perceived a time constraint, it was not likely due to the course requirements.

Another possible explanation is that the BYU researchers unintentionally communicated a time limit to their participants. One BYU participant (31563) clearly mentioned a perceived time limit when describing the difficult aspects of the CO tasks: "Figuring out what all is going on in the alloted time left, since I spent all of my 2 hours download and installing Eclipse" (sic). Incidentally, this participant's data were excluded from analysis as unusable, as described in Appendix E.

Table Q.4: Average task time and correctness for each participant (same data as that shown in Figure Q.2). The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

<table>
<tr><th></th><th>BYU
(21)</th><th>FUB
(12)</th><th>UA
(14)</th><th>UPM
(6)</th><th>All
(53)</th><th><em>sig.</em></th></tr>
<tr><td>Average
Task Time
(seconds)</td><td colspan="5">
</td><td><b>&lt;0.001</b></td></tr>
<tr><td>Average
Task
Correctness
(0–100%)</td><td colspan="5">
</td><td><b>0.018</b></td></tr>
</table>

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)

At BYU, the participants were told that the experiment would require 2–3 hours of their time. This information was given as an estimate to help participants plan for the experiment. The estimate was provided verbally at the time of solicitation and in print on the BYU-specific instruction sheet for volunteers (a copy of which is included in the lab package). The verbal instructions were not recorded, but the instruction sheet stated, “Required 2–3 Hours.” In hindsight, the term “Required” was a poor choice, since it can be ambiguously understood as implying a time limit. Also, a better estimate may have been 2–4 hours.

Concerning FUB’s high times, Lutz Prechelt recalls:

> The FUB subjects were true volunteers. Martin Liesenberg produced a number of mishaps while he tried to get the portal to run and had to put off the subjects I think twice. Some of them disappeared. So the remaining ones were likely seriously interested in participating and not just doing minimal course duty. Also, I think they felt they were helping a fellow student (whose bachelor thesis involved needing participants for the experiment). Whether Martin ever uttered any specific expectations for the time required I cannot say. I found no email of

> his with such content, but he may not have copied me on that or may have said something orally. (email, Oct. 9, 2012)

Thus the FUB participants may have perceived a minimum time requirement, which may have led to their taking more time on average. However, based on the data, we can at least conclude that a minimum time limit was not prominently stressed. Thus it seems unlikely that all of the FUB participants would have perceived such a requirement.

Intrinsic motivation is the more likely explanation for FUB's high times, especially considering that the BYU and UA participants—who took the least time on the experiment (see Table Q.4)—were primarily extrinsically motivated (by course credit). Moreover, the UPM participants, whose times fall in the middle, did not receive course credit, but were solicited from the UPM research lab. Having prior relationships with their experimenters (who were their graduate advisors), they likely felt greater social pressure to do a good job than did the participants at BYU and UA who did not previously know their experimenters. Also, we find no evidence that the UPM participants were particularly intrinsically motivated, thus explaining why their times were not as high as those at FUB.

In general, the perception of time limits is a relevant variable in industry settings. For instance, if a developer feels short on time, s/he may avoid examining the overall program structure, and therefore miss some of the benefits of a given design pattern. However, our data are insufficient to test time perception as a moderator. We recommend that future studies more carefully control and report on this variable.

Q.5 Cultural (or Regional) Variation

During data analysis, we found several cases in which cultural (or regional) variation may have impacted the results. First, we discovered that the question for CO task 2 is slightly ambiguous, such that it resulted in two different interpretations. The question stated:

> For a given object *`CommChannel channel`*, the statement *`channel.reset()`* may produce the result *`CommChannel.IMPOSSIBLE`*. Under which condition does this result occur?

The intended meaning is that the first sentence provides a context in which the actual question (second sentence) is to be answered. The anticipated answer is thus: `CLOSED` channel or `FAILED` encrypted channel. However, the question can also be read such that the first sentence is merely an example, and "this result" refers to *all* cases of `IMPOSSIBLE` that can occur:

- `reset()` called on a `CLOSED` channel,
- `reset()` called on a `FAILED` encrypted channel,
- `close()` called on a `CLOSED` channel,
- `basicOpen()` called on a non-`CLOSED` channel.

Notice that the second interpretation is a superset of the first. Nine of the twelve FUB participants clearly provided the extra information, whereas none of the participants from any other site did so.[3] In Lutz Prechelt's words, "If it is indeed so that none of the non-FUB participants used this second interpretation, I would consider this a fascinating example of a subtle cultural dependency" (email, Aug. 25, 2011).[4]

A second example of a possible cultural dependency concerns self-reporting on the pre-questionnaires. As discussed in Appendix C, all of the extreme outliers for the lines of code questions came from an American school (BYU or UA), and most of those came from BYU. The BYU participants also reported substantially higher numbers of languages used. In this latter case, the participants listed the actual languages by name, so inflating the count

[3] One participant's response from BYU (64084) could, possibly, be construed as having followed the second interpretation, but the answer is so incomplete that it could just as well have been an incorrect response to the first interpretation.

[4] The second interpretation was not penalized in the grading. The extra information was simply ignored. Additionally, it may have taken longer to answer the question according to the second interpretation, which would create additional variance. However, we find no evidence that the choice of interpretation interacted with *variant*. Thus the final results should not be biased.

was not likely due to disinterest with the question. However, BYU participants may not have actually used more languages. Their threshold for counting a language may simply have been lower.

We discuss additional cultural variables in the following subsections, some of which are clearly relevant to industrial contexts. However, they may or may not interact with design patterns. We recommend future studies to report such variables as much as reasonably possible. More work is needed to identify cultural variables, as well as to formulate methods for controlling them, both within and across studies.

Q.6 IDE Preferences

Another cultural or regionally-influenced variable concerns programming environments.[5] On the source code download pages we provided the following instructions:

> Download the .zip file and import it into your workspace. In Eclipse this is done via: File - Import - General - Import existing project into workspace. Now select 'select archive file' and click 'Finish'.

The IDE instructions were meant to aid inexperienced participants. We did not require the use of Eclipse, nor was the code specific to any particular programming environment. However, in response to these instructions, 4 of the 21 BYU participants mentioned problems with and/or complained about having to use the Eclipse IDE. Several UPM participants also complained about Eclipse, as though it were a requirement, and half of the UPM participants complained about the difficulty of importing the CO and GR programs into the NetBeans IDE, as though the code was configured specifically for Eclipse (which it was not). Conversely, no participants at FUB or UA expressed these concerns or mentioned NetBeans.

Apparently, our advice impacted the sites unevenly. The imbalance could be due to a regional language effect, in which only participants at BYU and UPM mistakenly thought

[5] For a related discussion, see Section 4.4 of E_repl's published report [212, pp. 163–164].

Eclipse was required. However, since BYU and UPM do not share native languages (English vs. Spanish), and neither do FUB and UA (German vs. English), the more likely explanation is regional preference for programming environments. Quite possibly, FUB and UA participants already preferred Eclipse, so any misperceived requirement to use Eclipse was simply not viewed as a problem.

In general, we believe participants should be encouraged to work in their native environments. First, standardizing the environment for a specific experiment does not solve the problem of generalizability, since the programming environment may still vary across studies. Second, when experiments involve small-scale tasks, the time spent learning an unfamiliar environment could mask the main effect. Third, allowing participants to use their preferred environments increases the realism of the experiment.

However, if we allow participants to work in their preferred environments, we must either make no mention of development tools, or we must clearly instruct the participants to use the tools to which they are most accustomed. Also, providing project import instructions is helpful, but if given these instructions must include all relevant IDEs and development frameworks. Otherwise, only some participants are benefited, and as we find in E_joint, that bias is not likely to be random with respect to sites or studies.

Q.7 Language Barriers

This variable is also related to the issue of cultural variation. In fact, the first example of cultural variation discussed above (concerning two interpretations for CO task 2) appears to have been directly related to language issues.

In E_joint, we administered all instruments in English.[6] However, two participants mentioned problems understanding the English text. UPM participant 38048 commented, “It was quite more easy for me to perform all the task if the instructions was in spanish” (sic).

[6] E_orig administered German text to German participants, and concerning E_repl, Marek Vokáč comments, “The task descriptions the subjects got were in Norwegian to lessen the language friction” (email, Oct. 14, 2012).

Similarly, FUB participant 71173 commented, “[T]he task descriptions in English created some problems for me.” Quite possibly, other participants were also hindered by the English instructions, but simply did not notice the effect or viewed it as not worth mentioning. Concerning FUB, Lutz Prechelt comments:

> I expect the English text will have slowed down the FUB participants somewhat (and some more than others, creating some additional variance), but should not have distorted the PAT/ALT effect. Further, the FUB participants all know the pattern terminology mostly in its English form, rather than German. (email, Oct. 9, 2012)

Another example of language barriers impacting experimental measurements involves the student status question. The choices provided for the student status question were (in order): undergraduate student, graduate student, postgraduate student, not a student. In hindsight, *postdoctoral* may have been a better term instead of *postgraduate* since, in some countries, the designation *postgraduate* is synonymous with and preferred over that of *graduate*. Interestingly, the UPM participants, who were all master’s students, uniformly selected *postgraduate*. Conversely, all of the master’s and PhD students at the other three sites—including those at FUB, a German university—selected *graduate*.

Thus language had an impact on measurement in E_joint, and to some degree, that impact was contingent on *site*. We do not anticipate that language should directly influence the effect of patterns, but it may at least add sufficient variance to mask the effect in an experiment or to muddle results across studies.

Specific to replication, language poses a particular challenge because we often need to maintain instruments across sites and studies (especially in the early stages of investigation). On the one hand, ensuring that various translations convey the same meaning is difficult. On the other hand, to administer experiments in a single language introduces the potential for misinterpretation by non-native readers. At this point, it is not clear to us which approach is best.

Table Q.5: Prevalence of participants at each site who complained that the task instructions were difficult to understand.

Site	Complaints	Total Participants	Percentage
BYU	10	21	48%
FUB	0	12	0%
UA	3	14	21%
UPM	4	6	67%
Total	17	53	32%

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid

Q.8 Clarity of Task Instructions

This variable relates to the issue of language barriers (discussed previously). In post-questionnaire comments, 17 of 53 participants (almost 1/3) expressed difficulty understanding the task instructions (see Table Q.5). The prevalence of these complaints is worth examining because the same tasks were used by E_orig and E_repl, but in neither of those studies did the participants complain to such a degree.

Perhaps the problem is due to our use of student participants, as opposed to the professionals of the prior two studies. In this case, it may be that students have more trouble due to lack of experience, and they manifest that trouble by complaining about the instructions. Alternatively, students may simply complain more than professionals when faced with similar frustrations. Or, perhaps the professionals also complained, but that information has since been lost.

According to Lutz Prechelt, principal investigator for E_orig, the third hypothesis is very unlikely, and indeed, we find no evidence to support that hypothesis in either the published reports or the datasets. More importantly, all three hypotheses are inconsistent with the fact that *all of E_joint's participants were students, and yet the complaints were imbalanced across sites.* As a percentage, the complaints were much more prevalent at BYU and UPM than at FUB and UA (see Table Q.5). Further, the complaints do not correspond with student status—BYU and FUB participants were mostly undergraduates, whereas UA and UPM participants were entirely graduate students (as shown in Table B.1 in Appendix B). As discussed in Appendix C, the participants were also fairly homogeneous with respect to developer experience. Concerning pattern knowledge, the FUB and UA participants do report broader overall exposure than the participants at BYU and UPM. However, the UA participants also report greater pattern knowledge than those at FUB, which does not seem consistent with the results in Table Q.5.

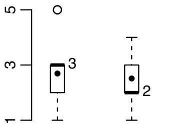
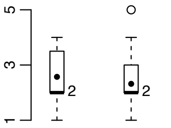
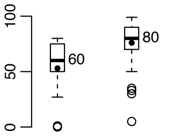
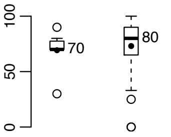
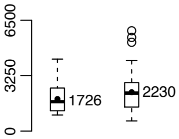
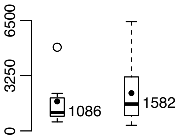
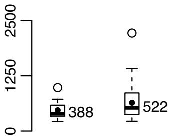
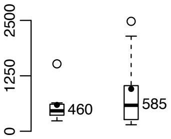
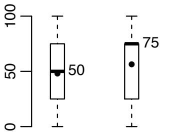
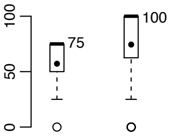
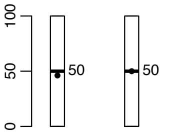
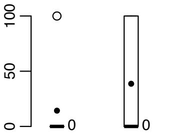
Another possibility is that the translation (from German to English) muddled the task instructions. We find some evidence for this hypothesis in that FUB provided the translation, and accordingly, none of the FUB participants complained. However, the complaints also do not appear to have been entirely contingent on native language, since the prevalence of complaints differed between the two English-speaking schools, BYU and UA. On the other hand, UA does report a significant portion of their participants as being international students (8/18; see Table B.1 in Appendix B).

Table Q.6 indicates that the participants who complained performed marginally different from those that did not complain. On average, they worked a little more quickly and scored a bit lower. They also showed less confidence in their answers and viewed the tasks as slightly more difficult. Given the imbalance in complaints across sites, as well as the data in Table Q.6, the task descriptions likely account for at least a small portion of the cross-site variance—which means they likely inhibit isolation of the main effect. However, since the task descriptions were the same for ALT and PAT, they should not have biased the conclusions.

Q.9 Compilation/Testing Expectations

E_orig was administered on paper, such that compilation was not a concern. Accordingly, some of the function bodies were replaced by the comment, “Body doesn’t matter.” When the code was translated to Java for E_joint, these stubs were maintained. Thus some of the Java methods contained only a comment, with no return statement, causing the code to not

Table Q.6: Comparison of participants—those who complained that the task instructions were difficult to understand versus those who did not. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. For a description of each variable, identified by field number, see Appendix F.

	Communication Channels (CO) C (13) ¬C (40)	Graphics Library (GR) C (7) ¬C (46)
Participant Assessment of Task Difficulty (scale 1–5)* (fields 43, 52)		
Participant Confidence in His/Her Answers (0–100%) (fields 44, 53)		
Task 1 Time (seconds) (fields 66, 75)		
Task 2 Time† (seconds) (fields 69, 78)		
Task 1 Correctness (0–100%) (fields 81, 89)		
Task 2 Correctness (0–100%) (fields 87, 91)		

C = Complained
¬C = Did not complain

*1=low difficulty, 5=high.
†Two outliers for GR program not shown: ¬C=7239 and ¬C=5842.

compile.[7] On one hand, maintaining the stubs without modification minimizes differences between the replication and the original study. However, the fact that the code did not compile appears to have caused problems for some participants—e.g., participant 74027

[7] E_joint's grading rubric included compilation as one of the criteria. However, that particular criterion was applied only to the code the participants wrote and/or modified. This was accomplished by extracting the modified portions of code and placing them in a framework for testing. Thus the participants were not penalized for non-compiling code that they did not write.

commented, “it didn’t compile with my version of java, which caused problems because my ide did not work properly.”

Additionally, the code did not support full execution by default for either the CO or GR programs. The CO program simply contained no main method by which to execute the code. For GR, a “testrun” class was provided, but it worked for only one of the two initial output modes of the program. Several participants complained about not being able to execute the code. For example, in response to the statement, “I found these to be the most difficult aspects of the task,” participant 26851 responded, “Not being able to run something to weed out stupid mistakes.”

The problem is that many developers write software by testing as they go. When the code is not compilable by default, participants either have to spend time fixing it—which some participants did, although it was not the assigned task—or they have to interact with the task in an abnormal way. Further, as the graders explained, “Sometimes the participants can compile, sometimes they can’t. Sometimes they can partially execute, sometimes they can’t. Sends mixed messages, mixed expectations.” Thus the problem was not just that the participants could not compile by default, but more particularly, we were unclear with them concerning our compilation and testing expectations.

Compilation and testing expectations can certainly be unclear in the real world. However, given that development practices can vary regionally (e.g., in response to where one was educated), compilation and testing issues likely affect sites unevenly, which means they induce extraneous variance unrelated to the question of interest. Therefore, in future PatMain studies we recommend eliminating compilation errors (as was done in E_repl[8]) even though doing so means modifying the original source code. It may also be worthwhile to clarify testing expectations and/or to systematically explore the use of testing to see if it influences the outcome of the PatMain experiment.

[8] Of E_repl, Marek Vokáč states, “The C++ code was as in Prechelt’s original, the only changes were those needed to make it compile, plus introduction of a small library to give a console user interface. This library had only a few functions and played no significant role in the code” (email, Oct. 16, 2012).

Appendix R

Detailed Program/Task Descriptions[1]

The text in this section is quoted from E_orig's published report [168, pp. 1140–1142], with minor modifications to adapt it to the present context. The text is also similar to that given in E_repl's published report [212, pp. 173–176], which mostly paraphrases the original.

The program metrics listed below (LOC and class counts) vary across the three studies. This is due to: 1) E_repl adjusted the programs from the original paper-based format to facilitate compilation, and 2) in E_joint, the programs were translated from C++ to Java. The metrics given in the text are those for E_orig; metrics for E_repl and E_joint are provided in footnotes.

The programs used by E_orig and E_repl included C++ header files, in which system components were declared. Since the system components were not technically part of the programs, the header files were excluded from the program metrics. Some testing code was also provided for the GR program, but that code was not provided in the form of a class. However, for E_joint—which administered the experiment in Java—both the header files and the testing code had to be translated into classes. To make E_joint's metrics comparable to those of the prior studies, we ignore the system classes entirely and consider the GR testing class only for LOC counts (not for class counts).

[1] Cited in Chapter 3.

R.1 Decorator: Communication Channels (CO)

Communication Channels is a wrapper library. A communication channel establishes a connection for transparently transferring arbitrary-length packets of data. One can turn on additional logging, data compression, and encryption functionality. The library does not implement the functionality itself, but only provides a facade to a system library. However, this application of the Facade pattern is irrelevant to the experiment.

The PAT variant, which comprises 365 LOC in six C++ classes,[2] is designed with a Decorator for adding functionality to a bare channel. Logging, data compression, and encryption are implemented as decorator classes. The ALT variant, which comprises 318 LOC in a single C++ class,[3] uses flags and if-sequences for turning functionality on or off; the flags can be set when creating a channel. Communication Channels is the only program where the ALT variant has a structured (as opposed to object-oriented) design.

R.1.1 CO—Work Task 1

"*Enhance the functionality of the program such that error-correcting encoding (bit redundancy) can be added to communication channels.*" The error-correcting functionality is already provided by a system class, so the participants only had to integrate that functionality into the program as a new wrapper. The PAT participants had to add a new Decorator class, while the ALT participants had to make additions and changes at various points in the existing program.

We expect two influences of the Decorator on the participants' behavior. First, the ALT variant is easier to understand because its behavior is not delocalized as in the multiple decorator classes. This would lead to the conclusion that the ALT groups should be faster than the PAT groups, especially for participants with low pattern knowledge. Second, a counter-influence results from the structure of the Decorator: The functionality is encapsulated

[2] [CO-PAT] E_repl = 404 LOC, 6 C++ classes; E_joint = 311 LOC, 6 Java classes (including 1 Java interface).

[3] [CO-ALT] E_repl = 342 LOC, 1 C++ class; E_joint = 267 LOC, 1 Java class.

in classes and one need hardly care about mutual influences. In particular, in the ALT variant, the participants have to ensure they add the new functionality at the correct places in the program for proper sequencing of the various switchable functionalities; this will consume time and may lead to mistakes. We expect the second influence to be stronger than the first and, hence, the PAT variant to be preferable, especially at higher levels of pattern knowledge.

R.1.2 CO—Work Task 2

A communication channel has different states (namely, `opened`, `closed`, or `failed`) and its operations have different result codes (`OK`, `failure`, or `impossible`). Work task 2 called to "*determine under which conditions a reset() call will return the 'impossible' result.*" To do this the participants had to find the spots where the states were changed. In the PAT variant, these spots are spread over the different decorator classes. Program understanding is gained in the first working task, so only the new details relevant for this task need to be understood now. We expect this task will be easier for the more localized ALT variant with respect to both work time and correctness.

The participants were also asked to "*create a channel object that performs compression and encryption.*" The ALT participants had to create only a single object (one statement), giving parameters for the functionality flags, while PAT participants had to determine the proper nesting of the decorators to get the required functionality in the requested order. We expect the PAT groups will take longer and commit more errors.

R.2 Composite and Abstract Factory: Graphics Library (GR)

Graphics Library contains a library for creating, manipulating, and drawing simple types of graphical objects (lines and circles) on different types of output devices (alphanumeric display, pixel display). In a central class (generator), the output device is selected. Depending on the device, the corresponding types of graphical objects are created. Some basic objects (lines and points) are implemented identically for all devices. However, for complex objects, like circles, implementation depends on the specific device. Furthermore, graphical objects can be collected into groups, that can be manipulated like individual objects.

The patterns used in the PAT variant of this program are Abstract Factory (for the generator classes) and Composite (for hierarchical object grouping). The ALT variant of the program uses switch statements, implemented in a single generator class, to select and instantiate the appropriate classes for each output device. The ALT variant also uses a quasi-Composite to implement object grouping. The only difference is that groups are not treated as objects themselves, as in the Composite. For instance, a group B is included in another group A by adding each element of B individually to A—i.e., there is no hierarchical group nesting.

The Graphics Library program has a smaller structural difference between its PAT and ALT variants than does the Communication Channels program. The PAT variant comprises 682 LOC in 13 C++ classes;[4] the ALT variant comprises 663 LOC in 11 C++ classes.[5]

R.2.1 GR—Work Task 1

"*Add a third type of output device (plotter).*" Participants maintaining the PAT variant had to introduce a new concrete factory class and extend the factory selector method. Participants in the ALT groups had to enhance the switch statements in all methods of the generator class. Both groups had to add two concrete product classes.

The time for finding the changes is expected to be almost equal for the PAT and ALT groups. Thus, we anticipate the main difference in time required for this task to be caused by program understanding. We expect the simpler ALT variant to be easier to understand, at least for participants with low pattern knowledge. Additionally, we expect that pattern knowledge will help both groups due to the Composite structure in both variants. However,

[4] [GR-PAT] E_repl = 683 LOC, 13 C++ classes; E_joint = 578 LOC, 13 Java classes (including 2 Java interfaces).

[5] [GR-ALT] E_repl = 667 LOC, 11 C++ classes; E_repl = 598 LOC, 11 Java classes (including 2 Java interfaces).

the PAT participants may profit a little more from pattern knowledge since they also interact with the Abstract Factory pattern.

R.2.2 GR—Work Task 2

Determine whether a specific sequence of operations will result in an x-shaped figure. This work task is a small comprehension test concerning the Composite structure. The key to the answer for both groups is finding out that only references to graphical objects (not copies of objects) are stored in an object group. The structure of both variants is quite similar in the region of interest, so we do not expect to observe significant differences between the ALT and the PAT groups. However, we do expect pattern knowledge to have an impact—participants with low pattern knowledge are expected to be slower than those with high pattern knowledge because the latter will be more familiar with the Composite pattern.

Appendix S

Comparison of Statistical Methods[1]

In this section, we compare statistical methods across the three PatMain studies. In particular, we show that the methods are sufficiently similar such that we can directly compare the numerical results. This discussion is particularly relevant to Table 3.7 in Section 3.6.3.

S.1 E_orig

To analyze the *time* response variable, E_orig used *analysis of variance* (ANOVA) and *nonparametric* (distribution-free) *bootstrap* methods. For *correctness*, E_orig simply compared the counts of solutions with errors. As Prechelt et al. explain, "For many tasks, all groups achieved near-perfect correctness" [168, p. 1136].

ANOVA was used for preliminary analysis to determine which variables best explain *time*. However, ANOVA is subject to normality assumptions, and the *time* data were skewed. Thus the authors avoided ANOVA for the final results. The authors also avoided Kruskal-Wallis and Wilcoxon (both of which are based on rank) because they wanted their results to summarize means, rather than medians. Bootstrap methods were a reasonable nonparametric alternative.

Bootstrapping is a resampling technique [64]. In E_orig, the authors randomly resampled their data 10,000 times with replacement (making sure to preserve group cardinalities). To compare two groups, they calculated the mean difference between the groups within each

[1] Cited in Chapter 3.

resampling. This process resulted in a distribution of 10,000 difference estimates for each test statistic. It was from these distributions that the authors essentially "read off" their p-values.

S.2 E_repl

E_repl's analysis was more complex than E_orig's. The authors modeled both *time* and *correctness*, as well as addressed an additional statistical concern besides non-normality—non-independence of the data. This latter concern existed as well with E_orig, but was not accounted for in that analysis.

Non-independence occurs when subsets of the data correlate in response to some other variable. In the case of PatMain, the experiment protocol requires each participant to complete multiple tasks—thus observations cluster around individual participants. Ideally, participant effects should be factored out to reduce variance. Statistical procedures that account for these blocking (i.e., grouping) variables are more precise, in that they can accurately reduce error estimates. Lower error terms mean lower p-values and tighter confidence intervals. Consequently, E_orig's analysis was not incorrect, simply less efficient. Where statistical significance was not obtained, accounting for non-independence may have yielded a significant result; otherwise, the results would not have changed much.

To account for non-normality and non-independence, E_repl used *generalized estimating equations* (GEE), which is a specialized form of *generalized linear models* (GLM). First, note that GLM is a generalization of linear regression, and by extension, of ANOVA. Consequently, GLM and GEE are both related to some of the methods used in E_orig. The benefit of GLM is that it allows the researcher to specify non-Gaussian distributional assumptions, thereby obviating the need for nonparametric methods. GEE further adds to GLM support for modeling blocking variables.

Using GEE, the authors modeled *time* with a gamma distribution and *correctness* with a normal distribution. The gamma distribution is ideal for skewed data, for which the range is strictly greater than zero. The authors also applied a log transformation to both *time* and *correctness*. Justifications for the transformations were not given in the paper. However, log transformations are typically used to normalize skewed data.

S.3 E_joint

For E_joint, we used both frequentist and Bayesian statistics. However, we do not discuss the Bayesian methods here because neither E_orig, nor E_repl performed a comparable analysis. Concerning the frequentist methods, we used *linear mixed models*, which are an extension of ANOVA and linear regression. These methods add support for modeling blocking variables and fitting non-Gaussian distributions. In these two respects, mixed models are similar to GEE.

Simply including a variable in a model does not achieve the same effect as blocking on that variable. For standard variables the data are assumed to be independent, whereas for blocking variables, the analysis specifically estimates and accounts for sub-correlations. In mixed models analysis, blocking variables are modeled as *random effects*. Accordingly, we model participant ID as a random effect in E_joint.

We also address the concern of non-normality for both *time* and *correctness*. For *time*, we apply a log transformation (like the analysis of E_repl), which effectively normalizes the observations. Accordingly, a normal distribution is an appropriate model for our data. Further, the range of *correctness* is discretized into only five buckets. As a result, it cannot take on much of a skew. Therefore, unlike E_repl, we do not apply a log transformation to *correctness*. That said, with only five buckets, the *correctness* variable is unlikely to fit a smooth normal distribution. Thus, although we assume normality for the frequentist analysis, we avoid that assumption entirely in the Bayesian analysis by modeling *correctness* with a beta distribution.

In summary, our methods are most similar to those of E_repl, particularly with respect to statistical efficiency. However, E_orig's methods are a bit more straightforward, which can be helpful in the early stages of investigation. Nevertheless, straightforward statistics come at a price. Although bootstrapping is unhampered by distributional assumptions, it sacrifices statistical power. Also, bootstrapping and ANOVA do not account for blocking variables. *Thus, we would expect little or no change in the results were E_repl and E_joint to swap statistical methods. However, using either GEE or mixed models in E_orig's analysis may increase statistical significance in some cases.*[2]

[2] In fact, as part of their study, E_repl reanalyzed E_orig's data using GEE methods. In general, the p-values either did not change much or they decreased, as expected.

Appendix T

Additional Results Data[1]

In Table 3.7 of Section 3.6.3, we provide a comparison of results across the three PatMain studies. For that comparison, we define concrete hypotheses based on the original hypothesis statements. Given the variables involved, twelve potential concrete hypotheses can be made for each task. For two of the tasks (CO and GR task 2), the original hypothesis statements do not address all of the possible combinations for concrete hypotheses. In this section, we provide results for the remaining combinations, which may be helpful for future meta-analysis (see Table T.1).

Table T.1: Comparison of results across the three PatMain studies. This table is a supplement to Table 3.7 in Section 3.6.3 and follows the same format. This table shows results for the remaining combinations of concrete hypotheses not addressed by the original hypothesis statements.

Hypothesis Statement	Concrete Hypotheses	Baseline & Expectation		E_orig	Reanalysis of E_orig	E_repl	E_joint
CO Task 2, Comprehension	$t : H\ ?\ L$	L	?	+23%	-	-	−36% (.090)*
	$t : PH\ ?\ PL$	PL	?	+6%	−5% (>.05)	−35% (>.05)	−61% (.007)
The PAT groups will take longer and commit more errors.	$t : AH\ ?\ AL$	AL	?	+34%	+40% (>.05)	+28% (>.05)	−5% (.866)
	$c : H\ ?\ L$	L	?	-	-	-	+39 pp (.049)
	$c : PH\ ?\ PL$	PL	?	-	+14 pp (>.05)	+1 pp (>.05)	INS
	$c : AH\ ?\ AL$	AL	?	-	−9 pp (>.05)	+15 pp (<.05)	
GR Task 2, Comprehension	$c : H\ ?\ L$	L	?	-	-	-	NS
	$c : PH\ ?\ PL$	PL	?	-	−10 pp (>.05)	−18 pp (>.05)	INS
ALT and PAT will not significantly differ; the task will require less time at higher levels of pattern knowledge for both variants.	$c : AH\ ?\ AL$	AL	?	-	−5 pp (>.05)	+15 pp (>.05)	

*This value is taken from the filtered CO_time model, as defined in Section 3.5, but with all interactions dropped.
? = hypothesis/expectation is undefined.

[1] Cited in Chapter 3.

Appendix U

Threats to Validity Continued[1]

This material is a continuation of Section 3.7. In this section, we list minor threats to validity.

U.1 Construct Validity

The possibility exists that self-reporting on the pre-questionnaires was inaccurate, biased, or somehow inconsistent across sites. For example, cultural trends may influence what a person considers to be "professional" experience. In fact, we found evidence that the BYU participants tended to exaggerate when reporting LOC written much more so than the FUB and UPM participants. For a discussion of this issue, see Appendix C. To mitigate inaccuracies, we collected several related metrics for developer experience and pattern knowledge. We then pruned away the least promising metrics and aggregated the rest. For an example with developer experience, see Appendix I.

The experiment was translated from German into English. Unfortunately, several grammar errors occurred which were not corrected prior to the first site (FUB) administering the experiment. For the sake of consistency, we did not correct the text at the other three sites. Some participants noticed the errors, but were not bothered by them. We also pre-tested the framework at BYU on several students not affiliated with the study. The test participants had no problem understanding the correct meanings. Thus, we do not think the errors impacted the results of the study.

[1] Cited in Chapter 3.

E_joint's programs were translated from C++ into Java. To minimize changes, we maintained the original structures as much as possible. However, this decision meant that the Java versions were written in a subtle C style. Possibly the style may have confused some participants. One participant (24085) did comment, "Those are not at all valid java codes. No experience programmer will write such code. That made understanding the code tough" (sic). In the worst case, the coding style could have slowed down some participants, creating additional variance, but we do not expect it to have influenced the main effect.

All participants took the experiment in Java, but undoubtedly, not all participants had equal familiarity with Java (as would be the case for any language). If Java familiarity varied significantly within or across sites it may have added considerable noise to the results. To investigate this possibility, we developed a metric to measure Java familiarity. However, when added to the statistical models, we found the metric to have almost no impact on the results (as described in Appendix I). Additionally, 1) almost all of the participants (47 of 53) listed java as a language they use often; 2) at two of the sites (FUB and UPM) the participants all voluntarily chose Java (over C++ and C#); and 3) based on course curricula, we know that almost all of the BYU and UA participants had recently received formal training in Java. Thus, we do not believe that Java familiarity had a significant impact on the results of the experiment.

Lastly, we observed a small learning (or maturation) effect in E_joint. However, we found the effect to be unrelated to the use of design patterns and were able to correct for it in the statistical analysis.

U.2 Conclusion Validity

Concerning statistical assumptions, the one assumption we could not fully ensure for our models is that of homoscedasticity. In the case of the *patKnow* explanatory variable, our sample is thin in the upper range, such that we could not confirm constant variance, although the variance may indeed be constant. As a result, p-values for E_joint, which concern the high range of *patKnow*, may be overestimated—i.e., the p-values may be biased toward type 2 errors or failure to reject the null hypothesis. For further details on model assumptions, see Appendix J.

To be clear, the results of the Bayesian analysis are unlikely to be biased by the selection of prior distributions. First, we enlisted a third party helper (who was not previously affiliated with the study) to estimate the priors using historical data from E_orig. Second, we intentionally chose broad priors so as to make them of little influence on the results.

Additionally (concerning the Bayesian analysis), in some cases only a few observations were available for estimating a particular parameter. In these cases, the minimal data do not mislead the models. The posterior distributions simply do not deviate much from the broad priors; thus the resulting probabilities are insignificant (i.e., near 0.5), as they should be. However, additional data could produce significance in these cases. Thus, the Bayesian analysis, by virtue of its broad priors, is biased toward type 2 errors, or failure to reject the null hypothesis. However, this bias is preferable because the Bayesian analysis is post-hoc, and post-hoc analyses are predisposed to type 1 errors.

U.3 Internal Validity

A potential threat to internal validity is survivor bias. Survivor bias could occur in a software engineering experiment as follows: Condition A is in fact worse than B. It is even so much worse that most of the low-performers in the A group drop out of the experiment prematurely due to frustration. The rest of the A group thus appears much stronger than it should and the inferiority of A versus B disappears.

Of the 61 participants to begin the experiment, 6 quit prematurely. For a list of the 6, see Appendix E. In general, we find no evidence that quitting in E_joint correlates with any particular experiment group or site. Thus, quitting was most likely due to general disinterest with the experiment, or possibly due to frustration at some condition unrelated to design patterns.

U.4 External Validity

Our use of a web portal may have impacted the participants' attitudes. Possibly the work-at-home nature of the portal led to reduced motivations or focus. In contrast, the participants in the prior two studies were required to work in dedicated rooms, which may have promoted a greater sense of seriousness about the experiment. Ultimately, both approaches have positives and negatives, inasmuch as they represent tradeoffs in experimental stress. Some developers work better under pressure, other do not. Thus, either setting could be viewed as more or less like industry. To an extent, having both represented in the final results strengthens external validity.

Appendix V

Researcher Interactions[1]

In this section, we describe interactions between replicating researchers and prior experimenters. We include this information per Carver's guidelines for reporting experimental replications [40]. This information is important for assessing shared bias between experiments.

V.1 E_repl

The published report from E_repl does not specifically mention interactions with E_orig's experimenters. However, the replication clearly reuses E_orig's artifacts, and the report states that Walter Tichy taught the patterns course for both E_orig and E_repl. Further, we know from email archives that Vokáč, Sjøberg, Tichy, Unger, and Prechelt were actively discussing the replication in 2002. Their discussions included whether to let participants use personal laptops, whether to incorporate pair programming, and whether to conduct a qualitative analysis. Lutz Prechelt further recalls that those interactions began as early as 1999, and in his words, "There was quite a bit of interaction overall."

V.2 E_joint

E_joint involved four types of interaction:

1. *Interactions between E_joint organizers and prior experimenters during replication design.* During this phase we did not interact with E_repl's researchers. However, Lutz Prechelt—principal investigator for E_orig—designed E_joint's protocol.

[1] Cited in Chapter 3.

2. *Interactions between E_joint organizers and the individual research teams.* To facilitate this interaction, we created a project website [75] and mailing list. Quite a few discussions happened on the mailing list, to which all research teams were privy. Those discussions concerned operation of the web portal, as well as clarifications about information posted on the project website.

3. *Interactions between the individual research teams and prior experimenters.* The research teams did not directly interact with prior experimenters, except for mailing list discussions involving Lutz Prechelt. Lutz answered many questions about the experiment and web portal.

4. *Interactions between E_joint organizers and prior experimenters during joint analysis.* Again, our interactions with E_orig were entirely through Lutz Prechelt, who participated throughout the analysis process. During analysis, Jonathan Krein also communicated with Marek Vokáč—E_repl's principal investigator. Since this communication occurred late in the analysis process, the only procedure impacted was data filtering (i.e., Section 3.6.1). We were already considering filtering, but Marek's feedback reinforced that intuition. Note that all other mentions of Marek in the paper reflect data we incorporated after the fact.

V.3 Summary

We see strong indications of shared bias (which is not necessarily a bad thing) between E_orig and each of the two PatMain replications (E_repl and E_joint). However, the level of interaction appears to be more significant between E_orig and E_joint, since E_orig's principal investigator, Lutz Prechelt, was also a primary designer and collaborator for E_joint.

Appendix W

Future Work[1]

Most of the ideas below focus on exploring variables that moderate the effect of design patterns. Ideally we can discover a handful of key variables sufficient to predict the outcome of pattern experiments in most contexts. Understanding the role of key moderators can also lead to the development of explanatory and predictive theories, which in turn may enable the formulation of transferable best practices.

W.1 PatMain Replications

The most obvious item for future work is to further replicate the PatMain experiment with additional controls and measurements for moderators. We recommend that the next round of testing should focus on a few of the most promising variables. Controlling additional variables can help to reduce extraneous variance, but ideally we want to identify the smallest set of moderators possible, sufficient to predict experimental outcomes. We recommend the following moderators:

- *pattern knowledge*: Pattern knowledge has been shown in all three PatMain studies to moderate the effect of design patterns. However, the extent of its influence, particularly with respect to the Abstract Factory pattern in the GR program, is still not fully worked out.

[1] Cited in Chapter 3. This appendix—which provides detailed information on future work specific to Chapter 3—is not to be confused with Chapter 6, which describes general ideas for future work relative to the overall dissertation.

- *developer experience*: Developer experience has also been considered in the prior PatMain studies, but its effect as a moderator has only been tested in E_joint; it needs to be further validated.
- *motivation*: Motivation is promising because it correlates with variance in E_joint, as well as with variance across the three PatMain studies. However, we were not able to directly test its interaction with design patterns in E_joint due to lack of the appropriate quantitative data.

Note that when investigating moderators, researchers should pay close attention to the level of heterogeneity in their samples. Insufficient heterogeneity can result in failure to detect a moderator, even in cases where the moderator is highly influential.

In addition to testing specific moderators, we also recommend that future replicators document other variables that appear to correlate with variance in their studies. Doing so will make those studies more useful in the future, especially if the variables identified above prove inadequate. Also, it may be worthwhile to explore relationships between moderators. For instance, can developer experience compensate for a lack of pattern knowledge (and vice versa)? Lastly, controlling variables within studies will not, by itself, solve the problem of generalizability. We also need to develop methods for mapping moderators across studies. This work could take the form of investigating best methods for assessing particular developer attributes and formulating standardized assessments.

W.2 PatMain Meta-analysis

Our analysis in this paper could be improved by statistically modeling all three PatMain studies together. A combined analysis is reasonable because the two replications both sought to closely duplicate the original. Thus, all three studies use nearly the same materials and comparable experimental designs. The resulting models would likely be similar in size to those we have already constructed, but the volume of data would nearly triple. For any attempt at such an analysis, note the following concerns:

1. E_joint used a survey to assess pattern knowledge instead of a training course. One approach to resolving this protocol difference would be to simply label all E_joint observations as “PRE” (meaning pre-training). Additionally, both of the prior PatMain studies collected demographic data on pattern knowledge. Those data could be used to create a unified pattern knowledge metric.

2. Three of E_joint’s sites initially graded their own participants’ solutions. However, for the joint analysis, we re-graded all solutions to ensure consistency. Correlating the centrally-graded scores with those of the individual sites revealed only marginal correspondence. For one task (at FUB) the correlation was perfect, but most correlations were in the range 0.25–0.75, and one was only 0.13.[2] Thus, care should be taken when comparing correctness scores between studies.

3. CO tasks 2 and 3 were combined in E_repl and E_joint, but not in E_orig. The combination involved simply adding times and averaging correctness scores [212, p. 179].

4. In E_repl, Vokáč et al. initially applied time corrections to adjust for participants who spent long periods resolving a technical nuance (e.g., finding a missing closing brace) [212, pp. 163–164]. However, they found the corrections to have little impact on the results and subsequently eliminated them from the analysis. When incorporating data from E_repl, time corrections should be ignored.

5. E_orig, E_repl, and one site from E_joint (UA) tested participants on the ST and BO programs. These programs could also be included in the combined analysis using the Bayesian methods shown in this paper, which allow for missing data.

6. Several variables, which could not be statistically modeled using E_joint data alone, could be investigated in the combined analysis. These variables include: student vs. professional status, paid vs. unpaid compensation, voluntary participation vs. participation by assignment, C++ vs. Java, and computer-based format vs. paper-based format.

[2] Pearson product-moment correlation coefficients.

W.3 Historical and Case Study Investigations of Moderators

In addition to further replicating the PatMain experiment, it would be interesting to review the design pattern literature for data on potential moderators. Many studies likely contain at least some traces of information on moderators, the synthesis of which may reveal significant insights. The results of such a literature review could be used to corroborate E_joint's findings, or even to generalize and extend them.

Another alternative would be industry case studies. It may be possible to find software projects involving maintenance tasks, wherein some parts of the software have been constructed with patterns and some parts without. We could then assess whether various moderators appear salient in practice. Such studies would only be observational, but in connection with experiments, they could help establish external validity. If enough is known about the identity of the developers, open source repositories may prove useful in this regard.

W.4 Taxonomy of Interfering Variables

In this paper, we have considered many different types of interfering variables. Some are inherent to the problem domain, whereas others are artifacts of the experimental setup. Some probably only influence overall variance, whereas others directly moderate the main effect. Some also overlap (e.g., site and culture), and many can be considered of one type or another, depending on one's frame of reference. As such, it may be beneficial to develop a structured understanding of interfering variables—e.g., to develop a taxonomy of variable types, along with methods for identifying and resolving each of the types. A literature review of interfering variables in software engineering experiments may be helpful in this regard. Also, it may be helpful to review how other disciplines deal with such variables.

W.5 Design Pattern Properties

According to Vokáč et al., "each design pattern. . . has its own nature, so that it is not valid to characterize design patterns as useful or harmful in general" [212, p. 191]. If this conclusion is true, as our results suggest, then presumably some set of design pattern properties must exist (e.g., complexity), which if understood, could be used to better predict a pattern's impact on software maintenance. Via the PatMain studies, we are in the process of directly testing several patterns. However, if we could understand the properties on which the usefulness of patterns depend, we could potentially predict outcomes for new and untested patterns.

For example, we find that the threshold of experience required for Abstract Factory to be beneficial during maintenance is greater than that required for Decorator. Similarly, the Visitor pattern (which was not tested in E_joint) was found by E_repl to be especially problematic, more so than both Decorator and Abstract Factory. Possibly pattern complexity explains these findings. If so, complexity is an example of a property that could be used to further generalize experimental findings.

W.6 Studies of Motivation

In our study, we found strong evidence to suggest that motivation affects the variance of developers. Based on our findings, we would expect intrinsically motivated developers to manifest less variance than extrinsically motivated developers. However, it is not clear whether these findings translate to industry. If they do, then a better understanding of developer motivations could enable greater control over the consistency (and therefore predictability) of software development outcomes.

Any study of motivation would need to develop (or borrow from other fields) a theoretical framework for differentiating types of motivation. Psychology or the social sciences may be a good place to start. Concerns include: whether an intrinsic/extrinsic distinction is the most effective characterization of motivation for the context of software engineering, as well as whether secondary motivations are as important as primary motivations in predicting developer performance.

Appendix X

Frequentist Statistical Results[1]

For a description of the frequentist models, see Section 3.5. Tables X.1–X.16 present results based on the full dataset (53 participants). Tables X.17–X.34 present results for the same models, but after applying participant filtering (as described in Section 3.6.1). Statistical source code (SAS 9.3) and output are included in the lab package. All p-values are two-sided. Variables not appearing in the results tables have been removed via model tuning due to lack of significance. For a description of the tuning process, see Appendix K.

X.1 Results Layout

Column headers are defined for all tables as follows:

- `Effect`: Explanatory variable.
- `Level`: Level for a categorical explanatory variable.
- `Level Diff`: Two categorical levels compared, the first minus the second.
- `F Value`: f-statistic.
- `Pr>F`: Two-sided p-value, computed using the f distribution.
- `Estimate`: Parameter estimate.
- `Orig Scale`: The parameter estimate converted back to the original scale (*time* models only).
- `t Value`: t-statistic.

[1] Cited in Chapter 3.

- `Pr>|t|`: Two-sided p-value, computed using the t distribution.
- `Adj P`: Two-sided p-value, adjusted to account for multiple comparisons (Tukey-Kramer). Has no effect for binary categorical variables.
- `Ratio`: As explained below, back-transforming difference estimates yields ratios.

X.2 Results Interpretation

Since we log-transformed the *time* variable, we must back-transform the results. When *time_ln* is the response variable, back-transformation requires computing e^x, where x is the log-scale estimate. In these cases, we must back-transform four types of estimates: slope estimates (e.g., Table X.2), marginal means (e.g., Table X.3), differences between marginal means (e.g., Table X.4), and differences between interaction levels (e.g., Table X.22).

When the log-scale estimate, x, represents a difference, i.e., $x = y - z$, back-transformation yields a ratio rather than an interval, as in $e^x = e^{y-z} = e^y/e^z$. Thus, the linear (or additive) effect on the log scale becomes a multiplicative effect on the original scale. In decimal form the back-transformed differences are essentially multiplicative factors that scale the response variable up or down by some percentage depending on whether the value is greater or less than one.

Note that interval differences can be computed on the original scale if y and z are known, as in $e^y - e^z$ (e.g., Table X.4). However, if y and z are not marginal means, then their estimates depend on an arbitrary selection of values for all other variables in the model. In this case, shifting other variables also shifts y and z. On the log scale, such shifts are linear, so differences remain constant, but on the original scale shifts translate into multiplicative changes. Consequently, interval differences are only meaningful on the original scale when computed from marginal means.

Slope is a measure of change in one variable in response to change in another variable and can be represented as $\Delta response/\Delta covariate$. For the slope estimates shown in the tables, $\Delta covariate = 1$. Thus, the log-scale slope estimates represent differences (i.e.,

$\Delta response/\Delta covariate = \Delta response/1 = \Delta response$), such that back-transformation yields ratios:

$$e^{\Delta response} = e^{y-z} = e^{y}/e^{z}$$

Thus, on the *log* scale, a slope estimate represents the *linear* change in *time_ln* expected to occur in response to a 1-unit *linear* change in the associated covariate; but on the *non-log* scale, a slope estimate represents the *multiplicative* change in *time* expected to occur in response to a 1-unit *linear* change in the covariate.

Further, since $\Delta covariate = 1$ for the slopes shown in the tables, back-transformation via e^{x} yields estimates relative to 1-unit changes in the covariates. To obtain estimates relative to other values for $\Delta covariate$, simply back-transform the log-scale estimate using the more general formula e^{xd}, where d is the desired $\Delta covariate$. For example, in the first footnote (*) of Table X.2, $d = 1$, such that back-transformation is computed as $e^{-0.1091*1}$; but in the second footnote (†), $d = 10$, such that back-transformation is computed as $e^{0.0022*10}$.

When *time_ln* is a covariate instead of the response variable (which occurs for slope estimates in the *correctness* models; e.g., Table X.6), interpretation is handled differently. Since *time_ln* is the covariate, interpretation of slope estimates requires back-transforming $\Delta covariate$ by computing $e^{\Delta covariate}$. As mentioned above, $\Delta covariate = 1$ by default and back-transformation of differences yields ratios.

Thus, on the original scale, the slope estimates shown in the tables represent the *linear* change in *correctness* expected to occur in response to a *multiplicative* increase in non-log time of e^{1} (≈ 2.7). To obtain a slope estimate relative to a multiplicative factor other than e^{1}, simply compute $ln(\theta)x$, where x is the slope estimate given in the table and θ is the desired multiplicative factor. For example, Table X.6 indicates that a 1-unit increase in *time_ln* yields an average *correctness* increase of about 7.17 percentage points. Alternatively, on the original scale, an approximately 2.7-fold increase in *time* yields an average *correctness* increase of about 7.17 percentage points. Or stated more intuitively, a 2-fold increase in *time* yields an average correctness increase of about $ln(2)7.17 = 4.97$ percentage points.

Table X.1: CO_time, unfiltered (52 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
site	8.46	<**0.001**
order	8.03	**0.007**
task	270.93	<**0.001**
devExp	3.28	0.076
correctness	3.40	0.071
variant	0.01	0.925

Table X.2: CO_time, unfiltered (52 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	6.8147	21.79	<**0.001**
site	BYU	−0.5129	−3.10	**0.003**
site	FUB	0.0029	0.02	0.988
site	UA	−0.5069	−2.83	**0.007**
site	UPM	0	-	-
order	1	0.2732	2.83	**0.007**
order	2	0	-	-
task	1	1.2887	16.46	<**0.001**
task	2	0	-	-
devExp	-	−0.1091*	−1.81	0.076
correctness	-	0.0022†	1.84	0.071
variant	ALT	−0.0094	−0.09	0.925
variant	PAT	0	-	-

* A 1-unit increase in developer experience yields an average time decrease of about 10.3%.

† A 10-point increase in correctness yields an average time increase of about 2.2%.

Table X.3: CO_time, unfiltered (52 participants). Marginal means (least squares estimates).

Effect	Level	Estimate	Orig Scale*
site	BYU	6.7265	834
site	FUB	7.2423	1397
site	UA	6.7324	839
site	UPM	7.2394	1393
order	1	7.1217	1239
order	2	6.8485	942
task	1	7.6295	2058
task	2	6.3408	567
variant	ALT	6.9805	1075
variant	PAT	6.9898	1086

* Computed as e^x, where x is the log-scale estimate.

Table X.4: CO_time, unfiltered (52 participants). Differences for marginal means.

					Orig
Effect	Level Diff	Estimate	t Value	Adj P*	Scale†
site	BYU−FUB	−0.5158	−4.01	**0.001**	−563
site	BYU−UA	−0.0060	−0.05	1.000	−5
site	BYU−UPM	−0.5129	−3.10	**0.016**	−559
site	FUB−UA	0.5098	3.52	**0.005**	558
site	FUB−UPM	0.0029	0.02	1.000	4
site	UA−UPM	−0.5069	−2.83	**0.033**	−554
order	1−2	0.2732	2.83	**0.007**	296
task	1−2	1.2887	16.46	<**0.001**	1491
variant	ALT−PAT	−0.0094	−0.09	0.925	−10

* Adjusted for multiple comparisons (Tukey-Kramer).

† Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table X.3 above).

Table X.5: CO_correctness, unfiltered (52 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
site	5.20	**0.003**
devExp	2.78	0.101
time_ln	2.64	0.110
variant	2.90	0.095

Table X.6: CO_correctness, unfiltered (52 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−70.60	−1.73	0.090
site	BYU	37.31	2.94	**0.005**
site	FUB	51.72	3.82	<**0.001**
site	UA	29.92	2.17	**0.035**
site	UPM	0	-	-
devExp	-	7.41*	1.67	0.101
time_ln	-	7.17†	1.63	0.110
variant	ALT	12.17	1.70	0.095
variant	PAT	0	-	-

* A 1-unit increase in developer experience yields an average correctness increase of about 7.4 percentage points.

† A 2x increase in work time yields an average correctness increase of about 5.0 percentage points.

Table X.7: CO_correctness, unfiltered (52 participants). Marginal means (least squares estimates).

Effect	Level	Estimate
site	BYU	53.93
site	FUB	68.35
site	UA	46.55
site	UPM	16.62
variant	ALT	52.45
variant	PAT	40.28

Table X.8: CO_correctness, unfiltered (52 participants). Differences for marginal means.

Effect	Level Diff	Estimate	t Value	Adj P*
site	BYU−FUB	−14.42	−1.52	0.436
site	BYU−UA	7.39	0.80	0.855
site	BYU−UPM	37.31	2.94	**0.025**
site	FUB−UA	21.80	2.09	0.169
site	FUB−UPM	51.72	3.82	**0.002**
site	UA−UPM	29.92	2.17	0.147
variant	ALT−PAT	12.17	1.70	0.095

* Adjusted for multiple comparisons (Tukey-Kramer).

Table X.9: GR_time, unfiltered (51 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
site	3.62	**0.019**
order	10.91	**0.002**
task	34.40	<**0.001**
devExp	20.00	<**0.001**
correctness	12.94	<**0.001**
variant	6.18	**0.016**

Table X.10: GR_time, unfiltered (51 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	7.8042	18.06	<**0.001**
site	BYU	−0.4337	−2.00	0.051
site	FUB	0.0846	0.36	0.722
site	UA	−0.0794	−0.34	0.736
site	UPM	0	-	-
order	1	0.4348	3.30	**0.002**
order	2	0	-	-
task	1	0.8358	5.87	<**0.001**
task	2	0	-	-
devExp	-	−0.3573*	−4.47	<**0.001**
correctness	-	0.0057†	3.60	<**0.001**
variant	ALT	−0.3343	−2.49	**0.016**
variant	PAT	0	-	-

* A 1-unit increase in developer experience yields an average time decrease of about 30.0%.

† A 10-point increase in correctness yields an average time increase of about 5.9%.

Table X.11: GR_time, unfiltered (51 participants). Marginal means (least squares estimates).

Effect	Level	Estimate	Orig Scale*
site	BYU	6.6131	745
site	FUB	7.1314	1251
site	UA	6.9674	1061
site	UPM	7.0468	1149
order	1	7.1571	1283
order	2	6.7223	831
task	1	7.3576	1568
task	2	6.5218	680
variant	ALT	6.7725	873
variant	PAT	7.1069	1220

* Computed as e^x, where x is the log-scale estimate.

Table X.12: GR_time, unfiltered (51 participants). Differences for marginal means.

					Orig
Effect	Level Diff	Estimate	t Value	Adj P*	Scale†
site	BYU−FUB	−0.5183	−2.97	**0.023**	−506
site	BYU−UA	−0.3543	−2.05	0.184	−317
site	BYU−UPM	−0.4337	−2.00	0.201	−404
site	FUB−UA	0.1640	0.85	0.832	189
site	FUB−UPM	0.0846	0.36	0.984	101
site	UA−UPM	−0.0794	−0.34	0.986	−88
order	1−2	0.4348	3.30	**0.002**	452
task	1−2	0.8358	5.87	<**0.001**	888
variant	ALT−PAT	−0.3343	−2.49	**0.016**	−347

* Adjusted for multiple comparisons (Tukey-Kramer).

† Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table X.11 above).

Table X.13: GR_correctness, unfiltered (51 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
task	4.89	**0.032**
time_ln	9.55	**0.003**
variant	1.00	0.322

Table X.14: GR_correctness, unfiltered (51 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−65.67	−2.02	**0.049**
task	1	20.80	2.21	**0.032**
task	2	0	-	-
time_ln	-	15.29*	3.09	**0.003**
variant	ALT	8.28	1.00	0.322
variant	PAT	0	-	-

* A 2x increase in work time yields an average correctness increase of about 10.6 percentage points.

Table X.15: GR_correctness, unfiltered (51 participants). Marginal means (least squares estimates).

Effect	Level	Estimate
task	1	64.29
task	2	43.49
variant	ALT	58.03
variant	PAT	49.75

Table X.16: GR_correctness, unfiltered (51 participants). Differences for marginal means.

Effect	Level Diff	Estimate	t Value	Pr>|t|
task	1−2	20.80	2.21	**0.032**
variant	ALT−PAT	8.28	1.00	0.322

Table X.17: CO_time, filtered (42 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
site	13.53	**<0.001**
order	9.36	**0.004**
task	218.26	**<0.001**
patKnow	3.96	0.053*
correctness	2.76	0.104
variant	5.95	**0.019***
patKnow × variant	5.12	**0.029**

*Results for patKnow and variant are not meaningful outside the interaction. See Tables X.19 and X.22 instead.

Table X.18: CO_time, filtered (42 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	7.1296	23.28	**<0.001**
site	BYU	−0.5887	−3.41	**0.002**
site	FUB	0.0833	0.45	0.657
site	UA	−0.3234	−1.57	0.125
site	UPM	0	-	-
order	1	0.2819	3.06	**0.004**
order	2	0	-	-
task	1	1.2974	14.77	**<0.001**
task	2	0	-	-
patKnow	-	−0.2527†	−2.86	**0.007**
correctness	-	0.0023*	1.66	0.104
variant	ALT	−0.8617†	−2.44	**0.019**
variant	PAT	0†	-	-
patKnow × variant	ALT	0.2388†	2.26	**0.029**
patKnow × variant	PAT	0†	-	-

*A 10-point increase in correctness yields an average time increase of about 2.3%.

†Results for patKnow and variant are not meaningful outside the interaction. See Tables X.19 and X.22 instead.

Table X.19: CO_time, filtered (42 participants). Slopes for *patKnow* × *variant* (from Table X.18).

	Level				
Effect	(variant)	Estimate	t Value	Pr>|t|	Ratio
patKnow	ALT	−0.0139	−0.17	0.866	0.986*
patKnow	PAT	−0.2527	−2.86	**0.007**	0.777†

*For ALT tasks, a 1-unit increase in pattern knowledge yields an average time decrease of about 1.4%.

†For PAT tasks, a 1-unit increase in pattern knowledge yields an average time decrease of about 22.3%.

Table X.20: CO_time, filtered (42 participants). Marginal means (least squares estimates).

Effect	Level	Estimate	Orig Scale*
site	BYU	6.6045	738
site	FUB	7.2764	1446
site	UA	6.8698	963
site	UPM	7.1932	1330
order	1	7.1269	1245
order	2	6.8450	939
task	1	7.6347	2069
task	2	6.3373	565

*Computed as e^x, where x is the log-scale estimate.

Table X.21: CO_time, filtered (42 participants). Differences for marginal means.

					Orig
Effect	Level Diff	Estimate	t Value	Adj P*	Scale†
site	BYU−FUB	−0.6720	−5.85	**<0.001**	−707
site	BYU−UA	−0.2653	−1.73	0.321	−224
site	BYU−UPM	−0.5887	−3.41	**0.008**	−592
site	FUB−UA	0.4067	2.80	**0.038**	483
site	FUB−UPM	0.0833	0.45	0.970	115
site	UA−UPM	−0.3234	−1.57	0.409	−368
order	1−2	0.2819	3.06	**0.004**	306
task	1−2	1.2974	14.77	**<0.001**	1503

*Adjusted for multiple comparisons (Tukey-Kramer).

†Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table X.20 above).

Table X.22: CO_time, filtered (42 participants). Differences for *patKnow* × *variant*.

	Level Diff				
PatKnow*	(variant)	Estimate	t Value	Pr>|t|	Ratio†
1.7 (min)	ALT−PAT	−0.4544	−2.45	**0.019**	0.635
3.3 (mean)	ALT−PAT	−0.0826	−0.89	0.381	0.921
5.4 (max)	ALT−PAT	0.4304	1.73	0.091	1.538

*Values shown are rounded to fit the table.

†Computed as e^x, where x is the log-scale estimate. Since each estimate (x) represents a difference ($y−z$), back-transformation yields a ratio ($e^x = e^{y-z} = e^y/e^z$). E.g., when *patKnow* is at its minimum value, the ALT/PAT ratio is 0.635, meaning ALT tasks require 36.5% less time than PAT tasks, on average.

Table X.23: CO_correctness, filtered (42 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
site	4.53	**0.008**
patKnow	4.11	**0.049**
variant	0.41	0.523

Table X.24: CO_correctness, filtered (42 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−11.46	−0.65	0.520
site	BYU	40.70	3.16	**0.003**
site	FUB	43.79	3.13	**0.003**
site	UA	24.30	1.49	0.144
site	UPM	0	-	-
patKnow	-	10.57*	2.03	**0.049**
variant	ALT	4.76	0.64	0.523
variant	PAT	0	-	-

* A 1-unit increase in pattern knowledge yields an average correctness increase of about 10.6 percentage points.

Table X.25: CO_correctness, filtered (42 participants). Marginal means (least squares estimates).

Effect	Level	Estimate
site	BYU	66.12
site	FUB	69.21
site	UA	49.72
site	UPM	25.42
variant	ALT	55.00
variant	PAT	50.24

Table X.26: CO_correctness, filtered (42 participants). Differences for marginal means.

Effect	Level Diff	Estimate	t Value	Adj P*
site	BYU−FUB	−3.09	−0.34	0.986
site	BYU−UA	16.40	1.37	0.524
site	BYU−UPM	40.70	3.16	**0.015**
site	FUB−UA	19.49	1.72	0.324
site	FUB−UPM	43.79	3.13	**0.016**
site	UA−UPM	24.30	1.49	0.453
variant	ALT−PAT	4.76	0.64	0.523

* Adjusted for multiple comparisons (Tukey-Kramer).

Table X.27: GR_time, filtered (42 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
site	8.97	<**0.001**
order	4.60	**0.038**
task	22.63	<**0.001**
patKnow	12.30	**0.001**
correctness	23.30	<**0.001**
variant	5.44	**0.025**

Table X.28: GR_time, filtered (42 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	7.0909	19.08	<**0.001**
site	BYU	−0.3862	−1.65	0.107
site	FUB	0.4631	1.75	0.087
site	UA	0.5660	1.84	0.073
site	UPM	0	-	-
order	1	0.3030	2.14	**0.038**
order	2	0	-	-
task	1	0.7334	4.76	<**0.001**
task	2	0	-	-
patKnow	-	−0.3569*	−3.51	**0.001**
correctness	-	0.0086†	4.83	<**0.001**
variant	ALT	−0.3409	−2.33	**0.025**
variant	PAT	0	-	-

* A 1-unit increase in pattern knowledge yields an average time decrease of about 30.0%.
† A 10-point increase in correctness yields an average time increase of about 9.0%.

Table X.29: GR_time, filtered (42 participants). Marginal means (least squares estimates).

Effect	Level	Estimate	Orig Scale*
site	BYU	6.4258	618
site	FUB	7.2751	1444
site	UA	7.3780	1600
site	UPM	6.8120	909
order	1	7.1242	1242
order	2	6.8212	917
task	1	7.3394	1540
task	2	6.6060	740
variant	ALT	6.8023	900
variant	PAT	7.1432	1265

* Computed as e^x, where x is the log-scale estimate.

Table X.30: GR_time, filtered (42 participants). Differences for marginal means.

					Orig
Effect	Level Diff	Estimate	t Value	Adj P*	Scale†
site	BYU−FUB	−0.8493	−4.64	<**0.001**	−826
site	BYU−UA	−0.9522	−4.07	**0.001**	−983
site	BYU−UPM	−0.3862	−1.65	0.364	−291
site	FUB−UA	−0.1029	−0.46	0.968	−156
site	FUB−UPM	0.4631	1.75	0.311	535
site	UA−UPM	0.5660	1.84	0.270	692
order	1−2	0.3030	2.14	**0.038**	325
task	1−2	0.7334	4.76	<**0.001**	800
variant	ALT−PAT	−0.3409	−2.33	**0.025**	−366

* Adjusted for multiple comparisons (Tukey-Kramer).
† Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table X.29 above).

Table X.31: GR_correctness, filtered (42 participants). Type 3 tests of fixed effects.

Effect	F Value	Pr>F
time_ln	34.87	<**0.001**
variant	1.39	0.245

Table X.32: GR_correctness, filtered (42 participants). Solution for fixed effects.

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−117.67	−3.88	<**0.001**
time_ln	-	25.40*	5.91	<**0.001**
variant	ALT	9.69	1.18	0.245
variant	PAT	0	-	-

* A 2x increase in work time yields an average correctness increase of about 17.6 percentage points.

Table X.33: GR_correctness, filtered (42 participants). Marginal means (least squares estimates).

Effect	Level	Estimate
variant	ALT	66.45
variant	PAT	56.76

Table X.34: GR_correctness, filtered (42 participants). Differences for marginal means.

Effect	Level Diff	Estimate	t Value	Pr>|t|
variant	ALT−PAT	9.69	1.18	0.245

Appendix Y

Bayesian Statistical Results[1]

For a description of the Bayesian models, see Section 3.5. Table Y.1 presents results based on the full dataset (53 participants). Table Y.2 presents results for the same models, but after applying participant filtering (as described in Section 3.6.1). Statistical source code (R 2.15.2) is included in the lab package.

Y.1 Results Layout

The Bayesian tables are abbreviated versions of a Microsoft Excel file, which is provided in the lab package (BayesianAnalysisResults.xlsx). The numbers at the right margin map the table rows to the Excel file. The Excel file adds additional data and visualizations. Coding and comprehension tasks are abbreviated in the tables as ‘t1’ and ‘t2’. All other abbreviations are as previously defined. All probabilities are rounded—i.e., none are exactly 1 or 0.

Results for the *time* and *correctness* models are represented as columns in the tables (labeled T1–T6 and C1–C6, respectively). Rows are grouped by bold subheadings, which identify two types of information. On the far left, the subheadings identify the variable or interaction under consideration (e.g., the results on rows 37–40 were computed from the *program* × *variant* interaction). At center and on the right, the subheadings identify the specific effect being analyzed—which effect corresponds to the variable listed on the left, or if an interaction is listed on the left, then it corresponds to a variable within the interaction. For example, on row 36, “compare: variant” means that ALT and PAT are being compared,

[1] Cited in Chapter 3.

and since the interaction *program* $\times$ *variant* is listed on the left, ALT and PAT are being compared separately for the CO and GR programs.

For each comparison, we provide two types of values: probabilities and differences. Probabilities are listed in black and are labeled $p(x > y)$, meaning the posterior probability that condition x either takes longer (models T1–T6) or scores higher (models C1–C6) than condition y. Differences are listed in gray and are labeled $x - y$, meaning the average difference in *time* or *correctness* between conditions x and y (computed as the difference between posterior distribution means).

Y.2 Results Interpretation

In addition to the "Results Interpretation" discussion in Section 3.5, note the following two concerns:

- Since the Bayesian analysis is based on binary variables, insignificant comparisons are those for which the probabilities are near 0.5. Thus, a probability of 0.25 is as significant as a probability of 0.75. Probabilities less than 0.5 simply indicate that the reverse comparison is more likely. The directionality of the comparisons shown in the tables (i.e., $x > y$ as opposed to $y > x$) is arbitrary. To reverse a comparison, simply compute $1 - p$ for probabilities and $-x$ for differences.
- Since *statistical power is influenced by both model size and by the distribution of observations over model parameters* [174, p. 347], probabilities should not be directly compared across models. Instead, cross-validation requires checking that two models support similar conclusions.

Table Y.1: Unfiltered Bayesian Results. See Appendix Y for a description of how to interpret this table.

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
order	**compare: order**						**compare: order**						11
p(1st > 2nd)	0.99	1.00	1.00	1.00	1.00	1.00	0.13	0.17	0.15	0.22	0.20	0.20	12
1st − 2nd	189	228	223	214	210	189	−3.4	−3.1	−3.5	−2.3	−2.6	−2.7	13
variant	**compare: variant**						**compare: variant**						33
p(PAT > ALT)	0.58	0.52	0.55	0.56	0.54	0.55	0.39	0.37	0.38	0.35	0.30	0.31	34
PAT − ALT	152	24	73	89	30	42	−5.1	−5.1	−4.3	−5.4	−5.7	−4.8	35
program × variant	**compare: variant**						**compare: variant**						36
p(CO PAT > CO ALT)	0.57	0.51	0.56	0.54	0.53	0.49	0.38	0.35	0.37	0.37	0.30	0.29	37
CO PAT − CO ALT	129	5	67	47	29	−13	−5.2	−5.6	−4.8	−5.2	−5.9	−5.4	38
p(GR PAT > GR ALT)	0.59	0.54	0.55	0.59	0.54	0.62	0.39	0.38	0.39	0.32	0.30	0.33	39
GR PAT − GR ALT	175	43	79	132	31	97	−5.0	−4.7	−3.7	−5.5	−5.4	−4.1	40
task × variant	**compare: variant**						**compare: variant**						41
p(t1 PAT > t1 ALT)	0.56	0.48	0.53	0.55	0.49	-	0.31	0.31	0.31	0.26	0.20	-	42
t1 PAT − t1 ALT	132	−17	26	86	−12	-	−9.2	−7.3	−7.1	−9.0	−8.6	-	43
p(t2 PAT > t2 ALT)	0.60	0.56	0.58	0.57	0.59	-	0.47	0.42	0.45	0.44	0.40	-	44
t2 PAT − t2 ALT	172	65	121	92	72	-	−1.0	−3.0	−1.5	−1.8	−2.7	-	45
program × task × variant	**compare: variant**						**compare: variant**						46
p(CO t1 PAT > CO t1 ALT)	0.56	0.52	0.61	0.61	0.57	-	0.33	0.34	0.30	0.32	0.23	-	47
CO t1 PAT − CO t1 ALT	111	33	129	126	62	-	−7.8	−5.9	−7.5	−8.0	−8.0	-	48
p(GR t1 PAT > GR t1 ALT)	0.56	0.44	0.44	0.49	0.41	-	0.28	0.29	0.32	0.19	0.17	-	49
GR t1 PAT − GR t1 ALT	154	−66	−78	46	−86	-	−10.5	−8.7	−6.7	−10.0	−9.3	-	50
p(CO t2 PAT > CO t2 ALT)	0.58	0.49	0.50	0.47	0.50	-	0.44	0.37	0.44	0.42	0.37	-	51
CO t2 PAT − CO t2 ALT	148	−23	5	−33	−4	-	−2.5	−5.2	−2.2	−2.5	−3.8	-	52
p(GR t2 PAT > GR t2 ALT)	0.62	0.64	0.66	0.68	0.67	-	0.50	0.47	0.47	0.46	0.43	-	53
GR t2 PAT − GR t2 ALT	196	153	237	218	149	-	0.5	−0.7	−0.8	−1.1	−1.5	-	54
time_or_correctness	**compare: correctness**						**compare: time**						61
p(Low > High)	0.01	0.03	0.09	0.32	0.05	0.10	0.15	0.16	0.18	0.37	0.19	0.12	62
Low − High	−232	−182	−135	−198	−155	−120	−4.4	−4.1	−4.0	−3.5	−3.6	−4.7	63
variant × patKnow	**compare: variant**						**compare: variant**						102
p(ALT Low > PAT Low)	-	0.46	-	-	-	-	-	0.68	-	-	-	-	103
ALT Low − PAT Low	-	−27	-	-	-	-	-	6.8	-	-	-	-	104
p(ALT High > PAT High)	-	0.49	-	-	-	-	-	0.59	-	-	-	-	105
ALT High − PAT High	-	−22	-	-	-	-	-	3.5	-	-	-	-	106
program × variant × patKnow	**compare: variant**						**compare: variant**						117
p(CO ALT Low > CO PAT Low)	-	0.44	-	-	-	-	-	0.62	-	-	-	-	118
CO ALT Low − CO PAT Low	-	−40	-	-	-	-	-	4.1	-	-	-	-	119
p(CO ALT High > CO PAT High)	-	0.54	-	-	-	-	-	0.67	-	-	-	-	120
CO ALT High − CO PAT High	-	31	-	-	-	-	-	7.0	-	-	-	-	121
													122
p(GR ALT Low > GR PAT Low)	-	0.48	-	-	-	-	-	0.74	-	-	-	-	123
GR ALT Low − GR PAT Low	-	−13	-	-	-	-	-	9.5	-	-	-	-	124
p(GR ALT High > GR PAT High)	-	0.44	-	-	-	-	-	0.50	-	-	-	-	125
GR ALT High − GR PAT High	-	−74	-	-	-	-	-	−0.1	-	-	-	-	126
task × variant × patKnow	**compare: variant**						**compare: variant**						137
p(t1 ALT Low > t1 PAT Low)	-	0.58	-	-	-	-	-	0.81	-	-	-	-	138
t1 ALT Low − t1 PAT Low	-	95	-	-	-	-	-	12.1	-	-	-	-	139
p(t1 ALT High > t1 PAT High)	-	0.45	-	-	-	-	-	0.56	-	-	-	-	140
t1 ALT High − t1 PAT High	-	−62	-	-	-	-	-	2.5	-	-	-	-	141
													142
p(t2 ALT Low > t2 PAT Low)	-	0.35	-	-	-	-	-	0.55	-	-	-	-	143
t2 ALT Low − t2 PAT Low	-	−148	-	-	-	-	-	1.5	-	-	-	-	144
p(t2 ALT High > t2 PAT High)	-	0.53	-	-	-	-	-	0.61	-	-	-	-	145
t2 ALT High − t2 PAT High	-	19	-	-	-	-	-	4.5	-	-	-	-	146

(continued on next page)

(Table Y.1 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
program × task × variant × patKnow	**compare: variant**						**compare: variant**						167
p(CO t1 ALT Low > CO t1 PAT Low)	-	0.53	-	-	-	-	-	0.70	-	-	-	-	168
CO t1 ALT Low − CO t1 PAT Low	-	41	-	-	-	-	-	6.8	-	-	-	-	169
p(CO t1 ALT High > CO t1 PAT High)	-	0.42	-	-	-	-	-	0.63	-	-	-	-	170
CO t1 ALT High − CO t1 PAT High	-	−107	-	-	-	-	-	5.0	-	-	-	-	171
													172
p(CO t2 ALT Low > CO t2 PAT Low)	-	0.36	-	-	-	-	-	0.54	-	-	-	-	173
CO t2 ALT Low − CO t2 PAT Low	-	−122	-	-	-	-	-	1.4	-	-	-	-	174
p(CO t2 ALT High > CO t2 PAT High)	-	0.67	-	-	-	-	-	0.72	-	-	-	-	175
CO t2 ALT High − CO t2 PAT High	-	169	-	-	-	-	-	9.1	-	-	-	-	176
													177
p(GR t1 ALT Low > GR t1 PAT Low)	-	0.62	-	-	-	-	-	0.92	-	-	-	-	178
GR t1 ALT Low − GR t1 PAT Low	-	149	-	-	-	-	-	17.3	-	-	-	-	179
p(GR t1 ALT High > GR t1 PAT High)	-	0.49	-	-	-	-	-	0.50	-	-	-	-	180
GR t1 ALT High − GR t1 PAT High	-	−17	-	-	-	-	-	0.1	-	-	-	-	181
													182
p(GR t2 ALT Low > GR t2 PAT Low)	-	0.34	-	-	-	-	-	0.56	-	-	-	-	183
GR t2 ALT Low − GR t2 PAT Low	-	−175	-	-	-	-	-	1.6	-	-	-	-	184
p(GR t2 ALT High > GR t2 PAT High)	-	0.39	-	-	-	-	-	0.49	-	-	-	-	185
GR t2 ALT High − GR t2 PAT High	-	−131	-	-	-	-	-	−0.2	-	-	-	-	186
variant × devExp	**compare: variant**						**compare: variant**						212
p(ALT Low > PAT Low)	-	-	0.41	-	-	-	-	-	0.70	-	-	-	213
ALT Low − PAT Low	-	-	−117	-	-	-	-	-	7.3	-	-	-	214
p(ALT High > PAT High)	-	-	0.48	-	-	-	-	-	0.53	-	-	-	215
ALT High − PAT High	-	-	−30	-	-	-	-	-	1.3	-	-	-	216
program × variant × devExp	**compare: variant**						**compare: variant**						227
p(CO ALT Low > CO PAT Low)	-	-	0.42	-	-	-	-	-	0.66	-	-	-	228
CO ALT Low − CO PAT Low	-	-	−92	-	-	-	-	-	5.9	-	-	-	229
p(CO ALT High > CO PAT High)	-	-	0.47	-	-	-	-	-	0.60	-	-	-	230
CO ALT High − CO PAT High	-	-	−42	-	-	-	-	-	3.7	-	-	-	231
													232
p(GR ALT Low > GR PAT Low)	-	-	0.40	-	-	-	-	-	0.75	-	-	-	233
GR ALT Low − GR PAT Low	-	-	−141	-	-	-	-	-	8.7	-	-	-	234
p(GR ALT High > GR PAT High)	-	-	0.49	-	-	-	-	-	0.47	-	-	-	235
GR ALT High − GR PAT High	-	-	−18	-	-	-	-	-	−1.2	-	-	-	236
task × variant × devExp	**compare: variant**						**compare: variant**						247
p(t1 ALT Low > t1 PAT Low)	-	-	0.53	-	-	-	-	-	0.81	-	-	-	248
t1 ALT Low − t1 PAT Low	-	-	47	-	-	-	-	-	11.3	-	-	-	249
p(t1 ALT High > t1 PAT High)	-	-	0.42	-	-	-	-	-	0.58	-	-	-	250
t1 ALT High − t1 PAT High	-	-	−98	-	-	-	-	-	2.9	-	-	-	251
													252
p(t2 ALT Low > t2 PAT Low)	-	-	0.30	-	-	-	-	-	0.60	-	-	-	253
t2 ALT Low − t2 PAT Low	-	-	−280	-	-	-	-	-	3.3	-	-	-	254
p(t2 ALT High > t2 PAT High)	-	-	0.53	-	-	-	-	-	0.49	-	-	-	255
t2 ALT High − t2 PAT High	-	-	38	-	-	-	-	-	−0.3	-	-	-	256
program × task × variant × devExp	**compare: variant**						**compare: variant**						277
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	0.34	-	-	-	-	-	0.76	-	-	-	278
CO t1 ALT Low − CO t1 PAT Low	-	-	−187	-	-	-	-	-	9.9	-	-	-	279
p(CO t1 ALT High > CO t1 PAT High)	-	-	0.44	-	-	-	-	-	0.64	-	-	-	280
CO t1 ALT High − CO t1 PAT High	-	-	−72	-	-	-	-	-	5.1	-	-	-	281
													282
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	0.50	-	-	-	-	-	0.56	-	-	-	283
CO t2 ALT Low − CO t2 PAT Low	-	-	2	-	-	-	-	-	1.9	-	-	-	284
p(CO t2 ALT High > CO t2 PAT High)	-	-	0.49	-	-	-	-	-	0.57	-	-	-	285
CO t2 ALT High − CO t2 PAT High	-	-	−11	-	-	-	-	-	2.4	-	-	-	286
													287
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	0.71	-	-	-	-	-	0.85	-	-	-	288

(continued on next page)

(Table Y.1 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
GR t1 ALT Low − GR t1 PAT Low	-	-	280	-	-	-	-	-	12.7	-	-	-	289
p(GR t1 ALT High > GR t1 PAT High)	-	-	0.40	-	-	-	-	-	0.52	-	-	-	290
GR t1 ALT High − GR t1 PAT High	-	-	−124	-	-	-	-	-	0.6	-	-	-	291
													292
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	0.10	-	-	-	-	-	0.65	-	-	-	293
GR t2 ALT Low − GR t2 PAT Low	-	-	−561	-	-	-	-	-	4.6	-	-	-	294
p(GR t2 ALT High > GR t2 PAT High)	-	-	0.58	-	-	-	-	-	0.41	-	-	-	295
GR t2 ALT High − GR t2 PAT High	-	-	88	-	-	-	-	-	−3.0	-	-	-	296
variant × time_or_correctness	**compare: variant**						**compare: variant**						322
p(ALT Low > PAT Low)	-	-	-	0.43	-	-	-	-	-	0.72	-	-	323
ALT Low − PAT Low	-	-	-	−96	-	-	-	-	-	7.6	-	-	324
p(ALT High > PAT High)	-	-	-	0.45	-	-	-	-	-	0.59	-	-	325
ALT High − PAT High	-	-	-	−82	-	-	-	-	-	3.1	-	-	326
program × variant × time_or_correctness	**compare: variant**						**compare: variant**						337
p(CO ALT Low > CO PAT Low)	-	-	-	0.50	-	-	-	-	-	0.74	-	-	338
CO ALT Low − CO PAT Low	-	-	-	26	-	-	-	-	-	9.8	-	-	339
p(CO ALT High > CO PAT High)	-	-	-	0.42	-	-	-	-	-	0.51	-	-	340
CO ALT High − CO PAT High	-	-	-	−119	-	-	-	-	-	0.7	-	-	341
													342
p(GR ALT Low > GR PAT Low)	-	-	-	0.35	-	-	-	-	-	0.69	-	-	343
GR ALT Low − GR PAT Low	-	-	-	−218	-	-	-	-	-	5.4	-	-	344
p(GR ALT High > GR PAT High)	-	-	-	0.48	-	-	-	-	-	0.66	-	-	345
GR ALT High − GR PAT High	-	-	-	−46	-	-	-	-	-	5.6	-	-	346
task × variant × time_or_correctness	**compare: variant**						**compare: variant**						357
p(t1 ALT Low > t1 PAT Low)	-	-	-	0.42	-	-	-	-	-	0.87	-	-	358
t1 ALT Low − t1 PAT Low	-	-	-	−144	-	-	-	-	-	13.5	-	-	359
p(t1 ALT High > t1 PAT High)	-	-	-	0.48	-	-	-	-	-	0.62	-	-	360
t1 ALT High − t1 PAT High	-	-	-	−28	-	-	-	-	-	4.5	-	-	361
													362
p(t2 ALT Low > t2 PAT Low)	-	-	-	0.43	-	-	-	-	-	0.57	-	-	363
t2 ALT Low − t2 PAT Low	-	-	-	−48	-	-	-	-	-	1.7	-	-	364
p(t2 ALT High > t2 PAT High)	-	-	-	0.42	-	-	-	-	-	0.56	-	-	365
t2 ALT High − t2 PAT High	-	-	-	−137	-	-	-	-	-	1.8	-	-	366
program × task × variant × time_or_correctness	**compare: variant**						**compare: variant**						387
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	-	0.61	-	-	-	-	-	0.93	-	-	388
CO t1 ALT Low − CO t1 PAT Low	-	-	-	120	-	-	-	-	-	18.0	-	-	389
p(CO t1 ALT High > CO t1 PAT High)	-	-	-	0.17	-	-	-	-	-	0.43	-	-	390
CO t1 ALT High − CO t1 PAT High	-	-	-	−373	-	-	-	-	-	−2.0	-	-	391
													392
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	-	0.40	-	-	-	-	-	0.56	-	-	393
CO t2 ALT Low − CO t2 PAT Low	-	-	-	−69	-	-	-	-	-	1.6	-	-	394
p(CO t2 ALT High > CO t2 PAT High)	-	-	-	0.67	-	-	-	-	-	0.60	-	-	395
CO t2 ALT High − CO t2 PAT High	-	-	-	135	-	-	-	-	-	3.3	-	-	396
													397
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	-	0.23	-	-	-	-	-	0.80	-	-	398
GR t1 ALT Low − GR t1 PAT Low	-	-	-	−409	-	-	-	-	-	9.1	-	-	399
p(GR t1 ALT High > GR t1 PAT High)	-	-	-	0.78	-	-	-	-	-	0.81	-	-	400
GR t1 ALT High − GR t1 PAT High	-	-	-	317	-	-	-	-	-	10.9	-	-	401
													402
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	-	0.47	-	-	-	-	-	0.58	-	-	403
GR t2 ALT Low − GR t2 PAT Low	-	-	-	−28	-	-	-	-	-	1.8	-	-	404
p(GR t2 ALT High > GR t2 PAT High)	-	-	-	0.17	-	-	-	-	-	0.51	-	-	405
GR t2 ALT High − GR t2 PAT High	-	-	-	−408	-	-	-	-	-	0.3	-	-	406
variant × site	**compare: variant**						**compare: variant**						538
p(ALT BYU > PAT BYU)	0.45	-	-	-	-	-	0.71	-	-	-	-	-	539

(continued on next page)

(Table Y.1 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
ALT BYU − PAT BYU	−66	-	-	-	-	-	8.2	-	-	-	-	-	540
p(ALT FUB > PAT FUB)	0.44	-	-	-	-	-	0.55	-	-	-	-	-	541
ALT FUB − PAT FUB	−145	-	-	-	-	-	1.9	-	-	-	-	-	542
p(ALT UA > PAT UA)	0.45	-	-	-	-	-	0.61	-	-	-	-	-	543
ALT UA − PAT UA	−87	-	-	-	-	-	5.3	-	-	-	-	-	544
p(ALT UPM > PAT UPM)	0.34	-	-	-	-	-	0.58	-	-	-	-	-	545
ALT UPM − PAT UPM	−311	-	-	-	-	-	5.0	-	-	-	-	-	546
program × variant × site	**compare:**	**variant**					**compare:**	**variant**					599
p(CO ALT BYU > CO PAT BYU)	0.39	-	-	-	-	-	0.77	-	-	-	-	-	600
CO ALT BYU − CO PAT BYU	−152	-	-	-	-	-	10.9	-	-	-	-	-	601
p(CO ALT FUB > CO PAT FUB)	0.54	-	-	-	-	-	0.53	-	-	-	-	-	602
CO ALT FUB − CO PAT FUB	74	-	-	-	-	-	1.5	-	-	-	-	-	603
p(CO ALT UA > CO PAT UA)	0.56	-	-	-	-	-	0.60	-	-	-	-	-	604
CO ALT UA − CO PAT UA	80	-	-	-	-	-	4.5	-	-	-	-	-	605
p(CO ALT UPM > CO PAT UPM)	0.23	-	-	-	-	-	0.56	-	-	-	-	-	606
CO ALT UPM − CO PAT UPM	−521	-	-	-	-	-	3.8	-	-	-	-	-	607
													608
p(GR ALT BYU > GR PAT BYU)	0.52	-	-	-	-	-	0.66	-	-	-	-	-	609
GR ALT BYU − GR PAT BYU	20	-	-	-	-	-	5.5	-	-	-	-	-	610
p(GR ALT FUB > GR PAT FUB)	0.34	-	-	-	-	-	0.56	-	-	-	-	-	611
GR ALT FUB − GR PAT FUB	−363	-	-	-	-	-	2.3	-	-	-	-	-	612
p(GR ALT UA > GR PAT UA)	0.34	-	-	-	-	-	0.62	-	-	-	-	-	613
GR ALT UA − GR PAT UA	−254	-	-	-	-	-	6.0	-	-	-	-	-	614
p(GR ALT UPM > GR PAT UPM)	0.44	-	-	-	-	-	0.60	-	-	-	-	-	615
GR ALT UPM − GR PAT UPM	−102	-	-	-	-	-	6.1	-	-	-	-	-	616
task × variant × site	**compare:**	**variant**					**compare:**	**variant**					669
p(t1 ALT BYU > t1 PAT BYU)	0.41	-	-	-	-	-	0.74	-	-	-	-	-	670
t1 ALT BYU − t1 PAT BYU	−125	-	-	-	-	-	9.4	-	-	-	-	-	671
p(t1 ALT FUB > t1 PAT FUB)	0.39	-	-	-	-	-	0.56	-	-	-	-	-	672
t1 ALT FUB − t1 PAT FUB	−277	-	-	-	-	-	2.7	-	-	-	-	-	673
p(t1 ALT UA > t1 PAT UA)	0.52	-	-	-	-	-	0.74	-	-	-	-	-	674
t1 ALT UA − t1 PAT UA	26	-	-	-	-	-	11.2	-	-	-	-	-	675
p(t1 ALT UPM > t1 PAT UPM)	0.43	-	-	-	-	-	0.72	-	-	-	-	-	676
t1 ALT UPM − t1 PAT UPM	−153	-	-	-	-	-	13.5	-	-	-	-	-	677
													678
p(t2 ALT BYU > t2 PAT BYU)	0.50	-	-	-	-	-	0.68	-	-	-	-	-	679
t2 ALT BYU − t2 PAT BYU	−6	-	-	-	-	-	7.1	-	-	-	-	-	680
p(t2 ALT FUB > t2 PAT FUB)	0.48	-	-	-	-	-	0.53	-	-	-	-	-	681
t2 ALT FUB − t2 PAT FUB	−12	-	-	-	-	-	1.1	-	-	-	-	-	682
p(t2 ALT UA > t2 PAT UA)	0.39	-	-	-	-	-	0.48	-	-	-	-	-	683
t2 ALT UA − t2 PAT UA	−199	-	-	-	-	-	−0.6	-	-	-	-	-	684
p(t2 ALT UPM > t2 PAT UPM)	0.24	-	-	-	-	-	0.43	-	-	-	-	-	685
t2 ALT UPM − t2 PAT UPM	−470	-	-	-	-	-	−3.6	-	-	-	-	-	686
program × task × variant × site	**compare:**	**variant**					**compare:**	**variant**					791
p(CO t1 ALT BYU > CO t1 PAT BYU)	0.24	-	-	-	-	-	0.81	-	-	-	-	-	792
CO t1 ALT BYU − CO t1 PAT BYU	−336	-	-	-	-	-	12.5	-	-	-	-	-	793
p(CO t1 ALT FUB > CO t1 PAT FUB)	0.62	-	-	-	-	-	0.53	-	-	-	-	-	794
CO t1 ALT FUB − CO t1 PAT FUB	199	-	-	-	-	-	1.4	-	-	-	-	-	795
p(CO t1 ALT UA > CO t1 PAT UA)	0.56	-	-	-	-	-	0.69	-	-	-	-	-	796
CO t1 ALT UA − CO t1 PAT UA	81	-	-	-	-	-	8.6	-	-	-	-	-	797
p(CO t1 ALT UPM > CO t1 PAT UPM)	0.32	-	-	-	-	-	0.64	-	-	-	-	-	798
CO t1 ALT UPM − CO t1 PAT UPM	−388	-	-	-	-	-	8.8	-	-	-	-	-	799
													800
p(CO t2 ALT BYU > CO t2 PAT BYU)	0.53	-	-	-	-	-	0.73	-	-	-	-	-	801
CO t2 ALT BYU − CO t2 PAT BYU	33	-	-	-	-	-	9.3	-	-	-	-	-	802
p(CO t2 ALT FUB > CO t2 PAT FUB)	0.45	-	-	-	-	-	0.53	-	-	-	-	-	803
CO t2 ALT FUB − CO t2 PAT FUB	−51	-	-	-	-	-	1.6	-	-	-	-	-	804
p(CO t2 ALT UA > CO t2 PAT UA)	0.57	-	-	-	-	-	0.51	-	-	-	-	-	805
CO t2 ALT UA − CO t2 PAT UA	79	-	-	-	-	-	0.4	-	-	-	-	-	806

(continued on next page)

(Table Y.1 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6
p(CO t2 ALT UPM > CO t2 PAT UPM)	0.14	-	-	-	-	-	0.48	-	-	-	-	-
CO t2 ALT UPM − CO t2 PAT UPM	−653	-	-	-	-	-	−1.2	-	-	-	-	-
p(GR t1 ALT BYU > GR t1 PAT BYU)	0.58	-	-	-	-	-	0.68	-	-	-	-	-
GR t1 ALT BYU − GR t1 PAT BYU	86	-	-	-	-	-	6.2	-	-	-	-	-
p(GR t1 ALT FUB > GR t1 PAT FUB)	0.16	-	-	-	-	-	0.60	-	-	-	-	-
GR t1 ALT FUB − GR t1 PAT FUB	−754	-	-	-	-	-	4.0	-	-	-	-	-
p(GR t1 ALT UA > GR t1 PAT UA)	0.48	-	-	-	-	-	0.79	-	-	-	-	-
GR t1 ALT UA − GR t1 PAT UA	−30	-	-	-	-	-	13.8	-	-	-	-	-
p(GR t1 ALT UPM > GR t1 PAT UPM)	0.54	-	-	-	-	-	0.81	-	-	-	-	-
GR t1 ALT UPM − GR t1 PAT UPM	83	-	-	-	-	-	18.2	-	-	-	-	-
p(GR t2 ALT BYU > GR t2 PAT BYU)	0.46	-	-	-	-	-	0.64	-	-	-	-	-
GR t2 ALT BYU − GR t2 PAT BYU	−45	-	-	-	-	-	4.9	-	-	-	-	-
p(GR t2 ALT FUB > GR t2 PAT FUB)	0.51	-	-	-	-	-	0.52	-	-	-	-	-
GR t2 ALT FUB − GR t2 PAT FUB	27	-	-	-	-	-	0.7	-	-	-	-	-
p(GR t2 ALT UA > GR t2 PAT UA)	0.21	-	-	-	-	-	0.46	-	-	-	-	-
GR t2 ALT UA − GR t2 PAT UA	−478	-	-	-	-	-	−1.7	-	-	-	-	-
p(GR t2 ALT UPM > GR t2 PAT UPM)	0.34	-	-	-	-	-	0.39	-	-	-	-	-
GR t2 ALT UPM − GR t2 PAT UPM	−287	-	-	-	-	-	−6.0	-	-	-	-	-

Table Y.2: Filtered Bayesian Results. See Appendix Y for a description of how to interpret this table.

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
order	**compare: order**						**compare: order**						11
p(1st > 2nd)	0.96	0.98	0.98	0.97	0.97	0.97	0.13	0.15	0.12	0.15	0.10	0.19	12
1st − 2nd	141	175	180	164	157	142	−3.8	−3.6	−4.5	−3.6	−4.3	−3.1	13
variant	**compare: variant**						**compare: variant**						33
p(PAT > ALT)	0.61	0.58	0.60	0.64	0.63	0.58	0.41	0.41	0.40	0.37	0.33	0.32	34
PAT − ALT	206	104	152	217	133	69	−4.2	−3.5	−3.9	−4.6	−4.9	−4.8	35
program × variant	**compare: variant**						**compare: variant**						36
p(CO PAT > CO ALT)	0.61	0.59	0.63	0.64	0.67	0.50	0.43	0.45	0.42	0.42	0.39	0.40	37
CO PAT − CO ALT	213	129	196	188	195	−2	−3.2	−2.4	−3.2	−2.9	−3.2	−2.7	38
p(GR PAT > GR ALT)	0.60	0.57	0.57	0.63	0.58	0.66	0.39	0.37	0.38	0.31	0.27	0.24	39
GR PAT − GR ALT	200	79	108	245	71	140	−5.2	−4.7	−4.5	−6.3	−6.6	−7.0	40
task × variant	**compare: variant**						**compare: variant**						41
p(t1 PAT > t1 ALT)	0.62	0.57	0.61	0.68	0.64	-	0.35	0.37	0.36	0.29	0.27	-	42
t1 PAT − t1 ALT	242	105	176	328	168	-	−7.5	−5.0	−5.5	−7.3	−6.8	-	43
p(t2 PAT > t2 ALT)	0.59	0.59	0.59	0.59	0.61	-	0.47	0.45	0.44	0.44	0.39	-	44
t2 PAT − t2 ALT	171	103	128	106	98	-	−0.9	−2.1	−2.3	−2.0	−3.0	-	45
program × task × variant	**compare: variant**						**compare: variant**						46
p(CO t1 PAT > CO t1 ALT)	0.67	0.71	0.76	0.81	0.85	-	0.42	0.47	0.40	0.41	0.38	-	47
CO t1 PAT − CO t1 ALT	309	298	398	404	404	-	−3.3	−1.4	−4.1	−3.6	−3.4	-	48
p(GR t1 PAT > GR t1 ALT)	0.58	0.44	0.47	0.55	0.44	-	0.27	0.27	0.33	0.18	0.16	-	49
GR t1 PAT − GR t1 ALT	174	−88	−46	251	−68	-	−11.7	−8.5	−6.9	−10.9	−10.2	-	50
p(CO t2 PAT > CO t2 ALT)	0.56	0.48	0.49	0.47	0.48	-	0.44	0.42	0.44	0.44	0.41	-	51
CO t2 PAT − CO t2 ALT	117	−40	−6	−27	−14	-	−3.2	−3.5	−2.4	−2.2	−2.9	-	52
p(GR t2 PAT > GR t2 ALT)	0.63	0.70	0.68	0.71	0.73	-	0.51	0.47	0.43	0.44	0.38	-	53
GR t2 PAT − GR t2 ALT	225	246	263	239	210	-	1.4	−0.8	−2.2	−1.8	−3.1	-	54
time_or_correctness	**compare: correctness**						**compare: time**						61
p(Low > High)	0.01	0.06	0.11	0.34	0.08	0.13	0.20	0.25	0.26	0.42	0.30	0.23	62
Low − High	−216	−159	−123	−215	−130	−102	−4.0	−2.9	−3.0	−2.2	−2.2	−2.9	63
variant × patKnow	**compare: variant**						**compare: variant**						102
p(ALT Low > PAT Low)	-	0.34	-	-	-	-	-	0.60	-	-	-	-	103
ALT Low − PAT Low	-	−207	-	-	-	-	-	3.6	-	-	-	-	104
p(ALT High > PAT High)	-	0.50	-	-	-	-	-	0.58	-	-	-	-	105
ALT High − PAT High	-	−1	-	-	-	-	-	3.5	-	-	-	-	106
program × variant × patKnow	**compare: variant**						**compare: variant**						117
p(CO ALT Low > CO PAT Low)	-	0.26	-	-	-	-	-	0.48	-	-	-	-	118
CO ALT Low − CO PAT Low	-	−309	-	-	-	-	-	−0.9	-	-	-	-	119
p(CO ALT High > CO PAT High)	-	0.56	-	-	-	-	-	0.63	-	-	-	-	120
CO ALT High − CO PAT High	-	51	-	-	-	-	-	5.7	-	-	-	-	121
													122
p(GR ALT Low > GR PAT Low)	-	0.41	-	-	-	-	-	0.72	-	-	-	-	123
GR ALT Low − GR PAT Low	-	−105	-	-	-	-	-	8.1	-	-	-	-	124
p(GR ALT High > GR PAT High)	-	0.45	-	-	-	-	-	0.53	-	-	-	-	125
GR ALT High − GR PAT High	-	−54	-	-	-	-	-	1.2	-	-	-	-	126
task × variant × patKnow	**compare: variant**						**compare: variant**						137
p(t1 ALT Low > t1 PAT Low)	-	0.37	-	-	-	-	-	0.65	-	-	-	-	138
t1 ALT Low − t1 PAT Low	-	−193	-	-	-	-	-	5.8	-	-	-	-	139
p(t1 ALT High > t1 PAT High)	-	0.48	-	-	-	-	-	0.61	-	-	-	-	140
t1 ALT High − t1 PAT High	-	−17	-	-	-	-	-	4.1	-	-	-	-	141
													142
p(t2 ALT Low > t2 PAT Low)	-	0.30	-	-	-	-	-	0.54	-	-	-	-	143
t2 ALT Low − t2 PAT Low	-	−221	-	-	-	-	-	1.4	-	-	-	-	144
p(t2 ALT High > t2 PAT High)	-	0.52	-	-	-	-	-	0.56	-	-	-	-	145
t2 ALT High − t2 PAT High	-	15	-	-	-	-	-	2.9	-	-	-	-	146

(continued on next page)

(Table Y.2 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
program × task × variant × patKnow	**compare: variant**						**compare: variant**						167
p(CO t1 ALT Low > CO t1 PAT Low)	-	0.16	-	-	-	-	-	0.44	-	-	-	-	168
CO t1 ALT Low − CO t1 PAT Low	-	−504	-	-	-	-	-	−2.1	-	-	-	-	169
p(CO t1 ALT High > CO t1 PAT High)	-	0.43	-	-	-	-	-	0.62	-	-	-	-	170
CO t1 ALT High − CO t1 PAT High	-	−92	-	-	-	-	-	4.9	-	-	-	-	171
													172
p(CO t2 ALT Low > CO t2 PAT Low)	-	0.37	-	-	-	-	-	0.51	-	-	-	-	173
CO t2 ALT Low − CO t2 PAT Low	-	−115	-	-	-	-	-	0.3	-	-	-	-	174
p(CO t2 ALT High > CO t2 PAT High)	-	0.68	-	-	-	-	-	0.64	-	-	-	-	175
CO t2 ALT High − CO t2 PAT High	-	194	-	-	-	-	-	6.6	-	-	-	-	176
													177
p(GR t1 ALT Low > GR t1 PAT Low)	-	0.59	-	-	-	-	-	0.86	-	-	-	-	178
GR t1 ALT Low − GR t1 PAT Low	-	118	-	-	-	-	-	13.7	-	-	-	-	179
p(GR t1 ALT High > GR t1 PAT High)	-	0.53	-	-	-	-	-	0.59	-	-	-	-	180
GR t1 ALT High − GR t1 PAT High	-	57	-	-	-	-	-	3.4	-	-	-	-	181
													182
p(GR t2 ALT Low > GR t2 PAT Low)	-	0.23	-	-	-	-	-	0.58	-	-	-	-	183
GR t2 ALT Low − GR t2 PAT Low	-	−328	-	-	-	-	-	2.4	-	-	-	-	184
p(GR t2 ALT High > GR t2 PAT High)	-	0.37	-	-	-	-	-	0.47	-	-	-	-	185
GR t2 ALT High − GR t2 PAT High	-	−165	-	-	-	-	-	−0.9	-	-	-	-	186
variant × devExp	**compare: variant**						**compare: variant**						212
p(ALT Low > PAT Low)	-	-	0.35	-	-	-	-	-	0.70	-	-	-	213
ALT Low − PAT Low	-	-	−224	-	-	-	-	-	7.9	-	-	-	214
p(ALT High > PAT High)	-	-	0.45	-	-	-	-	-	0.50	-	-	-	215
ALT High − PAT High	-	-	−81	-	-	-	-	-	−0.2	-	-	-	216
program × variant × devExp	**compare: variant**						**compare: variant**						227
p(CO ALT Low > CO PAT Low)	-	-	0.31	-	-	-	-	-	0.63	-	-	-	228
CO ALT Low − CO PAT Low	-	-	−294	-	-	-	-	-	5.1	-	-	-	229
p(CO ALT High > CO PAT High)	-	-	0.43	-	-	-	-	-	0.53	-	-	-	230
CO ALT High − CO PAT High	-	-	−98	-	-	-	-	-	1.4	-	-	-	231
													232
p(GR ALT Low > GR PAT Low)	-	-	0.39	-	-	-	-	-	0.78	-	-	-	233
GR ALT Low − GR PAT Low	-	-	−153	-	-	-	-	-	10.8	-	-	-	234
p(GR ALT High > GR PAT High)	-	-	0.46	-	-	-	-	-	0.46	-	-	-	235
GR ALT High − GR PAT High	-	-	−63	-	-	-	-	-	−1.7	-	-	-	236
task × variant × devExp	**compare: variant**						**compare: variant**						247
p(t1 ALT Low > t1 PAT Low)	-	-	0.39	-	-	-	-	-	0.73	-	-	-	248
t1 ALT Low − t1 PAT Low	-	-	−183	-	-	-	-	-	9.2	-	-	-	249
p(t1 ALT High > t1 PAT High)	-	-	0.39	-	-	-	-	-	0.55	-	-	-	250
t1 ALT High − t1 PAT High	-	-	−169	-	-	-	-	-	1.8	-	-	-	251
													252
p(t2 ALT Low > t2 PAT Low)	-	-	0.32	-	-	-	-	-	0.68	-	-	-	253
t2 ALT Low − t2 PAT Low	-	-	−265	-	-	-	-	-	6.7	-	-	-	254
p(t2 ALT High > t2 PAT High)	-	-	0.51	-	-	-	-	-	0.45	-	-	-	255
t2 ALT High − t2 PAT High	-	-	8	-	-	-	-	-	−2.1	-	-	-	256
program × task × variant × devExp	**compare: variant**						**compare: variant**						277
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	0.11	-	-	-	-	-	0.57	-	-	-	278
CO t1 ALT Low − CO t1 PAT Low	-	-	−600	-	-	-	-	-	2.5	-	-	-	279
p(CO t1 ALT High > CO t1 PAT High)	-	-	0.36	-	-	-	-	-	0.64	-	-	-	280
CO t1 ALT High − CO t1 PAT High	-	-	−196	-	-	-	-	-	5.6	-	-	-	281
													282
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	0.51	-	-	-	-	-	0.69	-	-	-	283
CO t2 ALT Low − CO t2 PAT Low	-	-	12	-	-	-	-	-	7.7	-	-	-	284
p(CO t2 ALT High > CO t2 PAT High)	-	-	0.50	-	-	-	-	-	0.42	-	-	-	285
CO t2 ALT High − CO t2 PAT High	-	-	0	-	-	-	-	-	−2.9	-	-	-	286
													287
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	0.66	-	-	-	-	-	0.89	-	-	-	288

(continued on next page)

(Table Y.2 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
GR t1 ALT Low − GR t1 PAT Low	-	-	235	-	-	-	-	-	15.8	-	-	-	289
p(GR t1 ALT High > GR t1 PAT High)	-	-	0.41	-	-	-	-	-	0.46	-	-	-	290
GR t1 ALT High − GR t1 PAT High	-	-	−143	-	-	-	-	-	−2.1	-	-	-	291
													292
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	0.12	-	-	-	-	-	0.66	-	-	-	293
GR t2 ALT Low − GR t2 PAT Low	-	-	−542	-	-	-	-	-	5.8	-	-	-	294
p(GR t2 ALT High > GR t2 PAT High)	-	-	0.51	-	-	-	-	-	0.47	-	-	-	295
GR t2 ALT High − GR t2 PAT High	-	-	16	-	-	-	-	-	−1.3	-	-	-	296
variant × time_or_correctness	**compare: variant**						**compare: variant**						322
p(ALT Low > PAT Low)	-	-	-	0.30	-	-	-	-	-	0.67	-	-	323
ALT Low − PAT Low	-	-	-	−327	-	-	-	-	-	5.6	-	-	324
p(ALT High > PAT High)	-	-	-	0.43	-	-	-	-	-	0.60	-	-	325
ALT High − PAT High	-	-	-	−106	-	-	-	-	-	3.7	-	-	326
program × variant × time_or_correctness	**compare: variant**						**compare: variant**						337
p(CO ALT Low > CO PAT Low)	-	-	-	0.33	-	-	-	-	-	0.65	-	-	338
CO ALT Low − CO PAT Low	-	-	-	−204	-	-	-	-	-	5.4	-	-	339
p(CO ALT High > CO PAT High)	-	-	-	0.39	-	-	-	-	-	0.51	-	-	340
CO ALT High − CO PAT High	-	-	-	−173	-	-	-	-	-	0.4	-	-	341
													342
p(GR ALT Low > GR PAT Low)	-	-	-	0.26	-	-	-	-	-	0.69	-	-	343
GR ALT Low − GR PAT Low	-	-	-	−451	-	-	-	-	-	5.7	-	-	344
p(GR ALT High > GR PAT High)	-	-	-	0.48	-	-	-	-	-	0.70	-	-	345
GR ALT High − GR PAT High	-	-	-	−39	-	-	-	-	-	7.0	-	-	346
task × variant × time_or_correctness	**compare: variant**						**compare: variant**						357
p(t1 ALT Low > t1 PAT Low)	-	-	-	0.19	-	-	-	-	-	0.77	-	-	358
t1 ALT Low − t1 PAT Low	-	-	-	−576	-	-	-	-	-	9.2	-	-	359
p(t1 ALT High > t1 PAT High)	-	-	-	0.45	-	-	-	-	-	0.64	-	-	360
t1 ALT High − t1 PAT High	-	-	-	−79	-	-	-	-	-	5.3	-	-	361
													362
p(t2 ALT Low > t2 PAT Low)	-	-	-	0.40	-	-	-	-	-	0.56	-	-	363
t2 ALT Low − t2 PAT Low	-	-	-	−79	-	-	-	-	-	1.9	-	-	364
p(t2 ALT High > t2 PAT High)	-	-	-	0.42	-	-	-	-	-	0.56	-	-	365
t2 ALT High − t2 PAT High	-	-	-	−133	-	-	-	-	-	2.1	-	-	366
program × task × variant × time_or_correctness	**compare: variant**						**compare: variant**						387
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	-	0.26	-	-	-	-	-	0.76	-	-	388
CO t1 ALT Low − CO t1 PAT Low	-	-	-	−331	-	-	-	-	-	9.4	-	-	389
p(CO t1 ALT High > CO t1 PAT High)	-	-	-	0.13	-	-	-	-	-	0.43	-	-	390
CO t1 ALT High − CO t1 PAT High	-	-	-	−477	-	-	-	-	-	−2.2	-	-	391
													392
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	-	0.40	-	-	-	-	-	0.54	-	-	393
CO t2 ALT Low − CO t2 PAT Low	-	-	-	−77	-	-	-	-	-	1.4	-	-	394
p(CO t2 ALT High > CO t2 PAT High)	-	-	-	0.66	-	-	-	-	-	0.58	-	-	395
CO t2 ALT High − CO t2 PAT High	-	-	-	131	-	-	-	-	-	3.0	-	-	396
													397
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	-	0.12	-	-	-	-	-	0.79	-	-	398
GR t1 ALT Low − GR t1 PAT Low	-	-	-	−821	-	-	-	-	-	8.9	-	-	399
p(GR t1 ALT High > GR t1 PAT High)	-	-	-	0.78	-	-	-	-	-	0.85	-	-	400
GR t1 ALT High − GR t1 PAT High	-	-	-	318	-	-	-	-	-	12.8	-	-	401
													402
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	-	0.41	-	-	-	-	-	0.58	-	-	403
GR t2 ALT Low − GR t2 PAT Low	-	-	-	−82	-	-	-	-	-	2.4	-	-	404
p(GR t2 ALT High > GR t2 PAT High)	-	-	-	0.17	-	-	-	-	-	0.54	-	-	405
GR t2 ALT High − GR t2 PAT High	-	-	-	−397	-	-	-	-	-	1.1	-	-	406
variant × site	**compare: variant**						**compare: variant**						538
p(ALT BYU > PAT BYU)	0.39	-	-	-	-	-	0.68	-	-	-	-	-	539

(continued on next page)

(Table Y.2 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
ALT BYU − PAT BYU	−147	-	-	-	-	-	7.0	-	-	-	-	-	540
p(ALT FUB > PAT FUB)	0.43	-	-	-	-	-	0.55	-	-	-	-	-	541
ALT FUB − PAT FUB	−166	-	-	-	-	-	2.2	-	-	-	-	-	542
p(ALT UA > PAT UA)	0.43	-	-	-	-	-	0.61	-	-	-	-	-	543
ALT UA − PAT UA	−140	-	-	-	-	-	6.2	-	-	-	-	-	544
p(ALT UPM > PAT UPM)	0.31	-	-	-	-	-	0.52	-	-	-	-	-	545
ALT UPM − PAT UPM	−372	-	-	-	-	-	1.2	-	-	-	-	-	546
program × variant × site	**compare: variant**						**compare: variant**						599
p(CO ALT BYU > CO PAT BYU)	0.34	-	-	-	-	-	0.65	-	-	-	-	-	600
CO ALT BYU − CO PAT BYU	−228	-	-	-	-	-	6.5	-	-	-	-	-	601
p(CO ALT FUB > CO PAT FUB)	0.54	-	-	-	-	-	0.54	-	-	-	-	-	602
CO ALT FUB − CO PAT FUB	75	-	-	-	-	-	1.7	-	-	-	-	-	603
p(CO ALT UA > CO PAT UA)	0.45	-	-	-	-	-	0.57	-	-	-	-	-	604
CO ALT UA − CO PAT UA	−120	-	-	-	-	-	3.9	-	-	-	-	-	605
p(CO ALT UPM > CO PAT UPM)	0.22	-	-	-	-	-	0.51	-	-	-	-	-	606
CO ALT UPM − CO PAT UPM	−579	-	-	-	-	-	0.7	-	-	-	-	-	607
													608
p(GR ALT BYU > GR PAT BYU)	0.44	-	-	-	-	-	0.70	-	-	-	-	-	609
GR ALT BYU − GR PAT BYU	−66	-	-	-	-	-	7.6	-	-	-	-	-	610
p(GR ALT FUB > GR PAT FUB)	0.32	-	-	-	-	-	0.56	-	-	-	-	-	611
GR ALT FUB − GR PAT FUB	−407	-	-	-	-	-	2.7	-	-	-	-	-	612
p(GR ALT UA > GR PAT UA)	0.40	-	-	-	-	-	0.65	-	-	-	-	-	613
GR ALT UA − GR PAT UA	−160	-	-	-	-	-	8.6	-	-	-	-	-	614
p(GR ALT UPM > GR PAT UPM)	0.41	-	-	-	-	-	0.53	-	-	-	-	-	615
GR ALT UPM − GR PAT UPM	−165	-	-	-	-	-	1.8	-	-	-	-	-	616
task × variant × site	**compare: variant**						**compare: variant**						669
p(t1 ALT BYU > t1 PAT BYU)	0.33	-	-	-	-	-	0.74	-	-	-	-	-	670
t1 ALT BYU − t1 PAT BYU	−243	-	-	-	-	-	9.5	-	-	-	-	-	671
p(t1 ALT FUB > t1 PAT FUB)	0.37	-	-	-	-	-	0.58	-	-	-	-	-	672
t1 ALT FUB − t1 PAT FUB	−316	-	-	-	-	-	3.3	-	-	-	-	-	673
p(t1 ALT UA > t1 PAT UA)	0.41	-	-	-	-	-	0.66	-	-	-	-	-	674
t1 ALT UA − t1 PAT UA	−156	-	-	-	-	-	9.3	-	-	-	-	-	675
p(t1 ALT UPM > t1 PAT UPM)	0.39	-	-	-	-	-	0.63	-	-	-	-	-	676
t1 ALT UPM − t1 PAT UPM	−251	-	-	-	-	-	7.7	-	-	-	-	-	677
													678
p(t2 ALT BYU > t2 PAT BYU)	0.46	-	-	-	-	-	0.61	-	-	-	-	-	679
t2 ALT BYU − t2 PAT BYU	−51	-	-	-	-	-	4.5	-	-	-	-	-	680
p(t2 ALT FUB > t2 PAT FUB)	0.48	-	-	-	-	-	0.52	-	-	-	-	-	681
t2 ALT FUB − t2 PAT FUB	−16	-	-	-	-	-	1.1	-	-	-	-	-	682
p(t2 ALT UA > t2 PAT UA)	0.44	-	-	-	-	-	0.56	-	-	-	-	-	683
t2 ALT UA − t2 PAT UA	−124	-	-	-	-	-	3.1	-	-	-	-	-	684
p(t2 ALT UPM > t2 PAT UPM)	0.24	-	-	-	-	-	0.41	-	-	-	-	-	685
t2 ALT UPM − t2 PAT UPM	−493	-	-	-	-	-	−5.2	-	-	-	-	-	686
program × task × variant × site	**compare: variant**						**compare: variant**						791
p(CO t1 ALT BYU > CO t1 PAT BYU)	0.18	-	-	-	-	-	0.78	-	-	-	-	-	792
CO t1 ALT BYU − CO t1 PAT BYU	−469	-	-	-	-	-	11.7	-	-	-	-	-	793
p(CO t1 ALT FUB > CO t1 PAT FUB)	0.62	-	-	-	-	-	0.55	-	-	-	-	-	794
CO t1 ALT FUB − CO t1 PAT FUB	209	-	-	-	-	-	2.2	-	-	-	-	-	795
p(CO t1 ALT UA > CO t1 PAT UA)	0.28	-	-	-	-	-	0.45	-	-	-	-	-	796
CO t1 ALT UA − CO t1 PAT UA	−393	-	-	-	-	-	−2.3	-	-	-	-	-	797
p(CO t1 ALT UPM > CO t1 PAT UPM)	0.25	-	-	-	-	-	0.52	-	-	-	-	-	798
CO t1 ALT UPM − CO t1 PAT UPM	−584	-	-	-	-	-	1.4	-	-	-	-	-	799
													800
p(CO t2 ALT BYU > CO t2 PAT BYU)	0.51	-	-	-	-	-	0.53	-	-	-	-	-	801
CO t2 ALT BYU − CO t2 PAT BYU	13	-	-	-	-	-	1.3	-	-	-	-	-	802
p(CO t2 ALT FUB > CO t2 PAT FUB)	0.45	-	-	-	-	-	0.53	-	-	-	-	-	803
CO t2 ALT FUB − CO t2 PAT FUB	−60	-	-	-	-	-	1.2	-	-	-	-	-	804
p(CO t2 ALT UA > CO t2 PAT UA)	0.62	-	-	-	-	-	0.69	-	-	-	-	-	805
CO t2 ALT UA − CO t2 PAT UA	153	-	-	-	-	-	10.1	-	-	-	-	-	806

(continued on next page)

(Table Y.2 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6
p(CO t2 ALT UPM > CO t2 PAT UPM)	0.18	-	-	-	-	-	0.50	-	-	-	-	-
CO t2 ALT UPM − CO t2 PAT UPM	−574	-	-	-	-	-	0.0	-	-	-	-	-
p(GR t1 ALT BYU > GR t1 PAT BYU)	0.49	-	-	-	-	-	0.70	-	-	-	-	-
GR t1 ALT BYU − GR t1 PAT BYU	−18	-	-	-	-	-	7.3	-	-	-	-	-
p(GR t1 ALT FUB > GR t1 PAT FUB)	0.12	-	-	-	-	-	0.61	-	-	-	-	-
GR t1 ALT FUB − GR t1 PAT FUB	−842	-	-	-	-	-	4.5	-	-	-	-	-
p(GR t1 ALT UA > GR t1 PAT UA)	0.54	-	-	-	-	-	0.87	-	-	-	-	-
GR t1 ALT UA − GR t1 PAT UA	81	-	-	-	-	-	21.0	-	-	-	-	-
p(GR t1 ALT UPM > GR t1 PAT UPM)	0.54	-	-	-	-	-	0.74	-	-	-	-	-
GR t1 ALT UPM − GR t1 PAT UPM	82	-	-	-	-	-	14.0	-	-	-	-	-
p(GR t2 ALT BYU > GR t2 PAT BYU)	0.40	-	-	-	-	-	0.70	-	-	-	-	-
GR t2 ALT BYU − GR t2 PAT BYU	−115	-	-	-	-	-	7.8	-	-	-	-	-
p(GR t2 ALT FUB > GR t2 PAT FUB)	0.52	-	-	-	-	-	0.52	-	-	-	-	-
GR t2 ALT FUB − GR t2 PAT FUB	29	-	-	-	-	-	0.9	-	-	-	-	-
p(GR t2 ALT UA > GR t2 PAT UA)	0.27	-	-	-	-	-	0.43	-	-	-	-	-
GR t2 ALT UA − GR t2 PAT UA	−401	-	-	-	-	-	−3.8	-	-	-	-	-
p(GR t2 ALT UPM > GR t2 PAT UPM)	0.29	-	-	-	-	-	0.32	-	-	-	-	-
GR t2 ALT UPM − GR t2 PAT UPM	−412	-	-	-	-	-	−10.4	-	-	-	-	-

Appendix Z

Example Bayesian Model Specification[1]

In this appendix, we provide a complete mathematical specification for the Bayesian models we constructed for E_joint. In addition to the likelihood and priors, we also describe the complete conditional distributions. The complete conditionals, which are derived from the likelihood and priors, are the distributions ultimately needed to implement Gibbs sampling. This appendix assumes familiarity with Section 4.4.3 of the main paper.

Z.1 Likelihood Functions

Following Felt's example [69], each Bayesian model assumes an average work cost κ, common to all observations in that model. We then add to or subtract from κ based on the experimental conditions associated with each observation. Explanatory variables are represented as sets of parameters, one parameter for each level of each variable—e.g., *variant* would be represented by two parameters, one for PAT and one for ALT. To accommodate continuous variables, we discretize them into categories (see below). By representing each explanatory variable as a set of categories, each of which is individually estimated by its own parameter, the Bayesian models avoid linearity assumptions. The mean of each observation is thus the sum of all applicable parameters (one for each explanatory variable), plus κ. Additive models of this form allow us to test the impact of experimental conditions by simply comparing posterior distributions.

[1] Cited in Chapter 4.

We run separate models for each response variable. All models incorporate the full 206 observations, comprising CO data from 52 participants, plus GR data from 51 participants, with two observations per participant per program. Since time is skewed high and cannot be negative, we model it as a gamma distribution—denoted $\Gamma(k, \theta)$, where k and θ represent the customary shape and scale parameters, respectively. Further, since correctness is a percentage, we model it as a beta distribution—denoted $\mathrm{B}(\alpha, \beta)$, where α and β are the customary shape parameters. To simplify interpretation, we use *method of moments*[2] to reparameterize both distributions in terms of mean (μ) and variance (σ^2), as follows:

$$\begin{aligned}
\Gamma(k,\theta) \;&\equiv\; \Gamma\left(\frac{\mu^2}{\sigma^2},\ \frac{\sigma^2}{\mu}\right) \;\equiv\; \Gamma'(\mu,\sigma^2),\ \text{for } \mu,\sigma^2 > 0 \\
\mathrm{B}(\alpha,\beta) \;&\equiv\; \mathrm{B}\left(\ x\mu,\ x(1-\mu)\ \right) \;\equiv\; \mathrm{B}'(\mu,\sigma^2),\ \text{where} \\
x &= \left(\frac{\mu(1-\mu)}{\sigma^2} - 1\right) \text{ and } \sigma^2 < \mu(1-\mu)
\end{aligned}$$

Accordingly, we model each time observation as a gamma-distributed sum of additive effects, denoted by $\Gamma'(\mu,\sigma^2) \equiv \Gamma'(\ \sum parameters + \kappa,\ \sigma^2\ )$, and we model each correctness observation as a beta-distributed sum of additive effects, denoted by $\mathrm{B}'(\mu,\sigma^2)$.[3]

In the following list, we describe the translation of explanatory variables into model parameters. In cases where we discretize a continuous variable, our divisions are based on clustering and/or interpretation of the metric's scale. Since the explanatory variables are represented as categorical effects, each defines a *set* of model parameters. Accordingly, we assign an upper-case letter to each explanatory variable for use in later equations. We skip some letters to avoid confusion with symbols that are used elsewhere in this appendix.

[2] This method can generate invalid parameters in some cases. When this happens, we simply reject the current candidate sample. Consequently, our models may take slightly longer to converge.

[3] For an up-opening parabolic beta distribution (i.e., α and β both between 0 and 1), observations of 0 and 1 cause divide by zero errors in the *probability density function* (PDF). In these cases, the only logical interpretation is infinite likelihood, which in turn causes Metropolis to accept candidates regardless of their value. To avoid this problem, we actually discretize the range of the beta PDF into five equal-width buckets, one for each of the five possible correctness scores. Integrating over each bucket yields a beta-like *probability mass function* (PMF), which no longer encounters infinite likelihoods, but which still sums to 1.

- *subjectID* (A): Identity of the participant, comprising 53 parameters, one for each participant in the study.
- *program* (C): Program being tested, comprising 2 parameters, representing CO and GR.
- *variant* (D): Program variant, comprising 2 parameters, representing PAT and ALT.
- *order* (E): Program order, comprising 2 parameters, representing whether the program was administered first or second.
- *task* (F): Program task, comprising 2 parameters, representing coding and comprehension.
- *site* (G): Experiment site, comprising 4 parameters, representing BYU, FUB, UA, and UPM.
- *devExp* (J): Developer experience aggregate metric, comprising 2 parameters, representing high and low developer experience (high = scores of 4.5–7.0 inclusive, matching 22 of 53 participants).
- *patKnow* (L): Pattern knowledge aggregate metric, comprising 2 parameters, representing high and low design pattern knowledge (high = scores of 3.5–7.0 inclusive, matching 21 of 53 participants).
- *time* or *correctness* (M): In a given model we use whichever effect is not the response variable. Controls for correlations between time and correctness (e.g., achieving a higher correctness score simply by working longer). Time comprises 2 parameters, representing high and low times. High times are determined on a per-task basis (matching a total of 91 out of 206 observations) as follows:
  - CO task 1: $\geq$2000 sec. (26 of 52 observations)
  - CO task 2: $\geq$500 sec. (25 of 52 observations)
  - GR task 1: $\geq$1900 sec. (22 of 51 observations)

  - GR task 2: $\geq$700 sec. (18 of 51 observations)

  Correctness comprises 2 parameters, representing high and low correctness (high = scores of 75–100 inclusive, matching 103 of 206 observations).

- *baseOffset* (κ): An offset common to all observations, comprising 1 parameter, representing the base time or correctness achieved by all observations in a given model.
- *obsVar* (V): Observation variance, comprising 4 parameters, one for each program task. We estimate variance separately for each task because tasks that naturally take more time also tend to display a larger variance.

Table Z.1 informally describes the Bayesian models as a set of templates, which list the explanatory variables and interactions involved. We instantiate each of the six templates twice to model both *time* and *correctness*, thus yielding a total of 12 models. We denote time models as T1–T6 and correctness models as C1–C6. We provide the likelihood function for model T1 below. All other likelihood functions can be derived from that of T1 by simply adjusting the placement of variables with respect to the interaction term. In this regard, notice that the six templates shown in Table Z.1 differ only by which variables are placed in the interaction.

In the likelihood function below, y denotes a single observation, whereas Y denotes the set of all observations. For each explanatory variable, we use its corresponding lower-case letter to denote its levels. For example, C is the set of all *program* parameters, one for CO and one for GR (as defined previously). Accordingly, $c \in \{\mathrm{CO}, \mathrm{GR}\}$. Thus C_c denotes a single model parameter corresponding with program c, and y_c denotes a single observation measured on program c. Continuing this scheme, V_{cf} denotes a single variance parameter corresponding with program c and task f. Further, we represent interactions by the capital letter I, with a subscript to denote the specific type—e.g., I_{CD} denotes the interaction *program* × *variant*, comprising 4 parameters. Thus I_{cd} denotes a single interaction parameter corresponding with program c and variant d.

Table Z.1: Templates for the 12 Bayesian models. Each template is instantiated twice, once for *time* (models T1–T6) and once for *correctness* (models C1–C6). The cardinality of each effect is shown in parentheses—i.e., the number of parameters required to represent all levels of the effect in a model.

	Mean Effects (comprising μ)			
Template Number	Individual Effects	Interaction Effects	Variance Effects (comprising σ^2)	Total Model Parameters
1	subjectID (53), order (2), devExp (2), patKnow (2), time_or_correctness (2), baseOffset (1)	program × variant × task × site (32)	obsVar (4)	98
2	subjectID (53), order (2), site (4), devExp (2), time_or_correctness (2), baseOffset (1)	program × variant × task × patKnow (16)	obsVar (4)	84
3	subjectID (53), order (2), site (4), patKnow (2), time_or_correctness (2), baseOffset (1)	program × variant × task × devExp (16)	obsVar (4)	84
4	subjectID (53), order (2), site (4), devExp (2), patKnow (2), baseOffset (1)	program × variant × task × time_or_correctness (16)	obsVar (4)	84
5	subjectID (53), order (2), site (4), devExp (2), patKnow (2), time_or_correctness (2), baseOffset (1)	program × variant × task (8)	obsVar (4)	78
6	subjectID (53), order (2), task (2), site (4), devExp (2), patKnow (2), time_or_correctness (2), baseOffset (1)	program × variant (4)	obsVar (4)	76

Based on these notations, the density of a single observation for model T1 is:

$$y_{acdefgjlm} | A_a, E_e, J_j, L_l, M_m, I_{cdfg}, \kappa, V_{cf}$$
$$\sim \Gamma'\big(A_a + E_e + J_j + L_l + M_m + I_{cdfg} + \kappa,\ \ V_{cf}\big)$$

Thus for model T1, each observation depends on 8 parameters. Since we explicitly model all known dependencies, e.g., *subjectID*, *program*, *variant*, etc., the data can reasonably be considered independent. Consequently, the likelihood (lik) for model T1 is simply the product of the density of each observation:

$$lik\,(Y | A, E, J, L, M, I_{CDFG}, \kappa, V)$$
$$= \prod_{y \in Y} p\,(y_{acdefgjlm} | A_a, E_e, J_j, L_l, M_m, I_{cdfg}, \kappa, V_{cf})$$

As a final explanation of the notation, consider the following observation for participant 48761:

- *subjectID* $(A) = 48761$
- *program* $(C) = \text{CO}$
- *variant* $(D) = \text{ALT}$
- *order* $(E) = 1$
- *task* $(F) = 1$
- *site* $(G) = \text{BYU}$
- *devExp* $(J) = 4.980224121$ (high)
- *patKnow* $(L) = 2.411764706$ (low)
- *correctness* $(M) = 75$ (high)
- *time* (response variable) $= 1439$

For this observation, the model T1 likelihood function would be configured as follows (comprising 8 parameters):

$$\begin{aligned} y_{48761,\text{CO},\text{ALT},1,1,\text{BYU},\text{high},\text{low},\text{high}} \mid A_{48761}, E_{\text{order1}}, J_{\text{high}}, L_{\text{low}}, M_{\text{high}}, I_{\text{CO,ALT,task1,BYU}}, \kappa, V_{\text{CO,task1}} \\ \sim \Gamma'\big(A_{48761} + E_{\text{order1}} + J_{\text{high}} + L_{\text{low}} + M_{\text{high}} + I_{\text{CO,ALT,task1,BYU}} + \kappa,\ V_{\text{CO,task1}}\big) \end{aligned}$$

Z.2 Prior Distributions

In this section, we describe prior distributions for all model parameters (see Table Z.2). The rationale for our choice of priors is described in Section 4.4.3.

Z.3 Complete Conditionals

As Felt describes, “A complete conditional distribution over [a parameter] represents the probability of that parameter given the data and the value of every other parameter in

Table Z.2: Prior distributions for all Bayesian model parameters.

Response Variable	Models	CO Task 1 Variance	CO Task 2 Variance	GR Task 1 Variance	GR Task 2 Variance	Base Offset	Explanatory Parameters
time	T1–T6	$\Gamma\,(3, 480000)$	$\Gamma\,(3, 83333)$	$\Gamma\,(3, 403333)$	$\Gamma\,(3, 163333)$	$\mathrm{N}\,(1900, 500^2)$ $3\sigma = 25$ min.	$\mathrm{N}\,(0, 1000^2)$ $3\sigma = 50$ min.
correctness	C1–C6	$\Gamma\,(2, 0.07)$	$\Gamma\,(2, 0.08)$	$\Gamma\,(2, 0.055)$	$\Gamma\,(2, 0.12)$	$\mathrm{N}\,(0.5, (0.4/3)^2)$ $3\sigma = 40$ pts.	$\mathrm{N}\,(0, 0.3^2)$ $3\sigma = 90$ pts.

$\Gamma(k, \theta)$ = gamma distribution, where k and θ represent shape and scale.
$\mathrm{N}(\mu, \sigma^2)$ = normal distribution, where μ and σ^2 represent mean and variance.
3σ = the approximate practical range of a normal distribution on either side of the mean.

the graph" [69, p. 105]. The easiest way to derive a complete conditional is to first derive the joint posterior distribution, $p(\Theta|Y)$, from Bayes' theorem. Recalling Bayes' theorem, $p(\Theta|Y) = p(Y|\Theta)p(\Theta)/p(Y)$, note that $p(Y)$ is constant with respect to Θ, and for Gibbs sampling with Metropolis, all constant terms can be ignored. Thus we only need to derive the numerator. Additionally, to avoid problems with machine precision, we perform all calculations on the log scale. Consequently, the final term we need to derive is $ln(p(Y|\Theta)p(\Theta))$, where $p(Y|\Theta)$ is the likelihood function previously discussed, denoted $lik(Y|\Theta)$.

To allow for a representation of the joint posterior applicable to all models, let Ω be the set of all explanatory parameters, including the interaction parameters—i.e., all parameters except for the base offset (κ) and the four task variances (V):

$$\Omega = \{\omega \in \Theta : \omega \neq k, \omega \notin V\}$$

Thus, $\Omega \subset \Theta$. Further, since our parameters are independent, we can write $p(\Theta)$ as the product of the probability of each parameter. Thus the log-scale joint posterior for all models is:

$$\begin{aligned} ln(p(\Theta|Y)) &= ln\left(\frac{p(Y|\Theta)p(\Theta)}{p(Y)}\right) \\ &\propto ln(p(Y|\Theta)p(\Theta)) \\ &\propto ln(lik(Y|\kappa, V, \Omega)p(\kappa, V, \Omega)) \\ &\propto ln(lik(Y|\kappa, V, \Omega)p(\kappa)p(V)p(\Omega)) \end{aligned}$$

$$
\begin{aligned}
&\propto ln(lik(Y|\kappa, V, \Omega)) + ln(p(\kappa)) \\
&\quad + ln(p(V)) + ln(p(\Omega)) \\
&\propto ln(lik(Y|\kappa, V, \Omega)) + ln(p(\kappa)) \\
&\quad + \sum_{V_{ce} \in V} ln(p(V_{ce})) \\
&\quad + \sum_{\Omega_\omega \in \Omega} ln(p(\Omega_\omega))
\end{aligned}
$$

Deriving complete conditionals from the joint posterior is simply a matter of recognizing that all terms not involving the parameter of interest are constants. Accordingly, the complete conditionals for each individual parameter are (where γ is the sum of all constant terms):

$$
\begin{aligned}
[\kappa] &= ln(lik(Y|\Omega, \kappa, V)) + ln(p(\kappa)) + \gamma \\
[V_{ce} \in V] &= ln(lik(Y|\Omega, \kappa, V)) + ln(p(V_{ce})) + \gamma \\
[\Omega_\omega \in \Omega] &= ln(lik(Y|\Omega, \kappa, V)) + ln(p(\Omega_\omega)) + \gamma
\end{aligned}
$$

As mentioned above, Gibbs sampling with Metropolis is setup such that we can ignore any constant terms, so all γ terms can be dropped in the implementation.

However, to implement these conditionals, we still need to know the precise form of the four non-constant terms. We begin with the likelihood term, which is common to all complete conditionals, $ln(lik(Y|\Omega, \kappa, V))$. Recall from the previous discussion that the likelihood of the data is equal to the product of the density of each observation. Also, recall that *time* is gamma-distributed and *correctness* is beta-distributed. Thus the log-likelihood for the *time* models (T1-T6) is:

$$
ln(lik(Y|\Omega, \kappa, V))_{\text{T1–T6}} = ln\left(\prod_{i=1}^{|Y|} \frac{y_i^{k_i-1} e^{-y_i/\theta_i}}{\Gamma(k_i)\theta_i^{k_i}}\right)
$$

$$
= \sum_{i=1}^{|Y|} \left[(k_i - 1)ln(y_i) - \frac{y_i}{\theta_i} - ln(\Gamma(k_i)) - k_i ln(\theta_i)\right]
$$

where $\Gamma(k_i)$ refers to the *gamma function*, and k and θ are defined for the given model in terms of $\Gamma'(\mu, \sigma^2)$, as previously shown. Similarly, the log-likelihood for the *correctness* models (C1-C6) is:

$$ln(lik(Y|\Omega, \kappa, V))_{\mathrm{C1–C6}} = ln\left(\prod_{i=1}^{|Y|} \frac{y_i^{\alpha_i-1}(1-y_i)^{\beta_i-1}}{\mathrm{B}(\alpha_i, \beta_i)}\right)$$
$$= \sum_{i=1}^{|Y|} \left[(\alpha_i - 1)ln(y_i) + (\beta_i - 1)ln(1-y_i) - ln(\mathrm{B}(\alpha_i, \beta_i))\right]$$

where $\mathrm{B}(\alpha_i, \beta_i)$ refers to the *beta function*, and α and β are defined for the given model in terms of $\mathrm{B}'(\mu, \sigma^2)$, as previously shown. Also notice that no terms can be dropped from any of the log-likelihoods, since the parameter of interest is always a component of μ or σ^2, both of which influence k, θ, α, and β.

The next complete conditional term, $ln(p(\kappa))$, denotes the log-probability of the base offset common to all observations. The base offset prior is normally distributed, such that:

$$\begin{aligned} ln(p(\kappa)) &= ln\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}\right) \\ &\propto -(x-\mu)^2/2\sigma^2, \end{aligned}$$

where x is a candidate sample for parameter κ, and where μ and σ^2 are defined as shown in Table Z.2. Since $1/\sqrt{2\pi\sigma^2}$ is constant with respect to x, it can be ignored as belonging to γ.

The next complete conditional term, $ln(p(V_{ce}))$, denotes the log-probability of a variance parameter. All variance priors are gamma distributed, such that:

$$\begin{aligned} ln(p(V_{ce})) &= ln\left(\frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta}\right) \\ &\propto (k-1)ln(x) - x/\theta, \end{aligned}$$

where x is a candidate sample for parameter V_{ce}, and where k and θ are defined as shown in Table Z.2. Since $1/\Gamma(k)\theta^k$ is constant with respect to x, it can be ignored as belonging to γ.

The last complete conditional term, $ln(p(\Omega_\omega))$, denotes the log-probability of an explanatory parameter. All explanatory parameter priors are normally distributed, such that:

$$\begin{aligned} ln(p(\Omega_\omega)) &= ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/2\sigma^2}\right) \\ &\propto -x^2/2\sigma^2, \end{aligned}$$

where x is a candidate sample for parameter Ω_ω, and where μ and σ^2 are defined as shown in Table Z.2. Since $\mu = 0$ for both *time* and *correctness*, and $1/\sqrt{2\pi\sigma^2}$ is constant with respect to x, both terms can be ignored.

References

[1] S. Adolph, W. Hall, and P. Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, 2011.

[2] C. Alexander. *The Timeless Way of Building*. Oxford University Press, Oxford, NY, 1979.

[3] P. D. Allison. *Multiple Regression: A Primer*. Sage Publications, Thousand Oaks, CA, 1999.

[4] J. P. F. Almqvist. Replication of controlled experiments in empirical software engineering—a survey. Master's thesis, Department of Computer Science, Lund University, Lund, Sweden, 2006.

[5] U. Alon. Simplicity in biology. *Nature*, 446(7135):497, 2007.

[6] A. Ampatzoglou, S. Charalampidou, and I. Stamelos. Research state of the art on GoF design patterns: A mapping study. *Journal of Systems and Software*, 86(7):1945–1964, 2013.

[7] P. G. Armour. The five orders of ignorance. *Communications of the ACM*, 43(10):17–20, 2000.

[8] P. G. Armour. *The Laws of Software Process: A New Model for the Production and Management of Software*. CRC Press, Boca Raton, FL, 2004.

[9] N. R. Augustine. Augustine's laws and major system development programs. *Defense Systems Management Review*, pages 50–76, 1979.

[10] L. Aversano, L. Cerulo, and M. D. Penta. Relationship between design patterns defects and crosscutting concern scattering degree: An empirical study. *IET Software*, 3(5):395–409, 2009.

[11] H. M. Bahr, T. Caplow, and B. A. Chadwick. Middletown iii: Problems of replication, longitudinal measurement, and triangulation. *Annual Review of Sociology*, 9(1):243–264, 1983.

[12] S. E. Bailey, S. S. Godbole, C. D. Knutson, and J. L. Krein. A decade of Conway's Law: A literature review from 2003–2012. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–14, 2013.

[13] R. M. Baron and D. A. Kenny. The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology*, 51(6):1173–1182, 1986.

[14] V. R. Basili. Software modeling and measurement: The Goal/Question/Metric paradigm. Technical report, UMIACS TR-92-96, Department of Computer Science, University of Maryland, 1992.

[15] V. R. Basili. The role of experimentation in software engineering: Past, current, and future. In *International Conference on Software Engineering*, pages 442–449, 1996.

[16] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *Transactions on Software Engineering*, 22(10):751–761, 1996.

[17] V. R. Basili, F. E. McGarry, R. Pajerski, and M. V. Zelkowitz. Lessons learned from 25 years of process improvement: The rise and fall of the NASA Software Engineering Laboratory. In *International Conference on Software Engineering*, pages 69–79, 2002.

[18] V. R. Basili and H. D. Rombach. The TAME project: Towards improvement-oriented software environments. *Transactions on Software Engineering*, 14(6):758–773, 1988.

[19] V. R. Basili, F. J. Shull, and F. Lanubile. Building knowledge through families of experiments. *Transactions on Software Engineering*, 25(4):456–473, 1999.

[20] K. Beck, J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides. Industrial experience with design patterns. In *International Conference on Software Engineering*, pages 103–114, 1996.

[21] I. Berlin. *Karl Marx: His Life and Environment.* Oxford University Press, New York, NY, 4th edition, 1996.

[22] M. J. A. Berry and G. S. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management.* Wiley Publishing, Indianapolis, IN, 2nd edition, 2004.

[23] S. Biffl, E. Serral, D. Winkler, O. Dieste, N. Juristo, and N. Condori-Fernández. Replication data management: Needs and solutions—an initial evaluation of conceptual approaches for integrating heterogeneous replication study data. In *International Symposium on Empirical Software Engineering and Measurement*, pages 233–242, 2013.

[24] M. Bissell. Reproducibility: The risks of the replication drive. *Nature*, 503(7476):333–334, 2013.

[25] K. L. Blatter, T. Gledhill, J. L. Krein, and C. D. Knutson. Impact of communication structure on system design: Towards a controlled test of Conway's Law. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 25–33, 2013.

[26] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[27] L. C. Briand, J. W. Daly, and J. Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3(1):65–117, 1998.

[28] L. C. Briand, J. W. Daly, and J. K. Wüst. A unified framework for coupling measurement in object-oriented systems. *Transactions on Software Engineering*, 25(1):91–121, 1999.

[29] L. C. Briand, S. Morasca, and V. R. Basili. Defining and validating measures for object-based high-level design. *Transactions on Software Engineering*, 25(5):722–743, 1999.

[30] A. Brooks, J. Chambers, C. N. Lee, and F. Mead. A partial replication with a sample size of one: A smoke test for empirical software engineering. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 56–65, 2013.

[31] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood. Replication of experimental results in software engineering. Technical report, RR/95/193, EFoCS-17-95, Department of Computer Science, University of Strathclyde, 1995.

[32] A. Brooks, M. Roper, M. Wood, J. Daly, and J. Miller. Replication's role in software engineering. In F. J. Shull, J. Singer, and D. I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 365–379, London, UK, 2008. Springer-Verlag.

[33] F. P. Brooks. *The Mythical Man Month*. Addison-Wesley, Boston, MA, 1995.

[34] J. S. Bruner and L. Postman. On the perception of incongruity: A paradigm. *Journal of Personality*, 18(2):206–223, 1949.

[35] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu. Automatic code generation from design patterns. *IBM Systems Journal*, 35(2):151–171, 1996.

[36] S. H. Burton, P. M. Bodily, R. G. Morris, C. D. Knutson, and J. L. Krein. Design team perception of development team composition: Implications for Conway's Law. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 52–60, 2011.

[37] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, Chichester, UK, 1996.

[38] BYU SEQuOIA Lab. Software Engineering Quality: Observation, Insight, and Analysis. `http://sequoia.cs.byu.edu/`, 2011. Last accessed: May 2011.

[39] T. Caplow, H. M. Bahr, B. A. Chadwick, V. R. A. Call, and L. Hicks. Compilation of Middletown III and Middletown IV data, 1977–1999 [Muncie, Indiana]. Inter-university Consortium for Political and Social Research, Ann Arbor, MI, 2007.

[40] J. C. Carver. Towards reporting guidelines for experimental replications: A proposal. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–4, 2010.

[41] K. Charmaz. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. Sage Publications, Thousand Oaks, CA, 1st edition, 2006.

[42] M. Ciolkowski. What do we know about perspective-based reading? An approach for quantitative aggregation in software engineering. In *International Symposium on Empirical Software Engineering and Measurement*, pages 133–144, 2009.

[43] R. Collins and M. Makowsky. *The Discovery of Society*. McGraw-Hill, New York, NY, 8th edition, 2010.

[44] T. D. Cook and D. T. Campbell. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin, Boston, MA, 1979.

[45] J. Corbin and A. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Thousand Oaks, CA, 3rd edition, 2007.

[46] M. K. Cowles and B. P. Carlin. Markov chain Monte Carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

[47] J. W. Creswell. *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*. Sage Publications, Thousand Oaks, CA, 2nd edition, 2007.

[48] D. S. Cruzes and T. Dybå. Synthesizing evidence in software engineering research. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.

[49] F. Q. B. da Silva, M. Suassuna, A. C. C. França, A. M. Grubb, T. B. Gouveia, C. V. F. Monteiro, and I. E. dos Santos. Replication of empirical studies in software engineering research: A systematic mapping study. *Empirical Software Engineering*, 19(3):501–557, 2012.

[50] F. Q. B. da Silva, M. Suassuna, R. F. Lopes, T. B. Gouveia, A. C. A. França, J. P. N. de Oliveira, L. F. M. de Oliveira, and A. L. M. Santos. Replication of empirical studies in software engineering: Preliminary findings from a systematic mapping study. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 61–70, 2011.

[51] J. W. Daly. *Replication and a Multi-Method Approach to Empirical Software Engineering Research*. PhD thesis, Department of Computer Science, University of Strathclyde, Glasgow, UK, 1996.

[52] C. V. C. de Magalhães and F. Q. B. da Silva. Towards a taxonomy of replications in empirical software engineering research: A research proposal. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 50–55, 2013.

[53] C. V. C. de Magalhães, F. Q. B. da Silva, and R. E. S. Santos. Investigations about replication of empirical studies in software engineering: Preliminary findings from a mapping study. In *International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10, 2014.

[54] M. H. DeGroot and M. J. Schervish. *Probability and Statistics*. Addison-Wesley, Boston, MA, 4th edition, 2012.

[55] P. Diesing. *How Does Social Science Work? Reflections on Practice*. University of Pittsburgh Press, Pittsburgh, PA, 1991.

[56] O. Dieste, E. Fernández, R. García, and N. Juristo. Hidden evidence behind useless replications. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–8, 2010.

[57] C. F. Dormann, J. Elith, S. Bacher, C. Buchmann, G. Carl, G. Carré, J. R. G. Marquéz, B. Gruber, B. Lafourcade, P. J. Leitão, T. Münkemüller, C. McClean, P. E. Osborne, B. Reineking, B. Schröder, A. K. Skidmore, D. Zurell, and S. Lautenbach. Collinearity: A review of methods to deal with it and a simulation study evaluating their performance. *Ecography*, 36(1):27–46, 2013.

[58] M. Dowson. The ARIANE 5 software failure. *SIGSOFT Software Engineering Notes*, 22(2):84, 1997.

[59] T. Duong. An introduction to kernel density estimation. `http://www.mvstat.net/tduong/research/seminars/seminar-2001-05/index.html`, 2001. Last accessed: May 2014.

[60] T. Dybå. Contextualizing empirical evidence. *IEEE Software*, 30(1):81–83, 2013.

[61] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, 2006.

[62] T. Dybå, R. Prikladnicki, K. Rönkkö, C. Seaman, and J. Sillito. Qualitative research in software engineering. *Empirical Software Engineering*, 16(4):425–429, 2011.

[63] T. Dybå, D. I. K. Sjøberg, and D. S. Cruzes. What works for whom, where, when, and why? On the role of context in empirical software engineering. In *International Symposium on Empirical Software Engineering and Measurement*, pages 19–28, 2012.

[64] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, London, UK, 1993.

[65] J. Elster. *Alexis de Tocqueville, the First Social Scientist*. Cambridge University Press, New York, NY, 2009.

[66] R. M. Emerson, R. I. Fretz, and L. L. Shaw. *Writing Ethnographic Fieldnotes*. University of Chicago Press, Chicago, IL, 1995.

[67] A. Endres and H. D. Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Addison-Wesley, Harlow, UK, 2003.

[68] D. E. Farrar and R. R. Glauber. Multicollinearity in regression analysis: The problem revisited. *The Review of Economics and Statistics*, 49(1):92–107, 1967.

[69] P. Felt. Improving the effectiveness of machine-assisted annotation. Master's thesis, Dept. of Computer Science, Brigham Young University, Provo, UT, 2012.

[70] R. Ferrari, N. H. Madhavji, O. Sudmann, C. Henke, J. Geisler, and W. Schafer. Transitioning from lab studies to large-scale studies: Emerging results from a literal replication. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–5, 2010.

[71] L. Festinger. *A Theory of Cognitive Dissonance*. Stanford University Press, Stanford, CA, 1957.

[72] J. H. Flavell. *The Developmental Psychology of Jean Piaget*. D. Van Nostrand Company, Princeton, NJ, 1963.

[73] G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. In *European Conference on Object-Oriented Programming*, pages 472–495, 1997.

[74] A. C. C. França, P. R. M. da Cunha, and F. Q. B. da Silva. The effect of reasoning strategies on success in early learning of programming: Lessons learned from an external experiment replication. In *International conference on Evaluation and Assessment in Software Engineering*, pages 81–90, 2010.

[75] Freie Universität Berlin. Replication of 'PatMain'. `http://www.inf.fu-berlin.de/w/SE/PatmainReplicationInfo`, 2011. Last accessed: April 2014.

[76] D. Fucci and B. Turhan. A replicated experiment on the effectiveness of test-first development. In *International Symposium on Empirical Software Engineering and Measurement*, pages 103–112, 2013.

[77] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[78] J. Garzás, F. García, and M. Piattini. Do rules and patterns affect design maintainability? *Journal of Computer Science and Technology*, 24(2):262–272, 2009.

[79] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall, Boca Raton, FL, 2nd edition, 2004.

[80] T. F. Gieryn. *Cultural Boundaries of Science: Credibility on the Line.* University of Chicago Press, Chicago, IL, 1999.

[81] H. P. Ginsburg and S. Opper. *Piaget's Theory of Intellectual Development: An Introduction.* Prentice-Hall, Englewood Cliffs, NJ, 1969.

[82] B. G. Glaser and A. L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Transaction Publishers, Piscataway, NJ, 1999.

[83] O. S. Gómez, N. Juristo, and S. Vegas. Replication, reproduction and re-analysis: Three ways for verifying experimental findings. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–7, 2010.

[84] O. S. Gómez, N. Juristo, and S. Vegas. Replications types in experimental disciplines. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.

[85] O. S. Gómez, N. Juristo, and S. Vegas. Understanding replication of experiments in software engineering: A classification. *Information and Software Technology*, 56(8):1033–1048, 2014.

[86] J. M. González-Barahona and G. Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering*, 17(1–2):75–89, 2012.

[87] J. F. González-Maya, D. Zárrate-Charry, Á. Hernández-Arévalo, A. A. Cepeda, S. A. Balaguera-Reina, C. Castaño-Uribe, and C. Ange. Traditional uses of wild felids in the Caribbean region of Colombia: New threats for conservation? *Latin American Journal of Conservation*, 1(1):64–69, 2010.

[88] N. Goodman. *Fact, Fiction, and Forecast.* Harvard University Press, Cambridge, MA, 4th edition, 1983.

[89] D. A. Grandy. *Everyday Quantum Reality.* Indiana University Press, Bloomington, IN, 2010.

[90] M. Hammersley. *What's Wrong with Ethnography?: Methodological Explorations.* Routledge, New York, NY, 1992.

[91] E. Harmon-Jones and J. Mills, editors. *Cognitive Dissonance: Progress on a Pivotal Theory in Social Psychology.* American Psychological Association, Washington, DC, 1999.

[92] K. Hunt. Do we really need more replications? *Psychological Reports*, 36(2):587–593, 1975.

[93] IEEE Computer Society. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[94] Inter-university Consortium for Political and Social Research. Utilization for compilation of Middletown III and Middletown IV data, 1977–1999 [Muncie, Indiana]. `http://www.icpsr.umich.edu/icpsrweb/ICPSR/studies/4604/utilization`, 2011. Last accessed: March 2012.

[95] S. Jeanmart, Y.-G. Guéhéneuc, H. Sahraoui, and N. Habra. Impact of the visitor pattern on program comprehension and maintenance. In *International Symposium on Empirical Software Engineering and Measurement*, pages 69–78, 2009.

[96] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *International Symposium on Empirical Software Engineering*, pages 95–104, 2005.

[97] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1):37–60, 2004.

[98] M. Jørgensen and D. I. K. Sjøberg. Generalization and theory-building in software engineering research. In *International Conference on Empirical Assessment in Software Engineering*, pages 29–35, 2004.

[99] J. Jung, K. Hoefig, D. Domis, A. Jedlitschka, and M. Hiller. Experimental comparison of two safety analysis methods and its replication. In *International Symposium on Empirical Software Engineering and Measurement*, pages 223–232, 2013.

[100] N. Juristo. Towards understanding the replication of software engineering experiments. Keynote address. In *International Symposium on Empirical Software Engineering and Measurement*, page 4, 2013.

[101] N. Juristo. Towards understanding the replication of software engineering experiments. Keynote slides. `http://www.slideshare.net/NataliaJuristo/towards-understanding-the-replication-27777763`, 2013. Last accessed: August 2014.

[102] N. Juristo and S. Vegas. Using differences among replications of software engineering experiments to gain knowledge. In *International Symposium on Empirical Software Engineering and Measurement*, pages 356–366, 2009.

[103] N. Juristo and S. Vegas. Design patterns in software maintenance: An experiment replication at UPM. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 7–14, 2011.

[104] N. Juristo and S. Vegas. The role of non-exact replications in software engineering experiments. *Empirical Software Engineering*, 16(3):295–324, 2011.

[105] N. Juristo, S. Vegas, M. Solari, S. Abrahão, and I. Ramos. A process for managing interaction between experimenters to get useful similar replications. *Information and Software Technology*, 55(2):215–225, 2013.

[106] G. M. Kapfhammer. Empirically evaluating regression testing techniques: Challenges, solutions, and a potential way forward. In *International Workshop on Regression Testing*, pages 99–102, 2011.

[107] B. Kitchenham. Procedures for performing systematic reviews. Technical report, TR/SE-0401, Department of Computer Science, Keele University and 0400011T.1, NICTA, 2004.

[108] B. Kitchenham. The role of replications in empirical software engineering—a word of warning. *Empirical Software Engineering*, 13(2):219–221, 2008.

[109] B. Kitchenham, H. Al-Khilidar, M. A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, 13(1):97–121, 2008.

[110] B. Kitchenham, T. Dybå, and M. Jørgensen. Evidence-based software engineering. In *International Conference on Software Engineering*, pages 273–281, 2004.

[111] B. Kitchenham, E. Mendes, and G. H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *Transactions on Software Engineering*, 33(5):316–329, 2007.

[112] B. Kitchenham, S. L. Pfleeger, and N. Fenton. Towards a framework for software measurement validation. *Transactions on Software Engineering*, 21(12):929–944, 1995.

[113] B. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *Transactions on Software Engineering*, 28(8):721–734, 2002.

[114] B. Kitchenham, L. Pickard, and S. L. Pfleeger. Case studies for method and tool evaluation. *IEEE Software*, 12(4):52–62, 1995.

[115] C. D. Knutson, J. L. Krein, L. Prechelt, and N. Juristo. 1st International Workshop on Replication in Empirical Software Engineering Research (RESER). In *International Conference on Software Engineering*, pages 461–462, 2010.

[116] C. D. Knutson, J. L. Krein, L. Prechelt, and N. Juristo. Report from the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER 2010). *SIGSOFT Software Engineering Notes*, 35(5):42–44, 2010.

[117] T. Kosar, M. Mernik, and J. C. Carver. Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments. *Empirical software engineering*, 17(3):276–304, 2012.

[118] S. Kotsiantis and D. Kanellopoulos. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58, 2006.

[119] H. C. Kraemer. Pitfalls of multisite randomized clinical trials of efficacy and effectiveness. *Schizophrenia Bulletin*, 26(3):533–541, 2000.

[120] J. L. Krein and C. D. Knutson. A case for replication: Synthesizing research methodologies in software engineering. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–10, 2010.

[121] J. L. Krein and C. D. Knutson. The humanity of science versus the complexity of reality: A theoretical study of replication and knowledge production. 2014, submission pending.

[122] J. L. Krein, C. D. Knutson, and C. Bird. Report from the 3rd International Workshop on Replication in Empirical Software Engineering Research (RESER 2013). *SIGSOFT Software Engineering Notes*, 39(1):31–35, 2014.

[123] J. L. Krein, C. D. Knutson, L. Prechelt, and C. Bird. Message from the RESER 2013 workshop chairs. In *International Symposium on Empirical Software Engineering and Measurement*, page 395, 2013.

[124] J. L. Krein, C. D. Knutson, L. Prechelt, and N. Juristo. Report from the 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER 2011). *SIGSOFT Software Engineering Notes*, 37(1):27–30, 2012.

[125] J. L. Krein, A. C. MacLean, D. P. Delorey, C. D. Knutson, and D. L. Eggett. Impact of programming language fragmentation on developer productivity: A SourceForge

empirical study. *International Journal of Open Source Software and Processes*, 2(2):41–61, 2010.

[126] J. L. Krein, L. J. Pratt, A. B. Swenson, A. C. MacLean, C. D. Knutson, and D. L. Eggett. Design patterns in software maintenance: An experiment replication at Brigham Young University. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 25–34, 2011.

[127] J. L. Krein, L. Prechelt, N. Juristo, A. Nanthaamornphong, J. C. Carver, S. Vegas, C. D. Knutson, K. D. Seppi, and D. L. Eggett. A multi-site joint replication of a design patterns experiment using moderator variables to generalize across contexts. 2014, under review.

[128] J. L. Krein, L. Prechelt, N. Juristo, K. D. Seppi, A. Nanthaamornphong, J. C. Carver, S. Vegas, and C. D. Knutson. A method for generalizing across contexts in software engineering experiments. 2014, submission pending.

[129] T. S. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, IL, 3rd edition, 1996.

[130] J. Lau, J. P. A. Ioannidis, and C. H. Schmid. Quantitative synthesis in systematic reviews. *Annals of Internal Medicine*, 127(9):820–826, 1997.

[131] N. G. Leveson. High-pressure steam engines and computer software. Keynote address. In *International Conference on Software Engineering*, pages 2–14, 1992.

[132] N. G. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[133] V. Li. Lost Koreans: Information technology and identity in the former Soviet Union. Master's thesis, School of Letters Sciences GTD, Arizona State University, Tempe, AZ, 2012.

[134] T. Lidz and S. Fleck. *Schizophrenia and the Family*. International Universities Press, New York, NY, 2nd edition, 1985.

[135] R. M. Lindsay and A. S. C. Ehrenberg. The design of replicated studies. *The American Statistician*, 47(3):217–228, 1993.

[136] J.-L. Lions. ARIANE 5, flight 501 failure. Technical report, Inquiry Board, ESA/CNES, 1996.

[137] J. Lung, J. Aranda, S. Easterbrook, and G. Wilson. On the difficulty of replicating human subjects studies in software engineering. In *International Conference on Software Engineering*, pages 191–200, 2008.

[138] N. Machiavelli. *The Essential Writings of Machiavelli.* Peter Constantine, editor and translator. Random House, New York, NY, 2007.

[139] B. B. Machta, R. Chachra, M. K. Transtrum, and J. P. Sethna. Parameter space compression underlies emergent theories and predictive models. *Science*, 342(6158):604–607, 2013.

[140] A. C. MacLean, L. J. Pratt, J. L. Krein, and C. D. Knutson. Threats to validity in analysis of language fragmentation on SourceForge data. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–6, 2010.

[141] C. Mair and M. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *International Symposium on Empirical Software Engineering*, pages 509–518, 2005.

[142] M. V. Mäntylä, C. Lassenius, and J. Vanhanen. Rethinking replication in software engineering: Can we see the forest for the trees? In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–9, 2010.

[143] P. McCullagh and J. A. Nelder. *Generalized Linear Models.* Chapman & Hall, New York, NY, 2nd edition, 1989.

[144] R. A. McLean, W. L. Sanders, and W. W. Stroup. A unified approach to mixed linear models. *The American Statistician*, 45(1):54–64, 1991.

[145] M. G. Mendonça, J. C. Maldonado, M. C. F. de Oliveira, J. C. Carver, S. C. P. F. Fabbri, F. J. Shull, G. H. Travassos, E. N. Höhn, and V. R. Basili. A framework for software engineering experimental replications. In *International Conference on Engineering of Complex Computer Systems*, pages 203–212, 2008.

[146] T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1):1–17, 2012.

[147] J. Miller. Applying meta-analytical procedures to software engineering experiments. *Journal of Systems and Software*, 54(1):29–39, 2000.

[148] J. Miller. Triangulation as a basis for knowledge discovery in software engineering. *Empirical Software Engineering*, 13(2):223–228, 2008.

[149] C. W. Mills. *The Sociological Imagination.* Oxford University Press, New York, NY, 1959.

[150] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Boston, MA, 1997.

[151] A. Mockus, B. Anda, and D. I. K. Sjøberg. Experiences from replicating a case study to investigate reproducibility of software development. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–5, 2010.

[152] T. Mora and W. Bialek. Are biological systems poised at criticality? *Journal of Statistical Physics*, 144(2):268–302, 2011.

[153] F. Moughrabi and P. El-Nazer. What do Palestinian Americans think? Results of a public opinion survey. *Journal of Palestine Studies*, 18(4):91–101, 1989.

[154] M. J. Mulkay. *Science and the Sociology of Knowledge.* George Allen & Unwin, London, UK, 1979.

[155] A. Nanthaamornphong and J. C. Carver. Design patterns in software maintenance: An experiment replication at University of Alabama. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 15–24, 2011.

[156] T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu. Work experience versus refactoring to design patterns: A controlled experiment. In *International Symposium on Foundations of Software Engineering*, pages 12–22, 2006.

[157] F. Nietzsche. *The Gay Science.* Walter Kaufmann, translator. Random House, New York, NY, 1974.

[158] T. J. Ostrand and E. J. Weyuker. Software testing research and software engineering education. In *International Workshop on Future of Software Engineering Research*, pages 273–276, 2010.

[159] K. Petersen and C. Wohlin. Context in industrial software engineering research. In *International Symposium on Empirical Software Engineering and Measurement*, pages 401–404, 2009.

[160] L. M. Pickard, B. Kitchenham, and P. W. Jones. Combining empirical results in software engineering. *Information and Software Technology*, 40(14):811–821, 1998.

[161] R. M. Pirsig. *Zen and the Art of Motorcycle Maintenance.* Bantam New Age, New York, NY, 1981.

[162] G. Poggi. *Durkheim*. Oxford University Press, New York, NY, 2000.

[163] G. Poggi. *Weber: A Short Introduction*. Polity Press, Malden, MA, 2006.

[164] K. R. Popper. *The Logic of Scientific Discovery*. Routledge, New York, NY, 2002.

[165] L. Prechelt. The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really? Technical report, 1999-18, Fakultät für Informatik, Universität Karlsruhe, 1999.

[166] L. Prechelt. PatmainPackage. `http://page.mi.fu-berlin.de/prechelt/Biblio/#package`, 2013. Last accessed: April 2014.

[167] L. Prechelt and M. Liesenberg. Design patterns in software maintenance: An experiment replication at Freie Universität Berlin. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–6, 2011.

[168] L. Prechelt, B. Unger, W. F. Tichy, P. Brössler, and L. G. Votta. A controlled experiment in maintenance comparing design patterns to simpler solutions. *Transactions on Software Engineering*, 27(12):1134–1144, 2001.

[169] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy. Two controlled experiments assessing the usefulness of design pattern documentations in program maintenance. *Transactions on Software Engineering*, 28(6):595–606, 2002.

[170] R. Premraj and K. Herzig. Network versus code metrics to predict defects: A replication study. In *International Symposium on Empirical Software Engineering and Measurement*, pages 215–224, 2011.

[171] Public Broadcasting Service. The first measured century: Middletown III. `http://www.pbs.org/fmc/timeline/dmiddletowniii.htm`, 2012. Last accessed: March 2012.

[172] G. K. Pullum. The great Eskimo vocabulary hoax. *Natural Language & Linguistic Theory*, 7(2), 1989.

[173] R Documentation. Kernel density estimation. `http://stat.ethz.ch/R-manual/R-devel/library/stats/html/density.html`, 2014. Last accessed: May 2014.

[174] F. L. Ramsey and D. W. Schafer. *The Statistical Sleuth: A Course in Methods of Data Analysis*. Duxbury, Pacific Grove, CA, 2nd edition, 2002.

[175] RESER Workshop. 2nd International Workshop on Replication in Empirical Software Engineering Research. `http://sequoia.cs.byu.edu/reser2011`, 2011. Last accessed: May 2011.

[176] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, and E. Astesiano. On the effectiveness of screen mockups in requirements engineering: Results from an internal replication. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.

[177] G. Robles and D. M. Germán. Beyond replication: An example of the potential benefits of replicability in the Mining of Software Repositories community. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 1–4, 2010.

[178] R. Rosenthal. *Experimenter Effects in Behavioral Research.* Appleton-Century-Crofts, East Norwalk, CT, 1966.

[179] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[180] P. Runeson, A. Stefik, and A. Andrews. Variation factors in the design and analysis of replicated controlled experiments. *Empirical Software Engineering*, 19(6):1781–1808, 2014.

[181] P. Runeson, A. Stefik, A. Andrews, S. Grönblom, I. Porres, and S. Siebert. A comparative analysis of three replicated experiments comparing inspection and unit testing. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 35–42, 2011.

[182] H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*, 11(1):3–11, 1968.

[183] N. J. Salkind, editor. *Encyclopedia of Measurement and Statistics.* Sage Publications, Thousand Oaks, CA, 2007.

[184] E. Sapir. *Language: An Introduction to the Study of Speech.* Harcourt Brace and Company, New York, NY, 1921.

[185] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects.* John Wiley & Sons, Chichester, UK, 2000.

[186] S. Schmidt. Shall we really do it again? The powerful concept of replication is neglected in the social sciences. *Review of General Psychology*, 13(2):90–100, 2009.

[187] T. A. Schwandt. Constructivist, interpretivist approaches to human inquiry. In N. K. Denzin and Y. S. Lincoln, editors, *The Landscape of Qualitative Research: Theories and Issues*, pages 221–259, Thousand Oaks, CA, 1998. Sage Publications.

[188] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *Transactions on Software Engineering*, 25(4):557–572, 1999.

[189] F. J. Shull, V. R. Basili, J. C. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonça, and S. Fabbri. Replicating software engineering experiments: Addressing the tacit knowledge problem. In *International Symposium on Empirical Software Engineering*, pages 7–16, 2002.

[190] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo. The role of replications in empirical software engineering. *Empirical Software Engineering*, 13(2):211–218, 2008.

[191] F. J. Shull, M. G. Mendoncça, V. R. Basili, J. C. Carver, J. C. Maldonado, S. Fabbri, G. H. Travassos, and M. C. Ferreira. Knowledge-sharing issues in experimental software engineering. *Empirical Software Engineering*, 9(1–2):111–137, 2004.

[192] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, London, UK, 1986.

[193] G. Simmel. *Georg Simmel on Individuality and Social Forms.* Donald N. Levine, editor. University of Chicago Press, Chicago, IL, 1971.

[194] D. I. K. Sjøberg. Confronting the myth of rapid obsolescence in computing research. *Communications of the ACM*, 53(9):62–67, 2010.

[195] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay. Building theories in software engineering. In F. J. Shull, J. Singer, and D. I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 312–336, London, UK, 2008. Springer-Verlag.

[196] D. I. K. Sjøberg, T. Dybå, and M. Jørgensen. The future of empirical methods in software engineering research. In *Future of Software Engineering*, pages 358–378, 2007.

[197] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanović, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *Transactions on Software Engineering*, 31(9):733–753, 2005.

[198] M. Solari. Identifying experimental incidents in software engineering replications. In *International Symposium on Empirical Software Engineering and Measurement*, pages 213–222, 2013.

[199] M. Solari and S. Vegas. Classifying and analysing replication packages for software engineering experimentation. In *International Conference on Product Focused Software Process Improvement, Workshop Series in Empirical Software Engineering*, pages 1–6, 2006.

[200] P. Solomon, M. M. Cavanaugh, and J. Draine. *Randomized Controlled Trials: Design and Implementation for Community-Based Psychosocial Interventions*. Oxford University Press, Oxford, UK, 2009.

[201] Stan Development Team. Stan: A C++ library for probability and sampling, version 2.3. `http://mc-stan.org/`, 2014. Last accessed: June 2014.

[202] G. J. Stephens, B. Johnson-Kerner, W. Bialek, and W. S. Ryu. Dimensionality and dynamics in the behavior of *C. elegans*. *PLoS computational biology*, 4(4):1–10, 2008.

[203] G. J. Stephens, L. C. Osborne, and W. Bialek. Searching for simplicity in the analysis of neurons and behavior. *Proceedings of the National Academy of Sciences*, 108(3):15565–15571, 2011.

[204] J. Surowiecki. *The Wisdom of Crowds*. Anchor Books, New York, NY, 2005.

[205] A. J. Sutton, K. R. Abrams, D. R. Jones, T. A. Sheldon, and F. Song. *Methods for Meta-Analysis in Medical Research*. John Wiley & Sons, Chichester, UK, 2000.

[206] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.

[207] B. Turhan. On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering*, 17(1–2):62–74, 2012.

[208] M. Twain. *Tom Sawyer Abroad*. Oxford University Press, New York, NY, 1996.

[209] H. A. Valantine, S.-Z. Gao, S. G. Menon, D. G. Renlund, S. A. Hunt, P. Oyer, E. B. Stinson, B. W. Brown, Jr., T. C. Merigan, and J. S. Schroeder. Impact of prophylactic immediate posttransplant ganciclovir on development of transplant atherosclerosis: A post hoc analysis of a randomized, placebo-controlled study. *Circulation*, 100(1):61–66, 1999.

[210] V. van Veen, M. K. Krug, J. W. Schooler, and C. S. Carter. Neural activity predicts attitude change in cognitive dissonance. *Nature Neuroscience*, 12(11):1469–1474, 2009.

[211] S. Vegas, N. Juristo, A. Moreno, M. Solari, and P. Letelier. Analysis of the influence of communication between researchers on experiment replication. In *International Symposium on Empirical Software Engineering*, pages 28–37, 2006.

[212] M. Vokáč, W. F. Tichy, D. I. K. Sjøberg, E. Arisholm, and M. Aldrin. A controlled experiment comparing the maintainability of programs designed with and without design patterns: A replication in a real programming environment. *Empirical Software Engineering*, 9(3):149–195, 2004.

[213] P. Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In *European Conference on Software Maintenance and Reengineering*, pages 77–84, 2001.

[214] E. J. Weyuker, R. M. Bell, and T. J. Ostrand. Replicate, replicate, replicate. In *International Workshop on Replication in Empirical Software Engineering Research*, pages 71–77, 2011.

[215] B. L. Whorf. *Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf.* John B. Carroll, editor. MIT Press, Cambridge, MA, 1956.

[216] K. Wnuk, M. Höst, and B. Regnell. Replication of an experiment on linguistic tool support for consolidation of requirements from multiple sources. *Empirical Software Engineering*, 17(3):305–344, 2012.

[217] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering.* Springer, New York, NY, 2012.

[218] K. H. Wolff, editor. *Georg Simmel, 1858–1918: A Collection of Essays, with Translations and a Bibliography.* Ohio State University Press, Columbus, OH, 1959.

[219] M. Wood, J. Daly, J. Miller, and M. Roper. Multi-method research: An empirical investigation of object-oriented technology. *Journal of Systems and Software*, 48(1):13–26, 1999.

[220] I. M. Zeitlin. *Nietzsche: A Re-examination.* Polity Press, Cambridge, UK, 1994.

[221] E. Zerubavel. *The Fine Line: Making Distinctions in Everyday Life.* The Free Press, New York, NY, 1991.

[222] C. Zhang and D. Budgen. What do we know about the effectiveness of software design patterns? *Transactions on Software Engineering*, 38(5):1213–1231, 2012.