

**Examining the Nature of Collaborative Learning in the Context of Problem Solving in
First-Year and Second-Year Computing Education,
A Naturalistic Exploratory Case Study**

by

Sharon Mason
February 5, 2019

A dissertation submitted to
The Faculty of the Graduate School of
The State University of New York at Buffalo
in partial fulfillment of the requirements for the
degree of

Doctor of Philosophy

Department of Learning and Instruction

ProQuest Number: 13808099

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 13808099

Published by ProQuest LLC (2019). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Acknowledgment

As I complete my dissertation, I am reflecting on my experiences and recognizing that as with so many things in life, while the days were long, the years were short. I loved every minute of this academic experience; the challenges, the surprises and the people I met along the way. As such, the completion of this project is bittersweet as I leave behind the daily classes, assignments, postings, and discussions with my peers and advisors, and I embark on the next steps after this incredible excursion. Thank you to all of the faculty and students at the University of Buffalo who fostered an environment where I learned new things every day. I looked forward to every class.

I am humbled by the energy and dedication beyond my own that contributed to this work, and I thank my committee members, Dr. Noemi Waight, Dr. Mary McVee and Dr. Sam Abramovich for challenging me to think in new directions. In particular, I must acknowledge and thank my advisor, Dr. Noemi Waight, who consistently pushed me to think more deeply about the issues at hand. Dr. Waight's steadfast guidance, encouragement, support and high expectations for quality throughout the process not only pressed me to evolve my thinking in ways that drove me toward the end work, but also modeled for me the qualities in a faculty member that I hope to emulate in the future.

I also want to thank my family who embarked on this voyage and evolution with me. Thank you to my husband, Chris, who was unwavering in his support of my efforts and overall confidence in me. Thank you to my daughters, Jessica and Maddie, for cheering me on every day, stepping up to do extra chores, and listening to all my stories as I read journal articles and learned new things.

Table of Contents

ABSTRACT	VII
CHAPTER 1	1
INTRODUCTION	1
OVERVIEW	1
BACKGROUND AND RATIONALE	3
COMPUTING EDUCATION RESEARCH	4
PROBLEM STATEMENT	9
THE NATURE OF PROBLEM SOLVING	10
<i>Problem solving defined</i>	11
PURPOSE OF THE STUDY AND RESEARCH QUESTIONS	16
SIGNIFICANCE OF THE STUDY	17
CHAPTER 2	19
REVIEW OF THE LITERATURE	19
THE NATURE OF PROBLEM SOLVING	19
CHARACTERISTICS OF ILL-STRUCTURED PROBLEMS AND WELL-STRUCTURED PROBLEMS	20
PROCESSES FOR SOLVING ILL-STRUCTURED PROBLEMS AND WELL-STRUCTURED PROBLEMS.....	22
<i>Ill-structured problem-solving processes</i>	22
Framing the problem.....	23
Identifying and clarifying alternative perspectives of the problem.	24
Generating possible problem solutions.....	24
Assessing the viability of alternative solutions.	25
Monitoring the problem space and solution options.....	25
Implementing, monitoring and adapting the solution.....	26
<i>Well-structured problem-solving processes</i>	26
Defining and representing the problem.....	27
Exploring possible solution strategies.	28
Recall analogical reasoning.	29
Means-end analysis.	29
Decomposing to subproblems.....	32
Generate and test.....	32
Implementing strategies and reflecting back to evaluate the effects of the solutions.....	32
PROBLEM SOLVING THROUGH SOCIAL CONSTRUCTIVIST THEORY.....	33
<i>Theoretical tenets of social constructivism for problem solving</i>	33
<i>Social constructivist problem solving through collaboration</i>	34
<i>Social constructivist problem solving through scaffolding</i>	39
COLLABORATIVE LEARNING AND PROBLEM SOLVING IN COMPUTING EDUCATION RESEARCH.....	44
CHAPTER 3	52
METHOD	52
RESEARCH DESIGN.....	52
PHASE I: PILOT STUDY.....	53
PILOT STUDY DATA ANALYSIS PROCEDURES AND CODES	53
<i>Pilot study faculty interview analysis</i>	53
<i>Pilot study instructional observation analysis</i>	55
<i>Pilot study course artifact analysis</i>	56

PILOT STUDY FINDINGS INFORMING THE CURRENT STUDY.....	57
PHASE II: THE CURRENT STUDY.....	59
CONTEXT OF THE STUDY.....	59
<i>Site</i>	59
<i>Courses</i>	60
<i>Participants</i>	61
RESEARCHER POSITIONALITY.....	62
<i>Research Design</i>	63
<i>Data Analysis</i>	64
DATA SOURCES.....	65
<i>Faculty interviews</i>	65
Interview protocol.....	65
<i>Instructional classroom and laboratory observations</i>	66
Observation protocol.....	66
Field notes.....	66
<i>Course artifacts</i>	67
DATA ANALYSIS.....	67
<i>Problem-solving approaches a priori codes</i>	68
<i>Problem types a priori codes</i>	70
<i>Social learning a priori codes</i>	74
<i>Data analysis and theming</i>	75
TRUSTWORTHINESS.....	77
<i>Worthy topic</i>	77
<i>Rich rigor</i>	78
<i>Sincerity</i>	78
<i>Credibility</i>	79
<i>Resonance</i>	79
<i>Significant contribution</i>	80
<i>Ethics</i>	80
<i>Meaningful coherence</i>	81
CHAPTER 4.....	82
FINDINGS.....	82
THE TYPES OF COMPUTING PROBLEMS POSED BY UNDERGRADUATE	
COMPUTING FACULTY.....	82
PROBLEM TYPES IN COURSE ARTIFACTS.....	85
<i>Algorithmic and rule-using problems in course artifacts</i>	85
<i>Troubleshooting problems in course artifacts</i>	104
<i>Design problems in course artifacts</i>	107
PROBLEM TYPES IN INSTRUCTIONAL CLASSROOM AND LABORATORY OBSERVATIONS.....	112
<i>Algorithmic problems in instructional classroom and laboratory observations</i>	112
<i>Rule-using problems in instructional classroom and laboratory observations</i>	113
<i>Troubleshooting problems in instructional classroom and laboratory observations</i>	114
<i>Design problems in instructional classroom and laboratory observations</i>	115
PROBLEM TYPES IN FACULTY INTERVIEWS.....	118
<i>Algorithmic problems in faculty interviews</i>	118
<i>Rule-using problems in faculty interviews</i>	121
<i>Troubleshooting problems in faculty interviews</i>	127
<i>Design problems in faculty interviews</i>	129

INSTRUCTIONAL APPROACHES TO PROBLEM SOLVING ENACTED BY UNDERGRADUATE COMPUTING FACULTY	135
DECOMPOSITION AS PROBLEM-SOLVING IN INSTRUCTIONAL CLASSROOM AND LABORATORY OBSERVATIONS	135
DECOMPOSITION AS PROBLEM SOLVING IN FACULTY INTERVIEWS.....	137
DECOMPOSITION AS PROBLEM-SOLVING IN COURSE ARTIFACTS	141
THE NATURE OF COLLABORATIVE LEARNING IN UNDERGRADUATE COMPUTING EDUCATION COURSES	145
INSTRUCTIONAL CLASSROOM AND LABORATORY OBSERVATIONS AND COURSE ARTIFACTS	145
<i>Instructor-student collaborative learning.</i>	145
<i>Student-student collaborative learning.</i>	147
Required student-student collaborative learning.	148
Prohibited or ignored student-student collaborative learning.....	150
FACULTY INTERVIEWS	152
<i>Instructor-student collaborative learning.</i>	152
<i>Student-student collaborative learning.</i>	155
Required student-student collaborative learning.	155
Prohibited or ignored student-student collaborative learning.....	158
CHAPTER 5	164
DISCUSSION	164
SUMMARY OF FINDINGS	164
PROBLEM TYPES: COMPUTING FACULTY PREDOMINANTLY ENACTED WELL-STRUCTURED PROBLEMS	165
PROBLEM-SOLVING APPROACH: COMPUTING FACULTY IMPLICITLY DECOMPOSED PROBLEMS AS A MECHANISM TO FACILITATE PROBLEM SOLVING	170
COLLABORATIVE LEARNING: COMPUTING FACULTY EMPHASIZED INFORMAL AND UNSTRUCTURED COLLABORATIVE LEARNING AMONG STUDENTS	172
A PRELIMINARY COLLABORATIVE PROBLEM-SOLVING FRAMEWORK FOR COMPUTING EDUCATION RESEARCH	175
IMPLICATIONS FOR PRACTICE.....	176
<i>Types of problems posed.</i>	177
<i>Problem-solving approaches.</i>	178
<i>Collaborative learning</i>	179
STUDY LIMITATIONS	180
RECOMMENDATIONS FOR FUTURE RESEARCH	181
REFERENCES	184
APPENDIX A - FACULTY INTERVIEW PROTOCOL.....	199
APPENDIX B – BACKGROUND IN COMPUTING EDUCATION CURRICULUM	202
PROBLEM SOLVING IN COMPUTING CURRICULUM GUIDELINES	205
TEAMWORK SKILLS IN COMPUTING CURRICULUM GUIDELINES.....	208
INSTRUCTIONAL APPROACHES IN COMPUTING CURRICULUM GUIDELINES	210

Abstract

This exploratory case study examined the nature of problem solving in terms of the types of problems posed, instructional approaches and collaborative learning, in first-year and second-year post-secondary computing education. Participants included faculty across three computing programs. Instructional classroom and laboratory observations, in-depth faculty interviews and course artifacts served as the data sources. A problem-solving typology and problem-solving literature alongside social constructivist theory, served as the framework to examine the above practices. Findings indicated that faculty predominantly presented aspects of well-structured problems, but also emphasized characteristics across several problem types, including the importance of real-world problems and better or worse solutions in allowing for multiple solution paths. Faculty approaches to problem solving were also well-structured, with a focus on decomposing problems into smaller problems. Collaborative problem solving was limited to informal classroom interactions and unstructured student group work, with faculty focusing on building individual student skillsets in the computing domain. These findings exposed opportunities for improvement related to more explicit and structured instructional experiences for students in posing problems, presenting problem-solving approaches and fostering a collaborative learning environment. Research related to computing education should expand to address the nature of problem types and problem solving in post-secondary computing education, instructional approaches that foster and inform the development of diverse problem-solving knowledge and skills among students and how collaborative learning aligns with real-world computing problems and applications. This study has implications for computing faculty pedagogical knowledge and training, curriculum development, and subsequent professional development related to collaborative problem solving in computing education.

Chapter 1

Introduction

Overview

From weather prediction systems that enable advanced warning of hurricanes to streaming media devices that bring movies and music to our living rooms to diabetes health monitors, computers serve as a tool for solving problems that positively impact people every day. This trend of solving problems using computers is expected to continue in the future. Over one million computer occupation openings are expected from 2014 to 2024 due to a combination of new job creation and workers permanently leaving occupations (Fayer, Lacey, & Watson, 2017). This is in conjunction with the nearly 8.6 million Science, Technology, Engineering and Math (STEM) jobs existing in 2015, as reported by the US Bureau of Labor. These 8.6 million jobs represented 6.2 percent of U.S. employment, with computer occupations making up nearly 45 percent of that (Fayer et al., 2017).

Many efforts are currently underway in both the academic and corporate realms to promote the field of computing to recruit the next generation of thinkers and developers entering the workforce. Resource development devoted to engaging students at the K-12 level is spearheaded by organizations such as the College Board with its work on Advanced Placement (AP) Computer Science Principles course, the Computer Science Teachers Association (CSTA) whose mission is “to empower, engage and advocate for K-12 CS teachers worldwide,” or the International Society for Technology in Education (ISTE) that helps “educators around the world use technology to solve tough problems in education” (CollegeBoard, 2017; CSTA, 2017; ISTE, 2017). Global computing companies including Google, Microsoft and Oracle sponsor organizations such as these while also developing their own programs and resources to promote

computing education (Apple, 2017; Google, 2017; Oracle, 2017). Amazon, Facebook, Google, Microsoft and Salesforce recently committed a total of \$300 million to bolster computer science efforts in the United States (Romm, 2017).

In looking to fill the job vacancies projected by the U.S. Bureau of Labor, it is critical to consider the demands from employers who identify problem solving, teamwork, and communication as key skills to succeeding in the computing workforce (Archer & Davison, 2008; Robles, 2012; Vivian, Falkner, Falkner, & Tarmazdi, 2016). Using a five point scale with five being “extremely important” and one being “not important,” the 2013 National Association of Colleges and Employers survey reported that employers rated Communication skills (4.63 average), ability to work in a team (4.6), problem-solving skills (4.51), planning and organizational skills (4.46), ability to obtain and process information (4.43), and ability to analyze quantitative data (4.3) as the most desired characteristics in employees (Ardis et al., 2014; NACE, 2016).

In considering workforce vacancies and needs, it becomes incumbent upon postsecondary computing educators to address these demands. The academic and technical community has stepped up to the demands by specifically addressing problem-solving skills as a key element in the seminal and joint 2013 Association for Computing Machinery (ACM)/Institute for Electronics and Electrical Engineers (IEEE) Curriculum Guidelines for Undergraduate Degree Programs in Computer Science,

Graduates need to understand how to apply the knowledge they have gained to solve real problems, not just write code and move bits. They should be able to design and improve a system based on a quantitative and qualitative assessment of its functionality, usability and performance. They should realize that there are

multiple solutions to a given problem and that selecting among them is not a purely technical activity, as these solutions will have a real impact on people's lives. Graduates also should be able to communicate their solution to others, including why and how a solution solves the problem and what assumptions were made. (Sahami et al., 2013)

Background and Rationale

Despite the rapidly emerging technologies of the past six decades, in comparison with other scientific and engineering fields like physics, the domain of computing is relatively young, with the first Department of Computer Sciences in the United States being established at Purdue University in October 1962. These beginnings included courses in programming, focused on graduate education that eventually evolved into an undergraduate degree in 1967. Although many colleges were offering courses in the 1950s that enabled students to make effective use of computers, only three computer-related degree programs existed in North America during the 1960s: computer science, electrical engineering, and information systems (Gupta, 2007; Shackelford et al., 2006). Early on, academics debated on where computing was situated within the academy, with domains ranging from mathematics to electrical engineering to its own emerging field (Gupta, 2007). Today, computing has clearly emerged as its own domain. The umbrella term of "computing" can be considered to encompass all computer-related programs of study, and as such is

any goal-oriented activity requiring, benefiting from, or creating computers. Thus, computing includes designing and building hardware and software systems for a wide range of purposes; processing, structuring, and managing various kinds of information; doing scientific studies using computers; making computer systems

behave intelligently; creating and using communications and entertainment media; finding and gathering information relevant to any particular purpose...

(Shackelford et al., 2006)

Given the growth and contributions of the computing domain, more research attention focused on the understandings of computing education is required. More specifically, given the applied nature of computing domains, research related to practices that inform on the nature of problem solving and collaborative learning is critical. Essentially, these understandings impact how students are prepared within the program and what it means for their transition into the workplace. Although limited in details on pedagogical approaches, the ACM and IEEE, in their flagship computing curriculum guidelines, have identified solving problems and teamwork skills as important for computing students.

Computing Education Research

While the ACM/IEEE guidelines focus only nominal attention on instructional approaches to problem solving and collaborative learning (via teamwork or other approaches), the signature computing education research (CER) conferences and publications devote significant attention to these specific areas and to computing education research as a whole. “CER seeks to gain deep understanding of multiple aspects of the teaching and learning processes of various topics in the computing curriculum, and to build generalizable evidence about problems in students’ learning and the efficacy of new teaching approaches to solve these problems” (Malmi et al., 2014). Notable computing education research venues, particularly from the ACM, include:

- ACM Transactions of Computing Education (TOCE) – “covers diverse aspects of computing education by publishing papers with a scholarly

- approach to teaching and learning, a broad appeal to educational practitioners, and a clear connection to student learning” (TOCE).
- Special Interest Group in Computer Science Education Technical Symposium (SIGCSE) – started in 1970 and currently drawing an average of 1,300 computing education faculty annually, SIGCSE “addresses problems common among educators working to develop, implement and/or evaluate computing programs, curricula, and courses” (SIGCSE).
 - Innovation and Technology in Computer Science Education (ITiCSE) Conference (ITICSE) – started in 1996, the global ITiCSE conference addresses current and urgent issues in computing education research (ITiCSE).
 - The International Computing Education Research (ICER) Conference – started in 2005, the ICER research conference focuses on “how the understanding of computation develops, and how to improve that understanding” (ICER).
 - Special Interest Group for IT Education Conference/Research in Information Technology (SIGITE/RIIT) – Started in 1999, the joint IT education and research conferences provide a “venue for the discussion and dissemination of IT education,’ and “articulate, promote, and dissemination of research related to IT” (SIGITE).

As the number of publications around CER has exploded over the last 40 years, so too has the breadth of topics examined. This has led to a number of works looking to present a taxonomy for CER publications according to the overarching aspects of content and process and

specifically to methodological quality (Randolph, Julnes, Bednarik, & Sutinen, 2007), research contributions and methodologies (Pears, Seidman, Eney, Kinnunen, & Malmi, 2005), publication venues (Joy, Sinclair, Sun, Sitthiworachart, & López-González, 2009), didactic foci (Kinnunen, Meisalo, & Malmi, 2010) and theoretical underpinnings (Malmi et al., 2014).

Regarding the theoretical underpinnings of CER, Malmi (2014) posed two essential questions: “(1) What are the main challenges that impede our students’ efforts to learn computing concepts, processes and/or practices? and, (2) How can we demonstrate or confirm that various teaching innovations actually improve our students’ learning process and learning results?” (p. 34). Theories are critical in addressing these questions as they (1) aid in designing and explaining empirical research, (2) aid in building hypothesis to make predictions for given settings, (3) provide a common terminology to be used in communication and discussion among researchers, (4) provide a structured set of lenses to approach, observe, study or interpret the target of investigation, (5) can be used as a safeguard against unscientific approaches, by explicating underlying assumptions and choices, thus setting the framework for research and (6) can be used to protect against attacks from dubious colleagues from other disciplines (Malmi, 2014; Niss, 2007).

The absence of theoretical frameworks for learning and cognition specific to computing education results in curriculum development and refinement that evolve based on instructor intuition as opposed to informed practices. This intuition, stemming from faculty serving as pedagogical content knowledge (PCK) experts and integrating their tacit knowledge, must be highly valued in instruction (Shulman, 1987). PCK “represents the blending of content and pedagogy into an understanding of how particular topics, problems, or issues are organized, represented, and adapted to the diverse interests and abilities of learners, and presented for

instruction” (Shulman, 1987, p. 8). As PCK experts and computing education practitioners, faculty hold a great deal of knowledge that they may never have attempted to articulate, yet this wisdom of practice, as identified by Shulman (1987), presents a “codifiable knowledge” that distinguishes the content specialist from the instructor as a domain specific educator (p. 12).

To this end, grounding existing computing education curriculum in a theoretical framework enables researchers to adjust faculty intuitions rooted in PCK to make them explicit and allow for continuous improvement, thereby promoting a cycle of research-based educational interventions that can be further tested (Martella, Nelson, Morgan, & Marchand-Martella, 2013). Bereiter (2014) echoed this idea with his thoughts around Principled Practical Knowledge (PPK) which may be informally defined as “know-how combined with “know-why,” but is “more precisely characterized as explanatorily coherent practical knowledge” (p. 5). Bereiter’s (2014) ideas of PPK look to combine best practices, evidence-based-practices and reflective practices to advance education in any domain. By combining and moving between theory and practice, domain area expert educators can provide a model behavior for students to advance their field in terms of strategies, methods and organizing concepts (Bereiter, 2014).

Yet theories, conceptual models and frameworks are predominantly absent from computing education work that builds on the practice reports and scholarship of learning that is inherent in computing education (Malmi et al., 2014). Malmi et al., (2014) in their analysis of theoretical background work that was used or extended in 308 publications in the leading computing education publications from 2005-2011, found that “there are no prevailing theoretical or technical works that are broadly applied across CER; about half the analyzed papers build on no previous theoretical work, but a considerable share of these are building their own theoretical constructions” (p. 27). While valuable and perhaps even critical in practical

considerations of day-to-day content delivery in the classroom, much of the existing educational research addresses questions of “how”. Malmi (2014) presented a compelling case for considering Pais, Stentoft, and Valero’s (2010) work grounded in the field of mathematics, that emphasized questions of ‘how’ versus questions of ‘why?’ For example, ‘how’ questions may examine how a new intervention, teaching method or tool improves student learning. Positive results based on empirical studies may support faculty claims regarding the methods or tools, but leave unattended an understanding of ‘why’ the learning actually occurred and why an intervention, teaching method or tool is effective (Malmi, 2014; Pais, Stentoft, & Valero, 2010). While general education theories along with theories from neighboring domains such as engineering and education research may prove useful as foundations, CER first needs to define a computing discipline-based understanding of its own resulting in its own theories (Malmi et al., 2014). This proves challenging as Malmi et al., (2014) also found that “almost half of the papers examined do not build on any theory, model, or framework, and 80% of the computing education papers in our data pool do not build on theoretical research from education” (p. 31).

While more than half of the publications did integrate some theory, model or framework, researchers primarily looked outside the field of CER, likely because the CER field has created so few theories, models and frameworks of its own, leading researchers to seek elsewhere to ground their work. Other fields included psychology, medicine, philosophy, linguistics, management, and systems theory. This diffuse base of theories may in fact prove a hindrance to CER, as mixed terminologies and varying ways of building arguments can lead to lack of cohesion in the computing education domain research area. Considering that CER researchers are PCK experts with a computing discipline as their academic foundation, applying theories in

the same manner as the disciplines from which they draw may prove difficult (Malmi et al., 2014).

In further considering the theories, models and frameworks that were present in the analytical survey, Malmi et al., (2014) reported the most common was constructivism or some of its sub theories including communal constructivism, constructionism, social constructivism, and situated learning. Curricular frameworks, pair programming as a model, Bloom's taxonomy and pedagogical patterns were also prevalent. Nearly two-thirds of the distinct theories, models and frameworks could be identified with a specific name such as Bloom's taxonomy or Kolb's experiential cycle (Malmi et al., 2014). Few theories were found to be originating in CER at all (23 of the 226), and approximately half of those were categorization schemes or taxonomies, indicating that some nominal theoretical work exists in CER, but a gap exists, leaving room for additional development in the field (Malmi et al., 2014).

Problem Statement

Problem solving, teamwork and communication are in demand from the computing workforce. Problem solving is also a central tenet of teaching and learning across disciplines and levels of understanding, and has been studied generally and across numerous domains such as physics, engineering and medicine as well as computing and information processing (J. Bransford, Sherwood, Vye, & Rieser, 1986; M. T. Chi, P. J. Feltovich, & R. Glaser, 1981; Elio & Scharf, 1990; Elstein, Shulman, & Sprafka, 1978; J. Greeno, 1978; J. G. Greeno, 2006; D. Jonassen, Strobel, & Lee, 2006; J. Larkin, J. McDermott, D. P. Simon, & H. A. Simon, 1980; Newell, Shaw, & Simon, 1958; D. P. Simon & H. A. Simon, 1978). The everyday computing problem solving scenarios of designing, building, securing, debugging and managing computer systems provide an opportunity to master problem-solving skills in a domain specific context.

Faculty, as computing domain experts and PCK experts, are educating incoming students on computing components or domain specific topics such as search algorithms, programming languages, databases and operating systems. This leads to the guiding research question: in what ways are computing faculty, as computing domain and PCK experts, preparing novice students to become expert domain problem solvers and effective team members?

The Nature of Problem Solving

Tasks that ask students to memorize facts or repeat already demonstrated procedures require one kind of student thinking; tasks that ask students to authentically engage in the disciplinary practices such as constructing arguments, modeling, and evidence-based reasoning to develop an understanding of disciplinary ideas demand another kind of thinking. (Tekkumru Kisa & Stein, 2015).

Considering the previously noted computing workforce demands of problem solving, teamwork and communication, the theories, methods and strategies for problem solving are critical in considering how problem-solving approaches are enacted in computing education to develop problem-solving skills that enable high-level thinking rather than simple memorization. In this regard, examination of problem-solving practices and the associated dynamics of collaborative learning in computing education will be critical in meeting the problem-solving and teamwork needs of the computing workforce, and mastering problem solving must be considered distinct from component skills or skills specific to a domain (Jonassen, 2000). The current study addressed this gap and contributes to an emerging theoretical framework that addresses the knowledge and skills of collaborative learning in the context of problem solving in post-

secondary computing education. What is more, the study also addressed notions of complex problem solving that are apparent across computing sub domains.

Problem solving defined. A traditional problem is composed of two critical attributes: (1) an unknown entity in some specific situation; also known as the difference between the current state and the goal state and (2) solving for the unknown must have some social, cultural or intellectual value (Jonassen, 2000).. This is distinct from the act of solving the problem which Martinez (1998) defined broadly as the “process of moving toward a goal when the path to the goal is uncertain” (p. 605). Chi and Glaser (1985) more specifically defined problem solving as performing a set of operations, identified through the problem constraints, on an initial state to reach a goal state. Anderson (2005) further elucidated problem solving as “any goal directed sequence of cognitive operations” that includes two essential attributes (p. 257). The first attribute is the most critical in solving the problem and requires that human problem solvers create internal mental models known as the problem space that is a multimodal representation of structural, procedural and reflective knowledge along with images and strategic knowledge (Anderson, 2005; Jonassen, 2000). The second critical attribute for the process of problem solving is the “activity based manipulation of the problem space,” either through internalized thinking or externalized physical manipulation (Jonassen, 2000). These definitions align with others summarizing that “in educational contexts, as in life in general, the problem-solving process can be characterized as a complex interaction of factual knowledge, cognitive and metacognitive strategies, experiences, belief systems, and social factors (Heppner & Krauskopf, 1987; Schoenfeld, 1983; Silver & Marshall, 1990; Taylor & Dionne, 2000).

Larkin, McDermott, Simon, & Simon, (1980) noted that people who are skilled at solving physics and engineering problems are often labeled as having “physical intuition.” Such people

appear to simply know how to proceed in solving problems quickly without much deliberate consideration about how to approach the problem (Larkin, McDermott, Simon, & Simon, 1980). This “physical intuition” might otherwise be noted as expertise in a domain. Reasonably, Larkin et al., (1980) suggested that the idea of intuitive problem solving that distinguishes novices from experts is not simply intuitive, but rather demands explanation. As such, novice problem-solving approaches and expert problem-solving approaches alongside the nature of the problem being solved, yield insights into the broad understandings of problem solving.

An expert is widely characterized as an individual with domain knowledge, experience and competence while a novice, in contrast, is described as an individual limited in these areas (M. T. Chi, P. J. Feltovich, et al., 1981; M. T. Chi, Glaser, & Rees, 1981; Heyworth, 1999). While it is well documented that experts outperform novices in the process of problem solving, a number of studies supported the idea that qualitative differences exist in how experts and novices organize and use their domain knowledge when solving problems (Chi & Glaser, 1985; Elio & Scharf, 1990; Jonassen, 1997). MacLellan, Langley and Walker (2012), echoed this, citing that as human beings, we exercise a variety of strategies to solve complex problems, and this serves as a hallmark of human intelligence (MacLellan, Langley, & Walker, 2012). Taylor and Dionne (2000) supported this with their views that “problem solvers manage this complex dynamic through the application of problem-solving strategies: goal-directed procedures that are intentionally evoked to monitor, direct, or evaluate problem-solving activity” (p. 413).

Key among these qualitative differences, is that expert problem solvers have more developed schemas, or knowledge bases from which they are able to draw, while novices do not. Therefore, experts are able to recognize a problem as belonging to particular class of problem with specific solutions. This allows the expert problem solver to approach the problem in a

forward working process, by using their experience to recognize the problem state, analyze the problem, produce pertinent information and apply an appropriate operator (Larkin et al., 1980; Simon & Simon, 1978). Novices, lacking these problem schemas are less able to recognize the problem states and must resort to general problem-solving strategies known as backward working processes or backward chaining. These backward-working approaches are rooted in information processing systems such as Newell, Shaw and Simon's (1958) General Problem Solver (GPS) where subgoals were identified and addressed in an iterative fashion until the goal state was met (Jonassen, 1997; Sweller, 1988).

Critical to examining the processes for problem solving is first understanding the nature of the problem being solved. This can be broadly categorized as well-structured or ill-structured problems, but is better yet considered as a continuum of decontextualized problems with clear and singular solutions to highly contextualized problems with multiple solutions (Jonassen, 1997). Ill-structured problems include one or more aspects that are not well specified, the problem descriptions are not clear or well defined and the problem statement does not include the information necessary for solving the problem. The solutions are not predictable, nor convergent, and many alternative solutions may exist (Chi & Glaser, 1985; Jonassen, 1997). Well-structured problems, in contrast, require the application of a set number of concepts, rules and principles to a predetermined problem situation. Well-structured problems are also known as transformation problems, consisting of a well-defined initial state and a known goal state to which a constrained set of logical operators is applied. Because they are constrained to the knowledge base presented alongside the problem, well-structured problems are typically found in academic settings, often included at the end of textbook chapters (Greeno, 1978; Jonassen, 1997). While the novice approaches of backward working are typically effective for well-

structured problems, this process proves ineffective for ill-structured problems. Distinguishing between ill-structured and well-structured problems is important in considering that the skills for solving well-structured, constrained problems have limited relevance and transferability to solving the ill-structured problems situated in everyday contexts (Jonassen, 1997).

Social Constructivist Theory to Examine Problem Solving

Again, noting the ACM/IEEE guidelines and publication research alongside the workforce demands for problem solving, teamwork and communication in the computing domain, social or collaborative learning in an academic program becomes paramount to preparing students for successful collaborative problem solving in their profession. Asserting that problem solving is conducted as part of a group in social or peer-learning setting, rather than in solitude, provided a context for further considering problem solving. Social learning in the form of collaboration in the classroom may occur through interactions among students, among teachers, and between students and teachers (Hogan, 1999, 2002; Schraw, Crippen, & Hartley, 2006). According to Webb (1989) interaction among peers in academic group work involves small groups of students who are given material to learn or a problem to solve. The expectations, and typically instructions as well, are that all students are to master the material while helping each other to learn the material or solve the problem. All group members, as part of the team, share the same information regarding the material or problem with no specific group roles identified. The resulting outcome of group work may be a group product or a demonstration of individual mastery of the material (Webb, 1989).

Given the constructs of social learning in the form of collaboration as noted above, social constructivism presented a compelling theory through which to further examine problem solving. Its appropriateness to the computing domain was echoed in Malmi et al.'s (2014) findings of

social constructivism as a predominant theory in computing education research. The social constructivist principle that knowledge results from meaning-making through social interaction indicates that education and learning is essentially a social process. This quality is realized in the degree in which individuals collaborate and form a community group. Lev Vygotsky, the founding father of social constructivism, believed that social interaction was the cornerstone of learning. Vygotsky (1930-34/1978) is credited with the conception that psychological functions such as attention, memory and cognition are socially distributed in a community of learners rather than properties of the individual mind (Kozulin, 2003). Learner construction of knowledge becomes the product of social interaction, interpretation and understanding (L. S. Vygotsky, 1962). Thus, creation of knowledge becomes inextricably intertwined with the social environment in which it is formed, and learning becomes a process of active knowledge construction within and from those social activities and practices (Adams, 2006).

Dewey (1938) also embraced the idea of experiential education as a social paradigm, suggesting that the quality of the educative experience is realized to the degree in which individuals form a community group for learning. In explicitly identifying education as a social and collaborative process, Dewey (1938) valued the role of the instructor, previously noted as the PCK expert, in leading group activities. The principle that development of experience comes about through collaboration and interaction suggests that education is essentially a social process.

Potosky (2014) underscored the value of communication in co-creating understanding. In this sense, communication is not the one-way transmission of ideas from a teacher to a learner, as is often the case in a didactic, teacher-centered classroom experience, but rather a social interaction that breaks down the historical hierarchical power structures of a traditional

classroom setting. The social construction of meaning through hopeful inquiry in the form of problem solving, implies a dialogic view where meanings are co-constructed through collaborative communication (Cooperrider, Barrett, & Srivastva, 1995; Gergen, 1985; Potosky, 2014). Often, these experiences may be posed as in the nature of problem-based learning, with an uncertain outcome, where students experience uncertainty in knowing what to do next, and leads to “finding answers through collaborative inquiry with fellow learners. This activity is a profound shift from dependence on available expertise and pride in self-learning to learning with and from others, disclosing doubts and admitting ignorance” (Potosky, 2014).

As such, social constructivism provided a framework to theorize the wisdom of practice of computing educators, bringing transparency to the tacit, underlying processes already embedded in the curriculum and the classrooms (Shulman, 1987). In this sense, considering collaborative learning in the context of instructor-student interaction and peer-to-peer student interactions served as a central point of examination in further understanding computing education problem solving through social constructivist theory.

Purpose of the Study and Research Questions

In light of the above discussion, this naturalistic exploratory case study examined (a) the kinds of problems that are posed/used by undergraduate computing faculty in their process of teaching and learning in computing education; (b) the instructional approaches undergraduate computing faculty use as a mechanism to facilitate teaching and learning problem solving in computing education; and, (c) the aspects of collaborative learning that are realized in undergraduate computing faculty instructional approaches to problem solving. The above aims were examined within the context of problem-solving approaches in first-year and second-year computing courses across three computing education programs: computer science, information

sciences and technologies and software engineering. Instructional classroom and laboratory observations of computing courses, in-depth faculty interviews, and course artifacts served as the main sources of data. This study was guided by social constructivist theory which undergirds social learning. In addition, problem-solving literature alongside Jonassen's (2000) characterization of problems and problem-solving typology guided understanding of the kinds of problems posed/used in computing education and the approaches to solving those problems.

The following research questions guided this study:

- R1. What kinds of problems are posed by computing faculty in their process of teaching and learning in first and second-year computing courses?
- R2. What instructional approaches do undergraduate computing faculty use as a mechanism to facilitate teaching and learning problem solving in first and second-year computing courses?
- R3. In what ways are aspects of collaborative learning realized in undergraduate computing faculty instructional approaches to problem solving in first and second-year computing courses?

Significance of the Study

The evolution and proliferation of computer-driven solutions to everyday problems over the past six decades has yielded a diversity of computing programs. This diversity of programs addresses the demands of the 21st century workforce, yet also yields a lack of clarity. Efforts by the seminal bodies of the ACM and IEEE to clarify the commonalities among and distinctions between the programs, focus on the domains in terms of outcomes, bodies of knowledge and problem spaces. Through computing education research domains, faculty attend to the instructional approaches that govern the recommendations set forth in domain specific curricular

guidelines. Given the workforce demands for problem solving, teamwork and communication in the computing domain and the prevalence of constructivist learning theories, models and frameworks in the existing computing education research, asserting a problem-solving typology alongside a social constructivist theoretical framework to examine faculty perceptions of and approaches to problem solving in a collaborative learning context across computing subdomains was advantageous in expanding the knowledge-base to theorize problem solving and collaborative learning in computing education research.

Considering the call for problem solving, teamwork and communication by the workforce, this study was significant in addressing how computing educators prepare students for those demands. Of particular importance was considering the differences of how novices and experts approach problems, and how the computing faculty, situated as domain specific experts and PCK experts, are preparing students who, as novice learners generally lack an established knowledge-base.

In summary, this study was significant because understandings of how computing education faculty perceive and approach problem solving and collaborative learning in the computing education domain may yield insight on practices that either promote or hinder learning and understanding in computing education. These understandings are specifically important to extending the codifiable base of knowledge in computing education research due to the nature of this domain of learning which hinges on application of knowledge and skills in the workforce. Given the limited practice of theoretical frameworks in computing education research, a secondary goal of this study was to contribute towards an emerging computing education research framework that informs on the enactment of collaborative learning within the context of problem solving in computing education.

Chapter 2

Review of the Literature

The Nature of Problem Solving

Modern-day models of student-centered learning such as problem-based learning recommend instructional strategies such as authentic cases, coaching and scaffolding all while integrating implicit problem-solving outcomes (Jonassen, 2000). Jonassen (2000) distinguished problem-solving learning outcomes from concept, or domain-content learning and asserted that all problems are not the same and as such cannot be addressed in the same manner as component or domain-skill learning. Because component skill mastery is insufficient in solving non-routine, or ill-structured problems that comprise the bulk of workforce and daily life problems, presenting students with well-structured problems leaves them unprepared to function in a professional setting following their formal education (Jonassen, 2000). Underlying these ideas is the central tenet that human beings exercise a variety of strategies to solve complex ill-structured and well-structured problems, and this serves as a hallmark of human intelligence. According to Taylor and Dionne (2000), “problem solvers manage this complex dynamic through the application of problem-solving strategies: goal-directed procedures that are intentionally evoked to monitor, direct, or evaluate problem-solving activity” (p. 413). Yet, these strategies and processes vary based on level of expertise in solving domain problems. This is not to suggest that expert problem-solving approaches are of sole importance. But rather, while the objective may be to reach an expert state, we must consider that at specific points in time, only novice approaches may be available or that novice approaches may be the most appropriate or effective.

The broad categories of (1) ill-structured problems rooted in constructivism and situated cognition, and (2) well-structured problems, rooted in information processing and relying on

general or novice problem-solving skills such as mean-ends analysis provided a foundation for examining problems that are solved in the computing domain. Distinguishing between ill-structured and well-structured problems is important when considering that the skills for solving well-structured, constrained problems have limited relevance and transferability to solving the ill-structured problems situated in everyday contexts (Jonassen, 1997). As such, the nature of problem solving as it relates to the broadly identified ill-structured and well-structured problems, along with the novice and expert strategies for solving them, presented a foundation for examining collaborative learning in the context of problem solving in computing education domain.

Characteristics of Ill-Structured Problems and Well-Structured Problems

Important to considering whether or not a problem is ill-structured or well-structured is the idea of problem representation, or more specifically, examining the concepts and relations of a problem, isolating the major factors causing the problem along with the constraints, and recognizing various perspectives (Voss & Post, 1988). Gick (1986) posited that ill-structured problems lack a well-specified goal and given information and require more understanding than problems with a well-defined goal. Yet also of central importance is the individual's perspective, as a problem may appear ill defined due to a lack of understanding on the part of the solver leading one to also consider the importance of the novice and expert problem-solver perspectives (Gick, 1986).

Jonassen (1997; 2000) contended that ill-structured problems comprise the bulk of workforce and daily life problems. Because ill-structured problems are characteristically emergent dilemmas situated in everyday practice, they are typically more interesting and meaningful to learners. When situated within a domain context, ill-structured problems include

one or more aspects that are not well specified, the problem descriptions are not clear or well defined and the problem statement does not include the information necessary for solving the problem. The solutions are not predictable, nor convergent, and many alternative solutions may exist (Chi & Glaser, 1985; Jonassen, 1997).

Well-structured problems, in contrast, are commonly experienced in schools and universities, often included at the end of textbook chapters. These problems necessitate the application of a set number of concepts, rules and principles to a predetermined problem situation. Consisting of a well-defined initial state, a known goal state, and a bounded set of logical operators, these problems are also known as transformation problems and are constrained to the knowledge base presented alongside the problem (Greeno, 1978; Jonassen, 1997). The key distinguishing characteristics of ill-structured and well-structured problems are summarized in Table 1.

Table 1
Ill-Structured and Well-Structured Problem Characteristics (Jonassen, 2000).

Well-structured Problems	Ill-Structured Problems
<ul style="list-style-type: none"> • Present all elements of the problem to the learner • Are presented to problem solvers as having a probable solution with parameters identified in the problem statement • Engage the application of a set of rules and principles organized in a predictive and prescriptive manner with well-defined and bound parameters • Require a limited number of well-structured rules be applied in an organized and prescriptive manner • Have correct and convergent answers • Identifiable solutions where the relationship between decision choices and problem states is known or probabilistic (Wood, 1983) 	<ul style="list-style-type: none"> • Encountered more often in everyday life and professional practice • Not constrained by classroom content domain • Unpredictable solutions • May require integration of several domains • Possess unknown elements (Wood, 1983) • Have vaguely defined or unclear goals (Voss & Post, 1988) • Possess multiple solutions, solution paths or no solutions (Kitchner, 1983) • Possess criteria for evaluating solutions leading to uncertainty about relevant concepts, rules and principles • Possess parameters for manipulation • Have no prototypic cases because problem elements are contextually

<ul style="list-style-type: none"> • Have a preferred and prescribed solution process 	<p>important and different (Spiro, Coulson, Feltovich, & Anderson, 1988; Spiro et al., 1987)</p> <ul style="list-style-type: none"> • Present uncertainty about necessary rules, concepts, principles and organization • Possess inconsistent concepts, rules and principles relationships between cases • Offer no general rules or principles for describing or predicting most cases • Have no explicit means for determining appropriate action • Require learners to make judgments and express beliefs about the problem
--	---

These problem characteristics are of key importance as the processes for problem solutions can be derived based on the problem representation and are influenced by the solver's level of expertise.

Processes for Solving Ill-Structured Problems and Well-Structured Problems

The problem solver's goal is to reach a solution for the presented problem. This solution may be either convergent, with a single solution, or divergent, with many possible solutions. Regardless of whether the problem type is ill-structured or well-structured, a critical attribute is that the solution is not readily apparent or specified in the problem statement. Thus, the solver must identify the nature of the problem, an acceptable solution and a process to reach that solution. Research has demonstrated that performance in solving well-defined problems is distinct from performance in solving ill-defined problems, engaging independent sets of epistemic beliefs and skills (Schraw, Dunkle, & Bendixen, 1995). As such, each will be examined in more detail to present the possible perspectives that may be enacted and experienced in a computing classroom environment by both faculty and students.

Ill-structured problem-solving processes. Ill-structured problem-solving processes may conceptually be considered as a design process rather than a systematic search for a problem

solution (Jonassen, 1997; Schön, 1990). With an ill-structured problem, the solver must (1) frame the problem, (2) identify and clarify alternative perspectives of the problem, (3) generate possible problem solutions, (4) assess the viability of alternative solutions (5) monitor the problem space and solution options and (6) implement, monitor and adapt the solution (Jonassen, 1997).

Framing the problem. In framing the problem, the problem solver must first determine if a problem really exists by examining the context from which the problem emerged. This is important because while many problems appear to have an unknown, there is in fact a hidden known, and therefore no problem. Then, the nature of the problem must be determined and an appropriate problem space identified from all the differing possibilities. Fogler, LeBlanc and Rizzo (1995) demonstrated this via an example where upper floor hotel guests complained about slow elevators. The real problem of taking the guests' minds off the wait time was masked by searching for a solution to speeding up the elevators themselves. By installing mirrors in front of the elevators, management eliminated the customer complaints, thereby solving the problem from the hotel's perspective (Fogler, LeBlanc, & Rizzo, 1995). This exemplified the idea that ill-structured problems are domain or context dependent because the solver must consider them as realistic situations rather than academically presented information that constrains the problem (Bransford, 1993; Jonassen, 1997). As such, Bransford (1993) suggested that experience, rather than predefined problems with convergent solutions is required for meaningful problem solving.

In contrast to well-structured problems where solvers recognize and classify the problems, ill-structured problems necessitate the solver assembling conceptual and relevant domain knowledge from memory rather than from the academic materials presented as part of the lesson and focusing on a constrained set of rules (Jonassen, 1997; Voss & Post, 1988). This

domain knowledge develops from experience in solving domain problems. As such, experts have been shown to solve problems in one quarter the time it takes novices while also making significantly fewer errors (Simon & Simon, 1978).

Ill-structured problems require the solver to examine all possible problem causes which may be constrained by incomplete, inaccurate or ambiguous details alongside divergent goals. Additionally, ill-structured problems cannot be solved by applying a prescribed ruleset. They require that the solver construct a problem space containing all possible problem states along with the problem operators and constraints, which may be contextual factors (Jonassen, 1997; Voss & Post, 1988). An important metacognitive strategy is to reflect on what is known about a problem domain to consider if previous accounts of the situation have been encountered or read about, or to consider what resources might be available to provide further information. Jonassen (1997) posited that solvers must learn how to relate the aspects of the problem to their own knowledge and that modeling or using a set of procedural prompts by the instructor to review what is already known about the problem can be useful. This will be further explored as a fundamental tenet of social constructivism through scaffolding.

Identifying and clarifying alternative perspectives of the problem. Ill-structured problems require the solver to construct multiple problem spaces, and then determine which problem solution is most useful for solving the problem. This involves identifying the differing goals and problems that may be involved, leading to divergent solutions.

Generating possible problem solutions. Here, solution states are identified through analysis of possible causes, thereby focusing the solution process on addressing the causes. This step typically requires a forward working approach to the problem that is consistent with expert problem-solving methods. The characteristics and constraints represented in the problem are

critical in accomplishing this. As such, because ill-structured problems have multiple representations based on the problem-solver's perceptions, and consistent with constructivist theory, grounding multiple solutions exist. Solvers must evaluate the possible solutions before selecting solutions that are known and within reach. This evaluation process, requiring prior experiences from which to draw, along with creativity, can further constrain the solution space. Overall, this process of generating possible solutions requires epistemic knowledge that is critical in synthesizing the viability of the solution alternatives as solvers are building their own mental models of the problem and solution space (Jonassen, 1997).

Assessing the viability of alternative solutions. In assessing the viability of alternative solutions, the solver must be able to present a preferred solution. This requires an understanding of contrasting views in order to agree or disagree with the alternatives. This results in the solver constructing his or her own arguments supporting a particular solution by selecting a position, reflecting on how they arrived at the position and providing justification for the position. Evidence gathering throughout serves as a key element of the process. By debating the viability of the solutions, either internally, or among others in a group as part of a collaborative learning experience, the problem solvers iteratively refine their own problem representations in finalizing a best solution (Jonassen, 1997). This will also be further examined in the context of social constructivist theory undergirding this study.

Monitoring the problem space and solution options. While well-structured problem solving relies on comprehension strategies, ill-structured problem solving requires engaging metacognitive structures where solvers integrate planning strategies as well as monitoring the epistemic nature of the various solutions (Jonassen, 1997; Kitchner, 1983). This involves problem solvers knowing about their own limits of knowing, and as such, the solver must first

determine if the given problem is solvable and if there are existing processes for solving it. This monitoring process is not a separate reflective process about what they and others know and what that means, but rather one that occurs in parallel with the previously outlined steps.

Jonassen (1997) contended that ill-structured problem solving typically ends at this stage, as solutions to highly complex and inaccessible problems often cannot be executed. This is not to suggest that ill-structured problems where solutions cannot be implemented are not purposeful, as working through the previous steps still engages learners and solvers in higher-order, problem-solving learning. Yet, real-world contexts necessitate applying solutions as in the final step (Jonassen, 1997).

Implementing, monitoring and adapting the solution. Because ill-structured problems do not result in a single, correct solution, any implemented solution must be monitored for performance. Monitoring involves reflecting on the scenario to determine if the solution is acceptable to everyone involved and if similar results could be accomplished more efficiently or effectively. As a result, both the problem solution and the solver's mental model may be adapted, including making inferences about the viability of such a solution for other problems. As problems are rarely solved through a single attempt, this reflective monitoring and adapting becomes an iterative process leading to better integrated mental models of the problem space for the solver (Jonassen, 1997).

Well-structured problem-solving processes. Well-defined problem-solving processes are grounded in the information processing models of problem solving. Newell, Shaw and Simon's (1958) seminal work on the General Problem Solver (GPS) systemically implemented problem solving by understanding the processes followed by searching through a problem space with the goal of transforming the initial state into one that satisfied the goal of the problem.

Further delineated by the work on the IDEAL (identifying, defining, exploring, acting and looking back) problem solver (Bransford & Stein, 1984), these stages may be broadly considered as, (1) defining and representing the problem, (2) exploring possible solution strategies, (3) implementing strategies and reflecting back to evaluate the effects of the solutions (Gick, 1986; Jonassen, 1997). While there appears to be some overlap with the six stages of approaching an ill-defined problem, the three stages of approaching a well-defined problem are inherently grounded in the previously described problem characteristics, and as such are examined within that context in the following sections.

Defining and representing the problem. Understanding the task by extracting the goal from the problem statement is critical in this step. Questions such as “What do I need to produce; what does an appropriate solution look like” are asked. Problem solvers also isolate specific problem attributes and they accomplish this by mentally decomposing the problem statement and mapping it onto their prior knowledge. It is during this stage that the solver searches through the mental problem space and accesses prior domain specific knowledge while generating hypotheses and possible solutions, extracting the given state and the goal state to understand the problem (Gick, 1986; Greeno, 1977). They further create a problem space that includes their understandings of the problem attributes, the goals and the possible solutions and strategies for solving the problem (Jonassen, 1997; Polson & Jeffries, 2014). In contrast to an ill-structured problem, these problem representations are a response to the problem scenario as it is presented, rather than one that emerges from the context of the problem or one that is generated independently by the problem-solver, as with the elevator and mirror example.

In representing the problem, links to existing or prior knowledge are activated. During this schema activation, if a problem solver possesses a complete schema for that type of problem,

then the problem statement is mapped to their problem schema, enabling them to move directly to the implementation phase (Gick, 1986; Jonassen, 1997). Experts are able to recognize more problem states that are associated with given solutions, requiring very little searching through the problem space and allowing them to move directly to a solution stage, where general strategies such as decomposing the problem into subproblems may be integrated or domain specific solutions may be generated (Chi & Glaser, 1985; Gick, 1986; Jonassen, 1997; Sweller, 1988). As domain knowledge is acquired, problem solving becomes more schema driven and is accompanied by domain specific strategies (Gick, 1986). Novices, with fewer problem states in memory, may be unable to recognize the problem types, and are forced to resort to general backward working problem-solving strategies such as means-end-analysis.

Exploring possible solution strategies. In solving a well-structured problem, the solver must be able to suitably represent the problem which necessitates searching for possible solution strategies. Because novices lack experience to effectively accomplish this, strategies must be implemented. Many strategies are in fact heuristics, or general suggestions or techniques which help problem solvers understand or solve a problem. Heuristics can also be considered processes for solving a problem and many heuristics exist in various domains. Examples of mathematical heuristic strategies include the “fewer variables,” “calculating special cases,” and “argument by contradiction” strategies. (Schoenfeld, 1980). Within the context of problem-solving skills and approaches, Schoenfeld (1980) expressed that while domain experts are likely to use heuristic strategies to solve problems, novices are much less likely to do so (p. 796). This stands to reason as using a problem-solving strategy or heuristic in-and-of-itself requires a significant skillset from the solver, contradicting the nature of a novice who lacks an established set of strategies and problem schemas. To mediate this, key strategies can be engaged to facilitate problem

solving including (a) recall analogical reasoning, (b) means-end analysis and (c) decomposing to subproblems and (d) generate and test.

Recall analogical reasoning. When considering a problem, a natural first response is to question whether or not a similar problem has been experienced. This requires the problem solver to recognize similarities between the current problem state and previous problem states along with recollection of the solution. Gick and Holyoak (1980) demonstrated that when asked to consider a previously known medical problem using prompts, solvers were able to solve a problem as opposed to being presented with a lack of prompts which resulted in a decreased mapping of solutions. This approach of using prompts for recalling analogical problems is a natural first step for novices in solving well-structured problems. Yet if this proves unsuccessful, a backward working approach of means-end-analysis may prove useful.

Means-end analysis. Means-end-analysis is a backward working approach that focuses on reducing the difference between the problem's goal state and the current state by applying a set of operators (Gick, 1986; Jonassen, 1997). First introduced in the Newell et al. (1958) GPS information processing system, the problem solver isolates the goals of the problem, and uses a systematic approach to select the methods or means to reach each of the goals. This backward-working approach is used to form new problems, where the problem is worked backward from the goal. This is a recursive process whereby the solver begins with the most important difference and selects a means to reduce the difference to the current state, proceeded by the next most important difference, until a complete solution is reached. While forward chaining, commonly applied by experts in solving ill-structured problems, uses the approach that if 'a implies b' is known, and 'a implies c' is desired, it is sufficient to prove that 'b implies c,' backward chaining runs opposite from this with the idea that knowing 'b implies c' and 'desiring

a implies c' results in 'a implies b' as an entirely new problem. In this manner, the system starts with the goal, and from that goal, subgoals are identified through a process of inference. This is an inference because the initial goal establishes the direction of problem examination and subgoal development until the condition is met with the lowest subgoal.

To further clarify, in backward chaining, the system sets a goal state and works in the backward direction. Conditions and subgoals are then established to reach the goal state of true. The system checks to determine if the goal state matches with the established states, and if this is true, then the goal is identified as one solution. For example, if the goal is to switch on a sprinkler system, one subgoal may be to identify if it is hot and smoky and another subgoal would be to identify that if the alarm is beeping, then it is smoky. The final goal is that if both these subgoals are met, then the sprinklers should be activated (Pathak, 2017). This system is very effective when the goal state is clear and well defined such as designing a system to always execute a particular action.

Chi et al. (1981) demonstrated the varying approaches of novices and experts in regards to means-end-analysis when they asked participants to categorize physics problems and associated solution using verbal protocols. Novices organized their knowledge around the problem along the surface features of the problem described in the problem statement (e.g., incline-plane, block, pulley) and generated a solution using simple associations. As a result, novices resorted to general problem-solving strategies such as means-end analysis because they were unable to group problem states based on their similarities. In contrast, experts organized a problem-solving solution using deeper problem statement aspects along with abstract physics principles and their associated conditions and constraints (Elio & Scharf, 1990; Sweller, 1988).

Experts, who possessed domain specific schemas that distinguish between the problem state and the goal state, likely categorized the problems according to their schemas.

Simon and Simon (1978) and Larkin et al. (1980) reported similar results with novices resorting to general problem-solving approaches such as means-end-analysis when posing problems to physics students. When solving for a desired quantity in a physics problem, novices typically generated an equation to solve the problem. Using a backward-working approach as with backward chaining, if the equation contained an unknown variable, novices selected another equation to solve for this variable. In finally solving the problem, the procedure was reversed with a forward-working approach. In contrast, physics experts presented with a problem were inclined to use the known problem statement variables to only work forward and generate equations producing additional known quantities. Expert physicists, grounded by their previous experience, used a forward-working approach whereby they recognized the problem and the problem state, qualitatively analyzed the problem, produced pertinent information about the problem and applied an appropriate operator. Experts eliminated the backward working sequence implemented by the novices in reaching the goal (Larkin et al., 1980; Simon & Simon, 1978).

Physics novices also tended to read the problem statement and quickly suggest solutions, whereas experts first analyzed the problem qualitatively, specifically generating additional, but undeclared, pertinent information about the problem situation (Larkin et al., 1980; Larkin & Reif, 1979). Larkin et al. (1980) described this as “knowledge development” that led to problem clarification along with appropriate solutions. Again, in contrast to expert problem solving, Chi, Glaser, and Rees (1981) identified that even when novices allocated additional time for problem analysis, it resulted in faulty inferences or failure to produce any inferences at all. Similar results

have been reported in a social science domain (M. T. Chi, R. Glaser, et al., 1981; Voss, Greene, Post, & Penner, 1983).

Decomposing to subproblems. Closely related to the means-end-analysis approach is the often-recommended general strategy of breaking down the problem into subproblems or subgoals, where the solver repeatedly divides the problem into smaller problems until they are small enough that a solution becomes apparent. In this decomposition strategy, the solver can reduce the number of solution paths if a known subgoal state with fewer steps is recognized. A limiting factor in this decomposition approach, like other general strategies, is that the solver must have a thorough understanding of the problem-solving domain (Jonassen, 1997).

Generate and test. The final, least structured and least effective process for solving well-structured problems is a generate-and-test method. In this approach, solutions are brainstormed and evaluated. Perhaps the most common method for novice problem solvers, this approach relies heavily on the abilities of the person generating the possible solutions. This may include a set of random solutions to solving the problem or some more systematic approach such as sequentially moving through a series of options in a particular order. Each solution must be tested to determine its effectiveness, and if the test does not yield a correct solution, the next option is generated and tested in an iterative fashion. While an exhaustive generate-and-test method may prove effective for smaller and simpler problems, its usefulness is severely limited in more complex problems. Regardless, this approach is not grounded in domain-specific knowledge.

Implementing strategies and reflecting back to evaluate the effects of the solutions.

This overall iterative process of testing solutions results in the solution either working, in which case the problem is solved, or the solution failing to work in which case the process is adjusted

or a new hypothesis is generated. Critical to this stage is coaching, as generating new hypotheses from the clues of failed attempts is particularly challenging for inexperienced problem solvers (Jonassen, 1997). Again, coaching will be further examined in regards to problem solving in the context of social constructivist theory.

Problem Solving Through Social Constructivist Theory

While well-structured problems are rooted in generalizable information processing theory approaches that can be transferred across domains (Newell et al., 1958), the ill-structured problems of everyday life share assumptions of constructivism that argue for domain specificity and authentic contexts (Jonassen, 2000). Given the workforce call for problem solving and teamwork specifically in the computing domain, social constructivist learning theory presented an opportunity to examine problem solving in a framework grounded in collaborative meaning making.

Theoretical tenets of social constructivism for problem solving. Objectivist theories assume a structured world, where knowledge that is external to the learner can be mapped from the instructor to the learner, and the goal is for the learner to reproduce the knowledge imparted from the instructor. Direct instruction, or teacher-centered methods where the teacher directly instructs students while maintaining control of the space, sequence and content of the lesson, have been demonstrated as an effective means of teaching factual content (Palincsar, 1998). However, didactic methods prove less effective in developing higher order cognitive skills such as reasoning and problem solving as well as preparing students with effective strategies for use in unique situations, such as those presented in ill-structured problem solving. Meaning making, in the form of schema development and effective use of heuristics as the representations of knowledge in memory are characteristics seen as the key underlying aspects of problem solving,

and as such constructivism presented an appropriate theoretical framework by which to examine collaborative problem solving.

Theories of constructivism serve as an alternative to traditional didactic teaching methods with the central tenet of the classroom environment moving beyond the traditional teacher-centered transmission of information toward an environment where students and instructors work together to build meaning, understanding, and relevant practice (D. Jonassen, Davidson, Collins, Campbell, & Haag, 1995). Constructivism is grounded in the idea that we as humans have no access to a single objective reality, but rather are constructing our own version of reality while at the same time transforming it and ourselves (Fosnot & Perry, 1996). Constructivism's assumptions around learning and knowledge lie in the conception that knowledge is constructed by learners in their minds as they make sense of their experiences rather than knowledge being a function of what someone else proclaims as real (Driscoll, 2000; D. Jonassen et al., 1995; Woo & Reeves, 2007). Learners shift from being passive recipients of information to active knowledge constructors. As such, constructivist theories hold knowledge as a function of how learners make meaning of their experiences rather than what someone imparts as truth. This necessitates that the learning environment move beyond learners simply listening and mirroring facts presented by an instructor, to actively participating with and engaging in the environment to create knowledge that is usable in new and differing situations rather than being inert (D. Jonassen et al., 1995). Thus, constructivist learning seeks to present an environment where learners must examine processes for thinking and learning; assemble, record, study, analyze data; present and test hypotheses; reflect on prior perceptions and knowledge; and make their own meaning (Crotty, 1994; D. Jonassen et al., 1995).

Social constructivist problem solving through collaboration. Social constructivism

provided a framework to theorize the wisdom of practice of computing educators, bringing transparency to the tacit, underlying processes already embedded in the curriculum and the classrooms (Shulman, 1987). Social constructivist theorists see learning as a social, dialogical process where communities of practitioners socially negotiate the meaning of understanding. This social negotiation of learning occurs through group conversation where knowledge is shared by the community of practice rather than being the privilege of a single individual (D. Jonassen et al., 1995). Applying this meaning making in a real-world practice is critical to constructivist theories, and as such, one goal of constructivism is to embed learners in a community of practitioners who are solving real-world problems, previously presented as ill-structured problems with no single correct solution (D. Jonassen et al., 1995; Lave & Wenger, 1991).

With social constructivism, ideas are constructed through collaboration and interaction among the community that includes the instructor and peers and may occur through interactions among students, among teachers, and between students and teachers (Hogan, 1999, 2002; Powell & Kalina, 2009; Schraw et al., 2006). Collaboration and social interaction are incorporated such that ideas are constructed from experiences to have a personal meaning for the student.

Lev Vygotsky, the founding father of social constructivism, believed that social interaction was the cornerstone of learning. Vygotsky (1930-34/1978) is credited with the conception that psychological functions such as attention, memory and cognition are socially distributed in a community of learners rather than properties of the individual mind (Kozulin, 2003). Learner construction of knowledge becomes the product of social interaction, interpretation and understanding (L. S. Vygotsky, 1962). Thus, creation of knowledge is inextricably intertwined with the social environment in which it is formed, and learning becomes a process of active knowledge construction within and from those social activities and practices

(Adams, 2006).

Social constructivism is rooted in the social interactions learners have, along with their personal critical thinking process and cooperative learning which serves as a fundamental aspect of deeper understanding (Vygotsky, 1930-1934/1978). This cooperative learning stems from students engaging with instructors as well as peers. “Learning awakens a variety of internal developmental processes that are able to operate only when the child is interacting with people in his environment and in cooperation with peers” (Vygotsky, 1930-1934/1978, p. 90).

Similarly, Dewey (1938), embraced the idea of experiential education as a social paradigm, suggesting that the quality of an educative experience is realized to the degree in which individuals form a community group for learning. This principle that development of experience comes about through interaction suggests that education and learning is essentially a social process. In explicitly identifying education as a social process, Dewey (1938) moved beyond the singular notion of peer-to-peer interactions to explicitly include the instructor as part of the collaborative effort.

It is absurd to exclude the teacher from membership in the group. As the most mature member of the group he has a peculiar responsibility for the conduct of the interactions and communications which are the very life of the group as a community.” (Dewey, 1938, p. 58)

Indeed, the educator was viewed as having a unique role as the individual responsible for selecting experiences and problems to stimulate new ways of observation and judgment to further expand experience. By constantly opening new fields of inquiry with new demands, the instructor, as part of the group, was responsible for presenting an educative experience to students. Dewey (1938) articulated the value of the instructor’s experience, previously noted as

PCK and content domain expertise. “When education is based upon experience and educative experience is seen to be a social process, the situation changes radically. The teacher loses the position of external boss or dictator but takes on that of leader of group activities” (Dewey, 1938, p. 59). Vygotsky (1930-1934/1978) predicated this idea by specifically noting that the role of the teacher should be that of a facilitator or a guide, rather than a director or dictator.

Potosky (2014) underscored the value of communication in co-creating understanding and further emphasized that communication is not the one-way transmission of ideas from a teacher to a learner, as is often the case in a didactic, teacher-centered classroom experience, but a social interaction that breaks down the historical hierarchical power structures of a traditional classroom setting. In this sense, the bi-directional process of communication is critical to the teacher-learner and learner-learner interactions where learning is an active process rather than passive reception on the part of the students (Potosky, 2014). Again, the social construction of meaning implies a dialogic view where meanings are co-constructed through communication and part of collaborative experiences (Cooperrider et al., 1995; Gergen, 1985; Potosky, 2014). Often, these experiences may be posed as in the nature of problem solving, with an unpredictable outcome, where students experience uncertainty in knowing what to do next, and which leads to “finding answers through collaborative inquiry with fellow learners. This activity is a profound shift from dependence on available expertise and pride in self-learning to learning with and from others, disclosing doubts and admitting ignorance” (Potosky, 2014, p. 53).

Through collaborative co-construction of knowledge, the learner internalizes knowledge at their own rate depending on their own individual experiences and individual mastery of new skills through solving problems. Vygotsky (1930-1934/1978) believed that internalization was more effective when integrated as part of a social interaction. By considering varying

perspectives on the content at hand, a key construct of problem solving, previously unconsidered opportunities can be opened for a student (Powell & Kalina, 2009). Instead of a prescriptive learning environment where the instructor controls the sequence and content of all instruction, a social constructivist learning environment presents a situation where learners engage in knowledge construction through collaborative activities that embed learning in a meaningful context through social discourse (D. Jonassen et al., 1995). Meaning making evolves from interactions and debate within a community of practitioners. As such, knowledge construction and sense making becomes a social process whereby meaning is socially negotiated through discourse among communities of learners in order to solve real-world problems (D. Jonassen et al., 1995).

Learners articulate experiences and reflect through debate, argumentation, and negotiation both internally and socially through interactions with others to construct meaning of their ideas, experiences and events (D. Jonassen et al., 1995). Reciprocal teaching, peer collaboration, cognitive apprenticeships, problem-based instruction, web-quests and anchored instruction approaches present examples of social constructivist classroom enactments (Kim, 1981). The interactive processes of negotiation, discussion, and information sharing promotes learning and results in knowledge construction in a social-cultural context (Wang, Teo, & Woo, 2009).

Whether considering an ill-structured problem or a well-structured problem, engaging in these interactive processes of group dialogue through discussion, negotiation, questioning, information sharing or debate can serve as an opportunity for problem solvers to iteratively refine their own problem representations in approaching a problem solution (Jonassen, 1997).

For example, social discourse can be used to work through the afore-mentioned problem-solving

steps of framing a problem, considering alternative aspects of a problem, identifying possible alternative solutions, and assessing the viability of the solutions, as well as monitoring and adapting solutions and presenting insights and solutions from multiple perspectives that might not otherwise be apparent. Through social constructivist problem solving, groups move beyond serving as more than a convenient way to accumulate the knowledge of the group members toward synergistically accruing insights and solutions that might otherwise not emerge. The group also enables multiple roles to be displayed to learners along with dialogue around those roles (Brown, Collins, & Duguid, 1989).

Social constructivist problem solving through scaffolding. One of Vygotsky's seminal theories, the zone of proximal development (ZPD), holds that learners learn best when they are assisted in their learning by having others involved. ZPD is recognized as "the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers" (Vygotsky, (1930-34/1978, p. 86).

A key construct of the ZPD includes scaffolding, where novice learners move to the next level of understanding with the assistance of teachers, peers or other adults. In this sense, individuals with more knowledge or expertise than the learner must be included in the ZPD and as part of the learning experience. This scaffolding enables a support system that assists the learner in internalizing the knowledge and ultimately enabling the learner to solve the problem at hand. Students experience their level of understanding and seek assistance in moving to the next level. While scaffolding focuses on collaborative learning in the student's ZPD, the end goal is that the learner develops individual skillsets that later do not require assistance such that "what the child is able to do in collaboration today he will be able to do independently tomorrow"

(Vygotsky, 1930-1934/1978, p. 211).

Social constructivism does not remove the need for the instructor, but rather redirects the instructor's attention toward creating a safe learning environment where knowledge co-construction and social mediation are critical. This requires instructors to be pedagogical content knowledge experts, whereby they understand the requirements and stages through which students travel on their journey towards understanding, particularly in a socio-cultural space (Adams, 2006; Ormrod, 1995; Vygotsky, 1930-1934/1978).

Scaffolds serve to aid in novices learning a complex skill from a more experienced instructor or peer, with the goal being to gradually remove the scaffolding as the collaborative learner's independent competence increases. Scaffolds may include classroom techniques or tools (Rosenshine & Meister, 1992). Within every domain lies a set of tools that is integral to that domain. For writers, words become tools, while for carpenters, hammers and chisels are tools. For computing and technology professionals, algorithms and protocols serve as conceptual tools for manipulating data. Yet, learning how to employ these tools moves beyond a simple set of rules. The conditions for using the tools arise from the activities of the communities and are framed by the way the members of the community see the world. Their meaning is a product of negotiation within the community. As such, the community members dictate how the tool is applied, reflecting the community accumulated insights. The conceptual tools become a function of the culture and community in which they are developed. For example, while carpenters and locksmiths use screwdrivers differently, mathematicians and physicists use formulas differently. As such, it stands to reason, that the problems being solved by the community of computing professionals through negotiation and dialogue impact the very nature of the tool use and the problems being solved (Brown et al., 1989).

Other classroom techniques that serve as tools may be dialogic in nature, mediating meaning-making in a social constructivist environment. Examples include reciprocal teaching, modeling, questioning, procedural prompting and conceptual modeling. Question prompts, including reason justification prompts are important to directing students attention to employing design principles and strategies as well as understating when, why and how to do so (Lin & Lehman, 1999). Question prompts are useful in compelling students to articulate the steps they have taken and their rationale for actions, and can be particularly useful to learners who tend to immediately jump into a solution when confronted with a complex task (Lin, Hmelo, Kinzer, & Secules, 1999).

Jonassen (1995) posited the benefits of reciprocal teaching, originally designed as a reading teaching method. Palincsar and Brown's (1984) study of reciprocal teaching revealed that through modeling, feedback and practice from a teacher with more expertise, learners significantly improved skill maintenance over time and skill transfer to novel tasks. Students were specifically required to engage in summarizing, questioning, clarifying, interpreting and predicting in order to enhance learner comprehension and monitor their own understanding (Palincsar & Brown, 1984). By using technology to move the constructivist driven reciprocal teaching approach to an online environment, Jonassen (1995) offered that learners are able to generate questions, summarize content, clarify points, and predict upcoming events. This approach shifts the dialogue from a purely verbal approach to one that relies on written dialogue, yet continues to emphasize the underlying constructs of meaning making.

In fact, whether considering well-structured or ill-structured problems, scaffolds can be integrated in each step of the problem-solving process. In addition to the afore-mentioned scaffolds, conceptual models can serve as a scaffold to illustrate the structural relationships

among problem components and aid the learner in representing the problem by scaffolding understanding (Jonassen, 1997). Analogical cases, previously described as a problem-solving heuristic, provide powerful scaffolding as they take the place of prior experiences and problem spaces not yet possessed by the novice learners. By making the elements of the analogical problem obvious to the learners, the instructor provides scaffolding that assists learners in mapping the previous problem onto a new one. Providing hints or suggestions on breaking down the problem into subproblems (or using means-end-analysis) presents yet another scaffold. Instructor presented prompts on operators or actions, cues and templates can serve to aid the more novice learner in approaching or solving the subproblems. Supporting argument construction with opposing underlying assumptions enables learners to articulate those differing assumptions and rationalize solutions. Debating and developing cogent arguments that support divergent thinking presents the learner with an opportunity to construct domain knowledge. Argumentation can be modeled for learners using a tool such as an argument template or checklist. Prompts that ask questions such as, 'How did you come to hold that point of view? On what do you base it?' can be also be used as scaffolding tools. All of these play a critical role in the problem-solving steps of identifying and clarifying alternative aspects of the problem, generating problem solutions and assessing the viability of alternative solutions.

Finally, instructor feedback serves as a key scaffold. Coaching from a more expert problem solver is critical as novice learners may struggle to generate new hypotheses from the clues of the failed attempts, as may become apparent during strategy implementation and evaluations stage of the problem-solving process. Scaffolded feedback should also address the relevant conceptual assumptions (possibly referring to the conceptual model) and methods used to solve the problem. Misconceptions and solution errors should be corrected as they are often

directly linked to incorrect assumptions. Activities such as debate and discussion become a means for instructors to highlight and correct learner misunderstandings and inconsequential knowledge (Adams, 2006). Through modeling and coaching, scaffolding is presented to the learner such that varying perspectives can be considered with the goal of selecting the best one based on reasoning and evidence, again a previously outlined and key aspect of the problem-solving process.

In summary, the three generalized stages to approaching a problem: (1) defining and representing the problem, (2) exploring possible solution strategies, (3) implementing strategies and reflecting back to evaluate the effects of the solutions present an ideal opportunity for learners to engage in collaborative learning and for faculty to scaffold collaborative learning. For example, in defining and representing the problem state from the context of the problem as presented, learners consider questions such as “what do I need to produce?” and “what does an appropriate solution look like?” Collaborative and scaffolded examination of questions such as these can provide additional perspectives on the problem attributes and conditions that may not be considered by a single individual. Likewise, the second stage of the problem-solving approach, exploring possible solution strategies, presents an ideal opportunity for problem solvers to negotiate, discuss and share information with the goal of moving beyond their individual solution strategies. Finally, implementing strategies and reflecting back to evaluate the effects of the solutions, necessitates examining the effectiveness of the solution at hand. Feedback and coaching from a more expert problem solver is critical to this stage as generating new hypotheses from the clues of the failed attempts is particularly challenging for novice problem solvers. The scaffolds, intended to provide assistance from a more experienced problem solver to a more novice problem solver in their ZPD while approaching a complex task, should

be temporary in nature, and they should be faded out as soon as possible and eventually removed, leaving the solver to independently apply their problem-solving skills to novel situations (Jonassen, 1997).

Collaborative Learning and Problem Solving in Computing Education Research

While social constructivist theory presented a suitable framework for examining collaborative learning in the context of problem solving, further consideration within the computing domain is warranted. Underscoring the critical importance of collaboration and communication skills for students entering the workforce, several studies demonstrated that new computing domain graduates struggle with teamwork and knowing when and how to articulate their need for assistance (Begel & Simon, 2008a, 2008b; Lingard & Barkataki, 2011; Loftus, Thomas, & Zander, 2011; Radermacher & Walia, 2013; Vivian et al., 2016). Additional work in the computing education research space consistently acknowledged the value and necessity of teamwork skills for graduates entering the workforce along with the positive impact on students' learning (Chidanandan, Russell-Dag, Laxer, & Ayfer, 2010; Hanks, 2008; Hanks, McDowell, Draper, & Krnjajic, 2004; Hazzan, 2003; Largent & Lürer, 2010; Lewis, Titterton, & Clancy, 2012; Murphy, Fitzgerald, Hanks, & McCauley, 2010; Pieterse, Thompson, Marshall, & Venter, 2012).

Despite the breadth of computing subdisciplines, the research commonly focused on some form of specific intervention in a classroom environment, frequently in the form of programming assignments or software engineering projects. With a few exceptions, studies also commonly focused on either problem solving or collaborative learning, rather than both. Often, problem solving was mentioned as the activity at hand with little or no clarification as to the type of problem or any specific problem-solving approaches. For example, Vihavainen, Airaksinen,

and Watson's (2014) literature coding review focusing on interventions for first-year programming students found that 14 of the 32 studies classified as having some form of collaborative intervention (peer-learning, cooperative learning, team-based learning or pair programming) yielded significant improvement in student passing rates (Vihavainen, Airaksinen, & Watson, 2014), indicating the positive impact of a social constructivist approach and echoing the predominance of constructivist theories previously exposed by Malmi et al. (2014). However, specific problem-solving approaches were not examined.

Faculty also instantiated the constructs of social constructivism through collaborative peer engagement. Commonly, positive outcomes were situated in a pair programming environment, where students worked in groups at a single terminal to write programming code. Positive effects of the collaborative student programming environment included increased transaction completions, more attempted problem-solving activities (although unclarified) along with more debugging of problems (also unclarified) (Murphy et al., 2010), reduced performance differences between inexperienced and experienced programming students (Lewis et al., 2012), significant attrition drops (from 37 percent to 5 percent) for female students (Lewis et al., 2012), and increased likelihood to submit working programs with a higher number of correctly implemented and required features (Hanks et al., 2004).

Pollock and Harvey (2011) moved beyond a single socially-constructed learning approach and combined multiple social constructivist instructional approaches including collaborative teams, student presentations, student critique of work, open-ended projects of student design, iterative processes, journal reflection, and motivation through helping others in a computer science independent study class with 12 students. Using quantitative and qualitative results from a student survey and students' reflective journals, they found that 82% of the student

journals demonstrated “evidence of reflection about the learning process beyond listing activities and challenges” (Pollock & Harvey, 2011).

While these research outcomes are imperative in driving computing education research, particularly around collaborative learning, the lack of problem-solving details, along with a focus on specific projects and specific domain-centered courses arguably represents a subset of the computing domain as a whole, and as noted by the ACM/IEEE curriculum guidelines. Despite some computing education research explicitly noting problem solving in the context of collaborative learning, details regarding the problems or the classroom problem-solving methods were typically ignored. For example, Valdivia and Nussbaum (2007) noted problem solving in their study that reported an increased ability by engineering students in a computer science programming class to communicate with both their peers and the instructor, along with increased social skills and student satisfaction. Yet the problem-solving activities were not detailed and the focus remained on the technological implementation of social constructivism. Specific collaborative aspects such as face-to-face interaction and group processing were required through the use of a collaboration algorithm that incorporated peer questions and required unanimous agreement among group members before moving onto a next step. The collaboration algorithm and questioning was reported through a piece of digital technology that enabled the instructor to monitor the students’ deficiencies and erroneous ideas in order to scaffold their progression. This presented a key scaffolding construct of social constructivism typically critical to the final stage of problem solving where coaching from a more expert problem solver is significant to generating new hypotheses from the clues of the failed attempts. The findings presented positive support for the technological co-construction of meaning and knowledge, yet

in the sense of the current study, a better understanding of the problem types and approaches would have presented a more robust contribution to computing education research.

Alternately, collaborative-based problems posed to students were often described with characteristics from either well-structured problems or ill-structured problems as previously elucidated, yet distinctions regarding the problem type or the associated problem-solving processes were rarely made. For example, Danielewicz-Betz and Kawaguchi (2015) demonstrated improved or newly acquired project management skills, communications skills, presentation skills, written skills, business skills and software development skills among students in a computer science course that collaboratively worked on software projects proposed by actual customers, functioning as potentially ill-structured problems. Yet no further details around problem-solving approaches were identified.

While most studies examining problem solving did not distinguish between ill-structured and well-structured problems and the associated problem-solving approaches, several did explore a very specific problem-solving intervention grounded in problem-solving research. Morrison, Margulieux, & Guzdial (2015), framed their study on cognitive load theory and demonstrated the use of subgoal labels through worked examples in programming to reduce cognitive load. Subgoal labels deconstruct problem-solving procedures into subgoals and functional pieces in order to better distinguish components of the fundamental problem-solving process. However, while it was briefly mentioned that the subgoal labeling exercise was peer-oriented in nature, no further details were presented around the level of collaboration (Morrison, Margulieux, & Guzdial, 2015).

In another study focusing on problem solving in computing education, Helminen, Ihantola, Karavirta, and Malmi (2012) built a tool to trace and analyze computer science

students' programming interactions during sessions to solve Parson's Problems. The goal was to "monitor students' programming process, collect data about the problems they face in the process, and give tailored feedback to them" (p. 124). Starter code was presented to students and the computer system presented feedback as two scaffolding methods. Using an interactive graph, the authors were able to visualize how students solved the problems in order to explore patterns and anomalies, but they found that student solutions varied and their overall problem-solving strategies were inconclusive (Helminen, Ihantola, Karavirta, & Malmi, 2012). Although unattended to in the Helminen, et al. (2012) study, the fact that student solutions varied may be interesting in-and-of-itself when further considering faculty approaches to and perceptions of collaborative learning within the context of problem solving as enacted in a classroom learning environment.

Citing that "general problem-solving issues are discussed but often not in the explicit and structured manner needed," Muller (2005) examined how using a pattern oriented instruction (POI) approach impacted students' analogical reasoning in regards to algorithmic problem solving and solution design in several introductory high school computer science courses in Israel. Again focusing on individual students rather than collaborative teams, the authors connected problem solving to schema theory and cognitive load theory, grounding the work in a cognitive theory framework with analogical reasoning, as previously described in well-structured problem-solving processes (Muller, 2005). Initial results indicated that the pattern oriented instruction proved beneficial in students extracting structural features of a problem rather than being distracted by surface features. Further investigations of collaboration in the use of POI may prove an interesting direction.

In contrast, Ge and Land (2003) did marry the constructs of problem solving and

collaborative learning in their focus on peer scaffolding in ill-structured problem solving. The authors examined the effects of peer-led questioning using prompts to solve an ill-structured information technology problem. Quantitative results demonstrated that the question prompts had significant positive effects on student problem-solving performance, while peer interactions did not show significant effects. In contrast, the qualitative findings revealed positive effects of peer interactions in facilitating cognitive thinking and metacognitive skills, suggesting that peer interactions must be guided and monitored with various strategies in order to realize the benefits.

While many computing education research studies advocated the positive outcomes of using collaborative teamwork in computing education, Yadin and Or-Bach (2010) moved in the opposite direction by examining ‘The Importance of Emphasizing Individual Learning in the “Collaborative Learning Era.”’ The authors asserted that, “current emphasis on the benefits of collaborative learning belittles the importance of individual learning processes and reduces the opportunities to require and assess individual learning within IS [Information Science] education.” The research assigned individual homework to students and compared the students’ failure rates to that of previous years where group assignments were used. The students in the individual assignments course performed with a significantly lower failure rate, and surveys indicated that students had an overall better experience in the classes with the individual assignments. One student even went as far as to offer that, “Getting feedback adapted to me led me to invest more in the course because I felt I was treated individually by the teacher” (Yadin & Or-Bach, 2010). Yadin and Orbach (2010) contended the results support their impressions regarding group work that “many students do not invest the time and effort required in thoroughly thinking about the courses’ assignments, about possible ways to solve them, and about how to evaluate the solution they submit;” however, their claims would have been greatly

strengthened by including an overview of how the group assignments were previously approached and what instruction regarding performing as an effective team-member was provided to students, contributing to considering *why* the results are integral to computing education research (p. 191).

In summary, considering the computing education research presented in the areas of social learning and problem solving, several areas of opportunity become apparent. Research typically focused on a single computing subdomain or content knowledge area, leaving unexplored the ideas of collaborative learning in the context of problem solving across computing subdomains. Consistent with the analysis from Malmi et al. (2014) CER, study scenarios were commonly presented with limited theories or framework references to undergird the research. When theories are presented in this manner, they serve as a tangential connection for testing teaching methods and tools rooted in the instructors' intuition or PCK and rarely ground the study to a level of informing research questions or methodology. Positive results were regularly reported in regards to students' experiences and perceptions through surveys, peer reviews and exam/assignment scores, yet collaborative learning and problem solving were often studied in the absence of a detailed explication of the instructional approach involved. Excepting a few notable considerations, even when problem solving was indicated in the study, few details regarding the context of the problems were included. For example, what were the characteristics of problems presented, what problem-solving strategy or heuristic, if any, was presented to students and why was that method selected, and what are students learning in regards to appropriate collaborative learning skills and why? This, along with the underdeveloped associations to theories and frameworks presented in the Malmi et al. (2014) findings, presented

a stronger case to further consider questions of *why* rather than *how*, in contributing to the computing education research domain in the form of theories originating in CER.

Chapter 3

Method

Research Design

This exploratory case study followed a naturalistic inquiry approach (Lincoln & Guba, 1985). A naturalistic approach allowed for examination of the practices of problem solving in the natural instructional computing environment, a context free from requirements for the control or manipulations of behavioral events. Instructional classroom and laboratory observations, in-depth interviews and course artifacts contributed to a holistic perspective of the case. These aspects were particularly relevant as a case study relies on multiple sources of evidence and is informed by theoretical propositions that guide data collection and analysis (Yin, 2013).

A case study approach was appropriate for this study because it allowed for the examination of the nuances of practice within a domain where current understandings are limited. This nuanced examination of context and perceptions and approaches of the participants provided a lens into the issues of central importance for this study (Martella et al., 2013). Notably, collaborative learning in the context of problem solving and novice/expert approaches have been similarly examined through qualitative methods in the associated and applied STEM fields of geology, organic chemistry and science education using semi-structured interviews (Balliet, Riggs, & Maltese, 2015; Bhattacharyya & Bodner, 2014; Luft & Pizzini, 1998) and naturalistic observations (Balliet et al., 2015; Luft & Pizzini, 1998). The purpose and methodology of the current study was further informed by a 2017 exploratory pilot case study that also followed a naturalistic inquiry approach and is outlined in the following sections (Lincoln & Guba, 1985).

Phase I: Pilot Study

The trajectory of the current study was informed by an initial exploratory pilot study that examined faculty perceptions and approaches to problem solving and reflection and the nature of collaborative learning in classroom enactment and laboratory sessions in second-year undergraduate computing courses. Faculty interviews, field site observations and course artifact examination served as the data sources. The pilot study was guided by an experiential learning theory framework and produced a functional set of procedures and a priori codes that were employed for the current study's data analysis, and as such, are reviewed in the following sections.

Pilot Study Data Analysis Procedures and Codes

The exploratory pilot study followed Spradley's (1980) 12 steps of the developmental research cycle, and the data was analyzed using a domain analysis format for identifying included terms, semantic relationships, and cover terms. This overlaps with Lecompte's (1987) process for establishing categories as emerging patterns.

Pilot study faculty interview analysis. Semi-structured interviews were conducted with five computing faculty instructing second-year courses. As the focus of Interview I was distinct from Interview II, each was analyzed separately to identify codes, themes and overall patterns. All interviews were transcribed and read verbatim. After converting each of the documents to a delimited file, they were imported into Microsoft Excel for coding. Each of the fixed length text cell rows was numbered for reference during analysis. I then reread each transcript and highlighted phrases that referenced instructional styles and student learning approaches that served as the basis for coded terms. Using an amalgamation of descriptive and in vivo coding, I

reviewed each of the highlighted words and/or phrases and identified a code in column 2 of the spreadsheet (Saldaña, 2015).

Many of the coded terms appeared throughout all of the faculty interviews. To distill down the list, a nested indexing formula that extracted unique distinct terms from the full list of initial codes was run. The list was then sorted in alpha order for easier reading and comparison. Some examples of the codes included peer learning, teamwork, domain specific problems, problems vs. solutions and open-ended solution design. To validate the data list, a python script was run to extract all distinct terms from the list of codes. The resulting set of terms matched the list generated in my spreadsheet with the exception of a few terms that were identifiable with typographical differences such as trailing whitespace (text including spaces and carriage returns such as teams_ and teams). Using the alpha-ordered set of distinct terms, I examined the list for errors that led to data duplication during coding. Noted coding term differences included: plural versus singular, hyphenated versus non-hyphenated terms, spelling errors, etc. I then updated the spreadsheets such that coded terms were consistent throughout all the coding fields. Coding was not rolled up into themes at this point; rather this phase served solely to cleanse the data of discrepancies from human coding error. Coded fields where discrepancies were identified were highlighted in the original output sheet. I then reran all the indexing to regenerate the distinct set of codes.

After initial coding, term validation, and code cleansing, the codes produced from the initial data review were examined and categorized through analytic reflection (Saldana, 2015). This proved to be an iterative process as I moved back and forth between the theming and the codes repeatedly for faculty Interview I and this process was repeated for Interview II. The codes were then organized into overarching themes that distinctly aligned to the areas of problem

solving in regards to faculty and student approaches, problem-based learning within the context of problem solving, collaborative learning in the form of team and group work and reflection in relation to instructor feedback and student skills. These overarching emergent themes were then analyzed given the previously outlined problem-based learning and experiential learning frameworks.

Pilot study instructional observation analysis. A similar coding and data analysis method was used for the field observations. Each set of field notes was imported into fixed length cells in Excel where rows were numbered for easy identification and reference. Using descriptive and in vivo methods, codes were identified and recorded in column two of the spreadsheet. These codes were then organized into the overarching themes that supported the themes emerging from the faculty interview analysis.

Table 2 presents the key collaborative learning and problem-solving codes that emerged from the faculty interview and instructional observation analysis process.

Table 2
Resulting Pilot Study Inductive Codes

Collaborative Learning	Problem Solving
Group work challenges <ul style="list-style-type: none"> • grading/evaluation of individuals vs. group • forming teams/groups • workload disparity among group members (loafers) • time managing groups 	Real-world problems versus class-problems (games/other) versus abstract examples <ul style="list-style-type: none"> • instructor roleplaying as customer • project size
Cheating vs. collaboration	Employers and employment
Individual skillsets	Optimal answer vs. correct answer <ul style="list-style-type: none"> • Code/project works or not
Collaborative work vs. independent learning	Open-ended versus closed-ended problems <ul style="list-style-type: none"> • customer requirements • requirements vs. solutions • students make choices

Peer review	Structured vs. unstructured problems (student defined problem/project) Problem decomposition/breaking up the problem into subparts/subproblems Examples of instructor guided learning, faculty model problem solving by <ul style="list-style-type: none"> • posing questions • walking students through examples • walking students through steps
Student presentations	
Collaboration required in employment	
Students work together informally (outside of class)	
Students required to work in groups	
Collaboration forbidden	
Building social capital	

Pilot study course artifact analysis. Nearly 700 course artifacts were coded for analysis. These artifacts included course exams, in-class exercises, homework, projects, labs and graded exams and assignments from student participants. Many of the artifacts were very technical in nature, similar to technical manuals for programming. The nature of the course artifacts along with the volume necessitated an alternate coding methodology from the faculty interviews and instructional observations. Rather than identifying codes and themes directly on the documents, I developed a template for each set of course artifacts where findings were organized by assignment type: syllabus, presentations, projects, labs/exercises/homework and exams. Using an a priori method grounded by the codes that had previously emerged from the faculty interviews and instructional observations, I coded the course artifacts, identifying the tacit themes and themes that connect different subsystems of a culture, and specifically identifying examples of each in the template (Spradley, 1980). As seen in the sample excerpt in Table 3 below, the template was also used to record my overarching observations in regards to the constructs of problem-solving content, social/team/group learning requirements and guidelines and reflective learning expectations as delineated by the problem-based learning and experiential learning theory frameworks.

Table 3
Pilot Study Course Artifact Coding Template

172 files	Problem-solving	Reflection	Social learning	Summary
Summary	there is no evidence of PBL or formal problem-solving in the course	no evidence of reflection other than studying for exams	no evidence of social learning from artifacts	overall, there is little evidence of ELT in the artifacts examined.
	one might argue that every solution to a db question is solving a problem or modeling specific db problem solving, but no formal methodology for problem solving is posed. Steps for solving the db problems are presented and may be interpreted as database problem-solving steps.			
	class notes reflect what was covered technically as an example during class - students can use this to study or reflect upon later			
	rubrics well-defined			
Syllabus	PPTs	Projects	Labs	Exam Review
nothing in syllabus regarding social learning or working in teams or groups, not for an assignment or in regards to working after graduation	very factual and mechanical in nature. No breakout time, no formal group work in class.		really homeworks - no separate labs homeworks are structured with required answers - no why or explanation or reflection questions	Practice exams for students - both written and practical

Pilot Study Findings Informing the Current Study

The pilot study findings revealed three major themes: (1) faculty perceptions and approaches to problem solving centered on modeling problem decomposition to break down larger problems into smaller steps or subproblems, (2) reflection was informally modeled by faculty during instruction, but primarily occurred internally for students and (3) faculty noted the

importance of collaboration in the workforce, yet perceived collaborative student work to be at odds with developing individual skillsets, leading to the predominance of informal rather than structured social learning. These findings had implications for understanding instructional approaches and best practices in computing education as well as the development of a computing education theoretical framework.

Examining a subsample of the faculty that teach computing courses to undergraduate second-year students was appropriate to understanding approaches to problem solving in coursework that is intended to prepare students for co-operative education experiences. These cooperative education experiences which typically follow second year coursework, place students in paid internships with major computing stakeholders in government, industry and private organizations.

The current study adds to the initial pilot study by examining the nature of collaborative learning in the context of problem solving enacted by faculty in first-year courses for a more holistic view of problem solving in a naturalistic computing education context. By expanding the pilot study dataset to include first-year data, I sought to more deeply understand how faculty influence approaches to problem solving, particularly as students prepare for their first entry to the workforce through co-operative work experience. The additional focus on first-year computing courses was also based on the fact that first-year computing courses explicitly emphasized problem solving as a key course objective with the expectation that it further grounded students' preparation for cooperative education and workforce participation as well as downstream courses. Examining faculty perceptions of and approaches to problem solving may yield important insights on students' preparedness for co-operative education experiences and importantly to collaborate and engage in collaborative learning in the context of solving

problems. This natural extension thus involved understanding the ways in which the curriculum and pedagogical approaches in computing courses prepare students to solve problems for 21st century computing workplace environments.

Furthermore, the pilot study's emergent inductively developed codes were limited from being grounded in problem-based learning and experiential learning theory. However, the emergent codes were instrumental in informing the current study's direction that more broadly embraced the problem-solving literature and refocused collaborative learning within the context

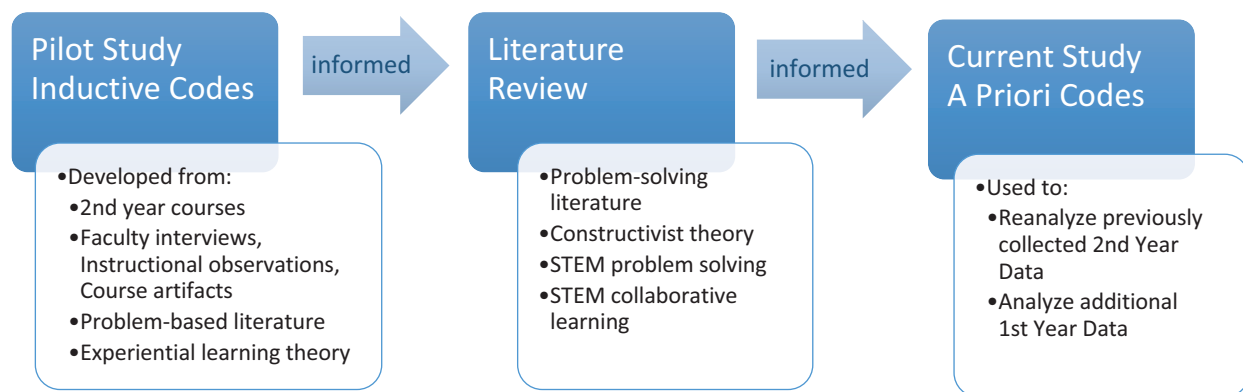


Figure 1. Evolution of the pilot study to the current study.

of social constructivist theory. This in turn informed the revision of pilot study codes to a priori codes grounded in the literature that served for the data analysis in the current study. This process of evolution from pilot study to current study is summarized in the Figure 2. Further details around the a priori codes and the current study's methodology are addressed in the following sections.

Phase II: The Current Study

Context of the Study

Site. This study was conducted at a four-year undergraduate Private Technical University (PTU) in the northeast. Founded in the early 1800's, PTU was an early pioneer in practice-based

and cooperative education. Today, PTU is one of the largest technical institutions of higher education in the United States. Over the past several years, PTU's incoming classes have not only increased in size but also have improved in quality and diversity. PTU offers a broad array of undergraduate and graduate programs in its nine colleges (Bailey et al., 2015). The context for this study focused on three departments in a computing college: the Department of Computer Science (CS), the Department of Information Sciences and Technologies (IST) and the Department of Software Engineering (SE). These three departments were selected for two reasons: (1) they were the earliest formed departments in the college and (2) many of the courses from these departments serve other programs in the college. The Department of Computer Science identifies itself as providing students with a deep foundation in theory and modern software and hardware concepts as well as introducing students to numerous programming languages and paradigms while providing both traditional and applied research opportunities. Hands-on work and experiential learning are cited as keys aspects of the program. The IST department distinguishes itself as having a focus on utilizing technology to solve real-world problems, specifically aimed at web and mobile computing, human-centered computing and database, and network and systems administration, all while using a hands-on approach to technology to design, build, evaluate, analyze and manage technology. The SE department defines their primary mission as educating professionals who can define, design, develop and deliver high quality software systems within resource constraints while focusing on the people involved in engineering teams, the selection of processes employed, and the characteristics of products created.

Courses. For purposes of this study, first-year and second-year classes in the Computer Science, Information Sciences and Technologies and Software Engineering departments were

identified. Given their long-standing history, the curriculum in these three departments is well-established in relation to other prospective areas and provided a fundamental set of offerings from which to select. First-year courses were selected because they provided a foundational knowledge-base and skillset for all computing students in all of their downstream coursework. Additionally, the first-year courses selected as part of this study specifically identified problem solving in their course descriptions. Second-year courses in particular were identified because of the positionality within the four-year curriculum: (1) all students have some established background in computing at the point they are second-year students, (2) faculty are continuing to prepare students for additional downstream coursework and (3) students are typically preparing for their cooperative education experience that occurs between the second and third academic year. Prospective courses were mapped to a schedule and then selected based on accessibility and with the rationale that they were required as part of their respective degree programs.

Participants. Seven full-time faculty served as participants in this study. The participants were purposely selected to represent first-year and second-year courses that establish fundamental computing concepts to students and serve as integral foundations to their academic programs and professional careers. This included two faculty members from the Department of Computer Science, four faculty members from the Department of Information Sciences and Technologies and one faculty member from the Department of Software Engineering. Two were female and five were male. The faculty instructors were a mix of tenured faculty and non-tenure-track lecturers. Both roles carry high expectations for quality teaching with the key differentiating factor being that tenured faculty are required to conduct research. The faculty participants experience as educators ranged from two to 39 years and industry experience (including consulting) ranged from five to 17 years. All of the faculty were deeply embedded

within the university with a strong understanding of the goals of their academic program, the courses and the curriculum. Table 4 summarizes the participants.

Table 4
Participants

Pseudonym	Department	Course Year	Gender	Faculty Position
Bill	Computer Science	1 st	Male	Tenured
Chris	Computer Science	2 nd	Male	Lecturer
John	Software Engineering	2 nd	Male	Tenured
Sue	Information Sciences and Technologies	2 nd	Female	Tenured
Dave	Information Sciences and Technologies	1 st	Male	Tenured
Joan	Information Sciences and Technologies	2 nd	Female	Lecturer
Tim	Information Sciences and Technologies	2 nd	Male	Lecturer

While all of the faculty in the units were capable of teaching students at all levels of the curriculum, in this study, two of the faculty served as the instructors for the first-year courses and five served for the second-year courses. While this may at first appear somewhat disparate in the nature of participants, the two different first-year courses served as a foundation for all the computing students regardless of their future domain specific curricular path, and as such grounded the study in examining faculty perceptions of and approaches to problem solving, and the nature of collaborative learning in preparation for cooperative education and eventual workforce participation.

Researcher Positionality

In contrast to Seidman's (2012) advice to move outside one's own student body for research, I conducted this case study as backyard research (Glesne & Peshkin, 1992). While there are reasons to be cautious of a backyard study, my inherent subjectivities strengthened the overall research, and influenced the entire research process (Peshkin, 1982, 1988). I recognized

these subjectivities and embraced them as grounding the work in a manner that joined my personal understandings with the data that I collected and allowed me to make a distinct contribution to the computing education research domain (Peshkin, 1985, 1988).

Research Design. My 20 years of institutional knowledge as a tenured computing faculty member along with my previous work designing and developing two undergraduate computing degree programs and more than a dozen academic courses, influenced the research design. Throughout my work as an instructional faculty member, I observed post-secondary computing course titles and descriptions using terms such as ‘problem solving’ and ‘collaborative work’. Spurred by reflections on my own curriculum development that centered on problems instinctively developed as a pedagogical content knowledge expert in the computing domain, I began to ask questions such as, ‘what exactly do the terms problem-solving and collaborative work mean in computing education?’ and more specifically, ‘what does this mean in terms of faculty developed curriculum?’ and “how are faculty fostering an environment of problem solving and collaboration in order to prepare students for the workforce?’

My subjectivities further led me to consider that, imperative to understanding how problem solving is enacted in the classroom is an understanding of the types of problems being posed. My subjectivities led me to believe that a deep understanding of faculty enactments in classrooms and their developed coursework as well as personal interviews and stories were critical in developing an understanding of how computing students are being instructed, and this in turn is critical in understanding how to proceed with our future curriculum development. My subjectivities influenced the focus in this research on faculty understandings, perceptions of, and enactments of computing problems, problem solving and collaborative learning, and directly led to the three research questions. In this sense, my subjectivities enabled me to distinctively

identify a new research direction that presented a shift from previous domain research focusing on student evaluation of classroom interventions around problem solving and collaboration.

Overall, my previously noted curiosities motivated inquiry in this faculty-focused research and, coupled with my affinity for social constructivist theory, shaped the development of the research questions and the theoretical framework for the study (Given, 2008).

Data Analysis. My 20 years of institutional knowledge served as an advantage in understanding the origins and structures of the courses and exercises that evolved as part of the curriculum. This institutional knowledge uniquely positioned me with a deep understanding of the computing context, which proved particularly useful in analyzing the faculty interviews, instructional observations and course artifacts. My backyard knowledge of the computing domain, program and participants presented a subjectivity the research that guided me in fully understanding the nuances being conveyed. For example, my deep understanding of the computing domain content enabled me to examine aspects of the problems being presented and identify the characteristics in relation to those presented in Jonassen's Typology. Yet, I also needed to acknowledge and understand these subjectivities in order to address some findings that misaligned with my professional affinity toward a social constructivist theory. For example, when data analysis revealed surprising findings that ran directly counter to social constructivist theory, I reflected deeply on the analysis and data in order to present the participants accounts with an open mind (Starks & Brown Trinidad, 2007). I did this by bracketing and again considering my subjectivities as a computing domain educator. This was with an eye toward working with curriculum stakeholders in moving the findings toward a prospective action plan for the college and computing education community in regards to future curriculum design and development as well as future research (Seidman, 2012).

Data Sources

This case study used a non-probabilistic sampling strategy where a purposeful set of participants was selected and provided an information-rich context for the case (Creswell, 2013).

Faculty interviews. Two interviews lasting approximately 40 minutes, were conducted with each of the faculty participants whose classrooms/labs were also be observed. Prior to the interviews, each faculty participant signed-off on IRB documentation that included an explanation of the purpose of the research.

Interview protocol. The interviews were conducted using a semi-structured standardized open-ended interview process with a defined protocol that allowed the participants to talk freely about their perspectives and experiences (Creswell, 2013; Martella et al., 2013). Interview questions were previously mapped to the research questions as part of the protocol to ensure that key points were addressed in each of the interviews. Using a series of experience or behavior questions to guide the interviews, I probed for evidence around problem-solving aspects of classroom approaches and the collaborative aspects of the learning environment. For example, one question probed what steps a faculty member expects from students as they are working through a problem. I also examined the faculty members' perceptions on their own approaches to teaching problem solving and their perceptions on how students approach and engage in problem solving with questions regarding how problems are typically posed to students. A similar method was implemented by Balliet, Riggs and Maltese (2015) with their semi-structured interview questions that asked geology students to present a general overview of their day in the field, with specific questions focusing on problem-solving strategies, failures and successes, and changes they would make to their solutions or strategies. Researcher reflections were recorded at the end

of each interview in a reflective journal. Interviews were recorded and transcribed verbatim. The faculty interview protocol is included as Appendix A.

Instructional classroom and laboratory observations. Instructional classroom and laboratory observations were conducted in the natural classroom and laboratory for the faculty members and their respective students. Two-to-three class, laboratory and recitation sessions were observed to study the participants' behavior within their natural context (Creswell, 2013). The number of observation sessions was dictated by the daily activities in the classroom and whether or not fully relevant data was gathered during each observation.

Observation protocol. Observations focused on documenting instructors and students during the nuanced processes of problem solving in a collaborative learning setting. Broad questions such as "What people are here?" and "What are they doing" were inherent in the observations. The activities of the people, the physical characteristics of the social situation, and what it felt like to be part of the scene were observed (Spradley, 1980). Broad descriptive observations as well as focused and selective observations were documented. Because of the extremely fast pace at which some instructors spoke, in lieu of recording verbatim observations at all times, I elected to expedite the documentation process by only recording nouns or verbs so as to capture the essence of the activity and setting (Spradley, 1980).

Field notes. Field notes recorded on my laptop during observations were collected primarily as a passive participant (Spradley, 1980). Engagement with the faculty and students was very limited in nature. I positioned myself in the classrooms and the laboratories in an out of the way place, typically toward the back of the room where I quietly took notes. I intentionally approached the setting in this manner such that my presence would not call extreme attention to my activities (Spradley, 1980). At the conclusion of each field site observation, I recorded my

overall senses, feelings and perceptions of the class. I then reviewed and edited my notes shortly afterward to fill in any gaps as well as correct any missing verbiage. The field site observations then further informed the interviews conducted as part of this case study.

Course artifacts. Course artifacts such as syllabi, classroom exercises, laboratory exercises, and examinations were collected during and after the observation period. All artifacts were examined in regards to problem-solving approaches and experiences in the context of a collaborative learning setting. Table 5 summarizes the qualitative data sources serving this study.

Table 5
Current Study Data Sources

Data Source	Computer Science Department Information Sciences and Technologies Department Software Engineering Department
Instructional Observations	Field notes for 15 observations (22 hours) in a class/lab/recitation setting across three departments, (setting was one instructor and ~30 students each)
Faculty Interviews	Transcripts for 14 interviews approximately 40 minutes in length with faculty (two interviews with each of seven faculty)
Course Artifacts	Syllabi, in-class exercises, projects, laboratory exercises, examinations (nearly 800)

Data Analysis

An amalgamation of inductive and deductive coding methods was implemented in this study. Inductive analysis included the use of emergent codes from the current dataset following a subset of the methodology as outlined in the pilot study. Inductively developed codes from the pilot study alongside the problem solving and constructivist theory literature informed the a priori codes for the current study. Creswell (2013) noted that when analyzing data with a priori or prefigured codes developed from a theoretical model or the literature, the additional use of

emergent codes is recommended in an effort to reflect the views of participants in a traditional qualitative manner.

Problem-solving approaches a priori codes. In considering how computing faculty who teach first-year and second-year computing courses enact problem solving in their courses, the approaches to well-structured and ill-structured problems were considered. To ensure that all inductively developed codes from the pilot study were captured in the a priori code set extrapolated from the problem-solving literature for the current study, each was mapped in Table 6 below. Note that upon literature review, some of the codes initially developed in the pilot study as problem-solving codes were reframed as social constructivist codes and problem-type codes and are presented in the following sections.

Table 6
Mapping of Pilot Study Codes to Current Study a Priori Codes

Problem-Solving Pilot Study Inductively Developed Codes	A Priori Codes for Current Study
Real-world problems versus class-problems (games/other) versus abstract examples <ul style="list-style-type: none"> instructor roleplaying as customer project size 	Jonassen's (2000) problem-solving typology
Employers and employment	Collaborative learning a priori codes developed from pilot study
Optimal answer vs. correct answer <ul style="list-style-type: none"> Code/project works or not 	Jonassen's (2000) problem-solving typology
Open-ended versus closed-ended problems <ul style="list-style-type: none"> customer requirements requirements vs. solutions students make choices 	Jonassen's (2000) problem-solving typology
Structured vs. unstructured problems (student defined problem/project)	Jonassen's (2000) problem-solving typology
Problem decomposition/breaking up the problem into subparts/subproblems, Chunking up the problem	Problem-solving approaches a priori codes
Examples of faculty modeling problem solving by	A priori social constructivism scaffolding codes

<ul style="list-style-type: none"> • posing questions • walking students through examples • walking students through steps 	
---	--

Table 7 below presents the a priori codes for examining problem-solving approaches that were developed from the problem-solving literature.

Table 7

Problem-Solving a Priori Codes Developed from the Literature and Informed by the Pilot Study (Jonassen, 2000)

Problem-Solving Approach Stages A Priori Codes Developed from Literature and Informed by the Pilot Study	Details
Ill-structured problem-solving approaches	
(1) frame the problem	Examine the context assemble conceptual and relevant domain knowledge from memory rather than from the academic materials presented as part of the lesson and focusing on a constrained set of rules (Voss & Post, 1988)
(2) identify and clarify alternative perspectives of the problem	Identify differing goals and problems and divergent solutions
(3) generate possible problem solutions	Generate possible solutions considering problem constraints Use heuristics as in well-defined problems
(4) assess the viability of alternative solutions	Present a preferred solution based on contrasting internal or group views
(5) monitor the problem space and solution options	Strategy planning and monitoring epistemic nature of solutions throughout the process. Problem solver must understand the limits of their knowing
(6) implement, monitor and adapt the solution	Reflecting on scenario and making inferences about the solution viability for other problems
Well-structured problem-solving approaches	
(1) defining and representing the problem	Extract the goal from the problem statement as it is presented Understand the problem attributes, the goals and the possible solutions and strategies for Solve the problem (Polson & Jeffries, 2014) Search through the mental problem space and Access prior domain specific knowledge while generating hypotheses and possible solutions
(2) exploring possible solution strategies	Heuristics

	Recall analogical reasoning Means-end-analysis Decomposing to subproblems Generate and test
(3) implementing strategies and reflecting back to evaluate the effects of the solutions (Gick, 1986)	Testing solutions Process adjustment if solution fails Coaching (from expert) to generate new hypothesis from failed attempts clues

Problem types a priori codes. In considering the types of problems that were being posed/used by computing faculty who teach first-year and second-year courses, the overarching attributes of well-structured and ill-structured problems were considered. While Jonassen (1997) had previously asserted these two broader categories of well-structured problems (rooted in information processing theory and considered generalizable) and ill-structured problems (established in constructivism and situated learning that argues for the domain specificity of an authentic context), his subsequent extensive analysis of problems revealed that these two categories alone were insufficient when considering the range and complexity of problem-solving outcomes. As a result, Jonassen (2000) proposed the set of more granular problem types as a problem-solving typology and seen in Table 8. It integrates learning activity, inputs, success criteria, context, structured-ness and abstractness into the problem-solving outcomes. The horizontal table header includes the following problem types: logical, algorithmic, story, rule using, decision-making, troubleshooting, diagnosis-solution, strategic performance, case analysis, design, and dilemma.

Table 8
 Typology of Problems (Jonassen, 2000)

	Problem Types										
	Well-structured						Ill-structured				
	Logical problems	Algorithmic problems	Story problems	Rule-using problems	Decision-making problems	Trouble shooting problems	Diagnosis-solution problems	Strategic performance problems	Case analysis problems	Design problems	Dilemmas
Learning activity	Logical control and manipulation of limited variables; solve puzzle	Procedural sequence of manipulations; algorithmic process applied to similar sets of variables; calculating or producing correct answer	Disambiguate variables; select and apply algorithm to produce correct answer using prescribed method	Procedural process constrained by rules; select and apply rules to produce system-constrained answers or products	Identifying benefits and limitations; weighing options, selecting alternative and justifying	Examine system, run tests, evaluate results, hypothesize and confirm fault states using strategies (replace, serial elimination, space split)	Troubleshoot system faults, select and evaluate treatment options and monitor, apply problem schemas	Applying tactics to meet strategy in real-time, complex performance maintaining situational awareness	Solution identification, alternative actions, argue position	Acting on goals to produce artifacts, problem structuring and articulation	Reconciling complex, non-predictive, vexing decision with no solution, perspectives irreconcilable
Inputs	Puzzle	Formula or procedure	Story with formula or procedure embedded	Situation in constrained system; finite rules	Decision situation with limited alternative outcomes	Malfunctioning system with one or more faults	Complex system with faults and numerous operational solutions	Real-time, complex performance with competing needs	Complex, leisure-time system with multiple ill-defined goals	Vague goal statement with few constraints, requires structuring	Situation with autonomous positions

Success criteria	Efficient manipulation; number of moves or manipulations required	Answer or product matches in values and form	Answer or product matches in values and form; correct algorithm used	Productivity (number of relevant or useful answers or products)	Answer or product matches in values and forms	Faults identification, efficiency of fault isolation	Strategy used, effectiveness and efficiency of treatment, justification of treatment selected	Achieving strategic objective	Multiple, unclear	Multiple, undefined criteria, no right or wrong – only better or worse	Articulated preference with some justification
Context	Abstract task	Abstract, formulaic	Constrained to predefined elements, shallow context	Purposeful academic, real-world, constrained	Life decisions	Closed system, real world	Real world, technical, mostly closed system	Real-time performance	Real world, constrained	Complex, real world, degrees of freedom, limited input and feedback	Topical, complex, interdisciplinary
Structured-ness	Discovered	Procedural, predictable	Well-defined problem classes; procedural, predictable	Unpredicted outcome	Finite outcomes	Finite faults and outcomes	Finite faults and outcomes	Ill-structured strategies, well-structured tactics	Ill-structured	Ill-structured	Finite outcomes, multiple reasoning
Abstractness	Abstract, discovery	Abstract, procedural	Limited simulation	Need-based	Personally situated	Problem situated	Problem situated	Contextually situated	Case situated	Problem situated	Issue situated

This typology resulted from Jonassen (2000) collecting and performing a cognitive task analysis on hundreds of problems to identify their attributes. An iterative sort based on the emergent characteristics resulted in the 11 distinct problem types and represents a continuum from left-to-right of well-structured to ill-structured problems (Simon, 1973). Jonassen (2000) also contended that the table is taxonomic in nature with the well-structured problems on the left serving as a prerequisite to the ill-structured problems on the right. For example, case-analysis problems require the individual to solve decision making problems and aspects of troubleshooting problems such as hypothesis generation and testing. According to Jonassen (1997), while the broad categories of ill-structured and well-structured problems are useful in considering the varying aspects, they should not be considered as separate entities, but rather points on a continuum, and as presented in the typology. Relevant to where a problem lies on this continuum is the problem complexity, goal state and criteria clarity, component domain skill prescriptiveness and the number of viable solutions or solution paths.

Jonassen (2000) also clearly noted that the typology's categories were neither absolute nor discrete, nor independent, nor mutually exclusive of each other. In line with this study's secondary goal that aimed to contribute to an emerging computing education problem-solving framework, Jonassen (2000) indicated his intent that additional research may identify new categories or restructure the existing categories. As such, this typology served as an ideal model for examining the types of problems being posed in problem solving in computing education. While considering the overarching and broad categorization of well-structured and ill-structured problems, the model provided a more granular structure in which to examine problems posed to students as well as allowing for consideration of an emerging computing education research framework. As such, each of the problem types was implemented as an a priori code in

examining course artifacts as well as problems noted during instructional observations and described in faculty interviews.

Social learning a priori codes. Social learning codes inductively developed during the pilot study were used along with additional social constructivist codes developed from the literature and focusing on scaffolding. Both are presented below as Tables 9 and 10.

Table 9
Social Learning Codes

Social Learning Pilot Study Inductively Developed Codes Used as A Priori Codes
Group work challenges <ul style="list-style-type: none"> • grading/evaluation of individuals vs. group • forming teams/groups • workload disparity among group members (loafers) • time managing groups
Cheating vs. collaboration
Individual student skillsets
Collaborative work vs. independent learning
Peer review
Student presentations
Collaboration required in employment
Students work together informally outside of class
Students required to work in groups
Collaboration forbidden
Building social capital

Table 10
Scaffolding Codes

Scaffolding A Priori Codes Developed from the Literature
reciprocal teaching
modeling
questioning
procedural prompting
conceptual modeling
question prompts
analogies

hints
cues
templates
argumentation
debate
instructor feedback
instructor coaching
examples
stepping through problem

Data analysis and theming. After coding the data utilizing the data-driven emergent codes alongside the theory-driven codes, thematic data analysis using a template approach was employed for this study (Braun & Clarke, 2006; Fereday & Muir-Cochrane, 2006). All four sets of a priori codes were used in analyzing the faculty interviews, instructional observations and course artifacts. Table 11 below summarizes the data sources and four sets of a priori codes implemented during coding and analysis.

Table 11
Current Study Data Sources and Associated Analysis Codes

Data Sources	Analysis Codes
Faculty interviews, Instructional observations, Course artifacts	A priori problem-solving approaches codes developed from pilot study and problem-solving literature
	A priori codes developed from Jonassen’s (2000) Problem-solving Typology
	A priori collaborative learning codes developed from pilot study and constructivist theory literature and STEM collaborative learning literature
	A priori scaffolding codes developed from pilot study and constructivist theory literature

A process similar to that described in the pilot study of importing each data set into a spreadsheet for coding was implemented. A spreadsheet template housed the a priori codes that were

configured as validation lists such that human error in entering codes would be nearly eliminated. Data was then coded using the a priori codes together with emergent coding. Considering the distinct focus of Interview I from Interview II, and as in the exploratory pilot study, each interview was coded and analyzed separately. Figure 3 below presents a screenshot of the data analysis template that demonstrates the a priori codes configured as validation lists along with room for emergent codes.

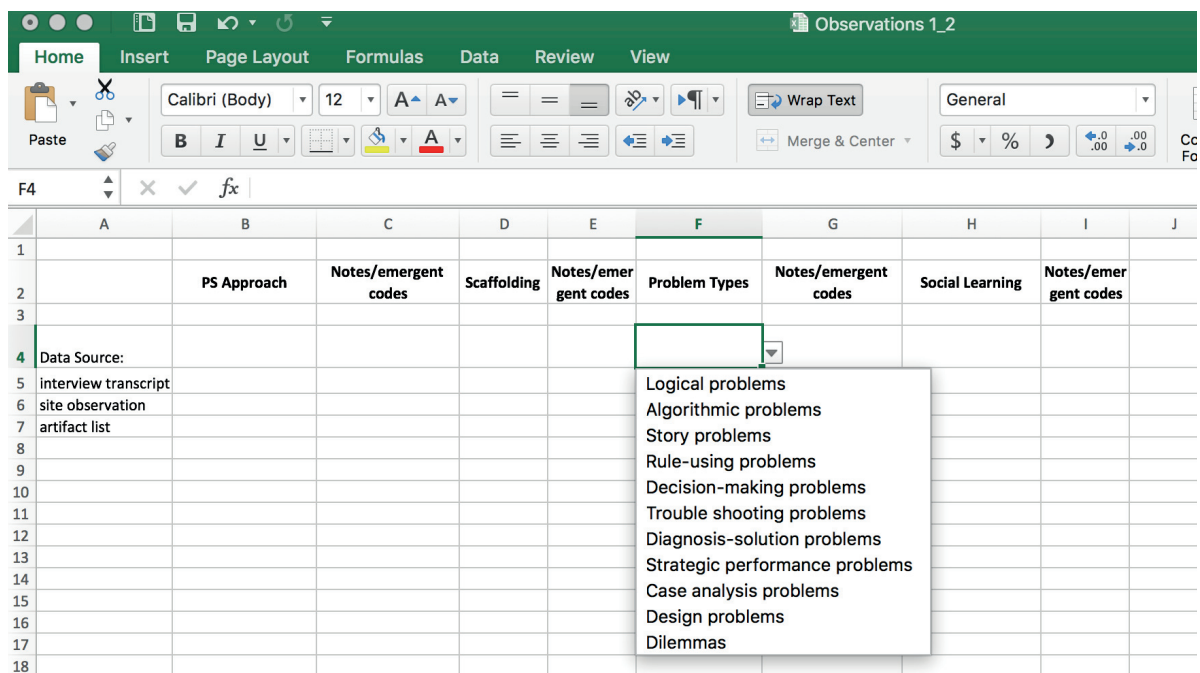


Figure 2. Data analysis template with a priori codes as validation list.

After using the hybrid process of inductive and deductive data analysis to interpret the raw data that integrated data-driven codes with theory-driven codes, the overall data was reviewed using thematic analysis with a template approach. Thematic analysis, is a search for emergent themes important to describing the phenomenon under investigation. The data is carefully read and re-read to identify themes. Patterns are recognized within the data, where emerging themes become the categories for analysis (Braun & Clarke, 2006; Daly, Kellehear, & Gliksman, 1997; Fereday & Muir-Cochrane, 2006). A template approach using codes configured as validation lists was applied to organize data for analysis and interpretation. The template was

developed a priori, along with the codes, informed by the pilot study, literature and the theoretical framework and served as a codebook for the in-depth data analysis (Crabtree & Miller, 1992). Although presented here as a linear, step-by-step process, the analysis was in fact iterative and reflexive in nature, a process of moving back and forth between the multiple data sources, codes and literature, and as modeled by Fereday and Muir-Cochrane (2006).

Trustworthiness

The overall rigor of this research was established in regards to Tracy's (2010) eight "Big-Tent" criteria for excellent qualitative research. The eight criteria present a framework to consider quality in qualitative research that is both comprehensive and flexible. Each of the eight criteria is examined in the following sections.

Worthy topic. A worthy topic is compelling because it emerges from disciplinary priorities (Tracy, 2010). This exploratory case study examined the nature of problem solving in terms of the types of problems posed, instructional approaches and collaborative learning, in first-year and second-year post-secondary computing education. This study's topic was particularly worthy in its timeliness and given the previously noted workforce demands of problem solving, teamwork, and communication as articulated by computing employers (Archer & Davison, 2008; Robles, 2012; Vivian et al., 2016). This research is also particularly relevant and significant as the United States looks to fill more than one million computer occupation openings expected from 2014 to 2024. (Fayer et al., 2017). Furthermore, this study was worthy in that it brings to light the findings around faculty enactments and perceptions of problem solving and collaborative learning in computing education that will potentially foster new computing education directions and serve the workforce demands.

Rich rigor. Rich rigor is exhibited by a study's methods, including a complexity and abundance of data, the appropriateness and thoroughness of the data, and the care and attention to the data collection and analysis (Tracy, 2010). Nearly 800 course artifacts (homework, exams, lab exercises, projects, and study guides from each course), field notes for 22 hours of instructional classroom/laboratory observations that occurred in the natural setting, and 14 semi-structured faculty interview transcripts served as the data sources for this research. Data collection processes were established by well-informed and structured procedures and included data logs, reflective researcher logs and data inventories. Previously noted and established inductive and deductive coding methods were employed followed by thematic data analysis using a template approach (Braun & Clarke, 2006; Fereday & Muir-Cochrane, 2006). These varied data sources presented an abundance of samples and analysis that presented an opportunity to see nuance and complexity that was appropriate for the research questions that focused on problems posed, problem solving approaches and collaborative learning (Tracy, 2010).

Sincerity. Sincerity in a study is apparent by its honesty and transparency in all aspects of the research (Tracy, 2010). This study was authentic and genuine in that my goals and subjectivities were transparent throughout the process, and in fact strengthened the overall research. In a self-reflexive manner, I examined my researcher subjectivities throughout the study and acknowledged their influence on all aspects of the research, from the design to the data collection and analysis. A reflective journal aided in acknowledging my subjectivities throughout the process. Field notes, data collection methods and analysis templates were clearly outlined and logged throughout the process, presenting an audit trail that was entirely transparent in

nature. In fact, the transparency in methods and analysis present an opportunity to extend and replicate the research to institutions in the future.

Credibility. Credibility may also be considered trustworthiness, and trustworthiness was established for this study through data triangulation of multiple data sources (Tracy, 2010). These sources included faculty interviews, instructional field-site observations, and course artifacts. Triangulation of the three data sources allowed for examination of how themes were represented in regards to problem solving and collaborative learning. By analyzing various sources of convergent evidence, the data sources provided multiple measures of the same phenomenon within the context of problem solving and social constructivist theory (Yin, 2013). Concrete details as examples of findings are presented in the findings as a means of showing rather than simply telling, and provide additional credibility to the research. Here again, my own researcher subjectivities and domain knowledge presented additional credibility in enabling me to understand the tacit issues noted during data analysis that allowed me to more deeply make sense of the data.

Resonance. Resonance, or evoking an effect on an audience, is achieved through research transferability (Tracy, 2010). The methods presented in this research are transferable to other post-secondary computing programs as well as programs outside of computing, inducing in the reader a sense of how this research might overlap with a scenario at their own environment. More specifically, examination of problem-types, problem-solving approaches and collaborative learning at other post-secondary programs including liberal arts colleges, public universities and community colleges would resonate with computing faculty who are dedicated to their students' learning, and likely be of particular interest in developing future computing curriculum and informing a computing education research framework.

Significant contribution. Significance of the research may be gauged in terms of the current climate of knowledge or practice (Tracy, 2010). Recalling the call for problem solving, teamwork and communication by the workforce, this study was significant in addressing how computing educators prepare students for those demands. Of particular importance was considering the differences of how novices and experts approach problems, and how the computing faculty, situated as domain specific experts and PCK experts, are preparing students who, as novice learners generally lack an established knowledge-base. More broadly, this study was significant because understandings of how computing education faculty perceive and approach problem solving and collaborative learning in the computing education domain may yield insight on practices that either promote or hinder learning and understanding in computing education. These understandings are specifically important to extending the codifiable base of knowledge in computing education research due to the nature of this domain of learning which hinges on application of knowledge and skills in the workforce. Given the limited practice of theoretical frameworks in computing education research, this study was also theoretically significant in its contribution towards an emerging computing education research framework that informs on the enactment of collaborative learning within the context of problem solving in computing education.

Ethics. Ethics serve as a universal end goal of qualitative quality, and this research study followed the ethical guidelines inherent in qualitative research (Tracy, 2010). Procedurally, institutional protocols for the welfare of human research subjects recruited to participate in research were followed. Data was secured in a locked office and on a password protected laptop. Participant identities were guarded by using pseudonyms. The research goals were described to

the participants in a manner that presented overarching transparency and all their questions were answered in a way to make them comfortable with the research.

Meaningful coherence. Meaningful coherence in a study centers on achieving the stated purpose, using appropriate methods and interconnecting the literature throughout the entire process (Tracy, 2010). This study exhibited meaningful coherence in that it achieved the initial goal in regards to examining the problem types, problem solving approaches and collaborative learning in post-secondary first-year and second-year computing education courses. The research questions were developed in the context of the problem-solving literature, a problem-typology and a social constructivist theoretical framework that also guided the overall study design and analysis, as well interpretation of the findings. A priori codes used for data analysis and interpretation were informed by literature as well as the pilot study. Throughout the process, the data was reflexively examined in order to draw conclusions that were grounded in the literature.

Chapter 4

Findings

The findings are organized to reflect the three research questions. Qualitative data analysis from instructional classroom and laboratory observations, in-depth faculty interviews and course artifacts revealed three key findings: (1) problems posed to students in course artifacts and classroom instruction predominantly presented well-structured characteristics, with faculty interviews emphasizing the importance of real-world problems and allowing for multiple solution paths while focusing on better or worse solutions, (2) consistent with course artifacts, instructional approaches to problem solving were also well-structured, with faculty implicitly modeling problem decomposition to students and (3) collaboration was informally modeled through instructor-student interactions in the classroom along with some limited and unstructured student-student group work in solving problems, with faculty focusing on building individual student skillsets.

The Types of Computing Problems Posed by Undergraduate Computing Faculty

For each of the three data sources, the six problem-solving typology attributes (learning activity, problem inputs, success criteria, problem context, problem structure and problem abstraction) as defined in Jonassen's (2000) typology were analyzed across problems types and across the three data sources for each of the seven courses included in the study. As such, nearly 800 course artifacts including homework, exams, lab exercises, projects, and study guides from each course, field notes for 22 hours of instructional classroom/laboratory observations that occurred in the natural setting, and 14 semi-structured faculty interview transcripts were analyzed.

In considering well-structured and ill-structured problem types via Jonassen’s (2000) problem-solving typology, algorithmic problems, rule-using problems, troubleshooting problems and design problems emerged as dominant across all data sources; this includes explanations and descriptions from faculty interviews, classroom observations of enactment of classroom teaching and laboratory sessions, and examination of course artifacts. While there were obvious elements that characterized the problems as algorithmic, rule-using, troubleshooting, and design problems, it must be noted that the conceptions related to these problem types and the specific attributes remained incomplete. In other words, three of the six attributes from Jonassen’s (2000) problem-solving typology were emphasized; these included (a) learning activity, (b) success criteria and (c) context. As such, these three attributes serve as a foundation in presenting the problem type findings. The following excerpt in Table 12 from Jonassen’s (2000) problem-solving typology table highlights the four emergent problem types along with the three prevailing attributes and generalized examples from the literature.

Table 12
Four Emergent Problem Types

	Problem Types			
	Well-structured		Ill-structured	
	Algorithmic problems	Rule-using problems	Troubleshooting problems	Design problems
Learning activity	Procedural sequence of manipulations; algorithmic process applied to similar sets of variables; calculating or producing correct answer	Procedural process constrained by rules; select and apply rules to produce system-constrained answers or products	Examine system, run tests, evaluate results, hypothesize and confirm fault states using strategies (replace, serial elimination, space split)	Acting on goals to produce artifacts, problem structuring and articulation
Success criteria	Answer or product matches in values and form	Productivity (number of relevant or useful answers or products), multiple solution paths, procedures or methods	Faults identification, efficiency of fault isolation	Multiple, undefined criteria, no right or wrong, only better or worse

Context	Abstract, formulaic, presented to students using a finite set of procedures	Purposeful academic, real-world, constrained purpose	Real world, technical, mostly closed system, also known as fault state diagnosis problems	Complex, real world, degrees of freedom, limited input and feedback
Examples	Cooking recipes, long division or equation factoring problems, converting temperatures from Fahrenheit to Celsius, bisecting a given angle	Games such as cards or chess, expanding a recipe for more guests, completing tax forms, proving isosceles triangle angles are equal, calculating building materials, changing grammar to reflect a subjective case	Fixing a car, debugging a computer system, identifying committee communication breakdowns, diagnosing diminished dairy farm production, understanding why roof trusses are showing premature stress	Designing an electronic circuit, a house, a new restaurant entrée, composing a piece of music or an essay or any other product or system

Specific to this study, problems posed to students predominantly presented well-structured aspects, with findings revealing characteristics spanning the attributes of four problem types in Jonassen's (2000) problem-solving typology. Algorithmic, rule-using, troubleshooting and design problem characteristics emerged as dominant across all three data sources. However, in explicitly examining the three emergent attributes for each problem type (learning activity, success criteria and context), findings around the success criteria attribute revealed a tension between how faculty articulated this idea during interviews versus actual enactment and artifacts. Course artifacts and instructional observations revealed that faculty posed problems where the success criteria specifically required a correct answer as with algorithmic problems. However, during interviews, faculty emphasized the importance of multiple solutions, ideal approaches and better or worse solutions, as notable in both well-structured rule-using problems and ill-structured design problems. A similar finding emerged around the problem context attribute. Course artifacts and instructional observations revealed the use of both abstract problems (as found in algorithmic problems) and real-world problems (as with rule-using, troubleshooting and design problems); yet during interviews, faculty omitted the value of abstract problems and instead described the importance of using problems in a real-world situated context.

This section is organized to reflect the major findings for research question one related to problem types. First, the major problem types as revealed in course artifacts are presented. Second, the major problem types observed during classroom lectures and laboratories are presented, followed by the major problem types revealed through faculty interviews. The three emergent attributes of learning activity, success criteria and context are examined for each problem type by data source. These problem type findings reflected data for seven computing courses.

Problem Types in Course Artifacts

In considering problem-type analysis, the examination of course artifacts revealed evidence of the use of algorithmic and rule-using problems across all seven courses. Still significant, yet to a slightly lesser extent, there was also evidence of troubleshooting and design problems. All of the course artifacts emphasized the three attributes of (a) learning activity, (b) success criteria and (c) context for both algorithmic and rule-using problems.

Algorithmic and rule-using problems in course artifacts. As algorithmic and rule-using problems share several overlapping characteristics, they will be examined jointly in regards to course artifacts. For algorithmic problems, the learning activity encompassed the use of a “procedural sequence of manipulations” as well as an “algorithmic process applied to similar sets of variables” and “calculating or producing [a] correct answer”. Rule using problems, similar in nature regarding the learning activity attribute, are defined by a “procedural process constrained by rules,” whereby rules are selected and applied “to produce system-constrained answers or products.”

In considering the success criteria attribute, for algorithmic problems, the answer or product is expected to match in value and form while rule-using problems specify that there are a

number of relevant or useful answers. In the case of the artifacts examined, this commonly presented as a single successful, relevant or useful answer as with algorithmic problems. In terms of context, the documents presented both abstract/formulaic problems as with algorithmic problems alongside purposeful academic, real-world, constrained examples and scenarios as with rule-using problems.

One example of a problem that aligned with the criteria for algorithmic problems included the content around search and sort algorithms that was presented as a rigid set of procedures or steps as well as a set of rules to be followed. This is exemplified in Figures 5, 6 and 7 below. These course lecture notes from Dave's class, which guided the faculty member during class and were also available to students, demonstrated how the instructor explained the difference between a selection sort and an insertion sort by walking through the algorithm in a step-by-step fashion. These course artifact notes, identified as Figures, 5, 6 and 7, delineated which algorithm was appropriate to use in particular situations and aligned closely with the characteristics of algorithmic problems where a rigid set of steps is applied to similar scenarios to produce a correct answer.

Day25 – Sorting and Searching

Sorting is the rearranging of data (often in an array, `ArrayList`, or `Vector`) so that all of the items are in a particular order. We will start with sorting a simple array of `ints`, and progress to other types of data.

Consider this array of `ints`:

```
private int[] list = { 5, -1, 10, 23, 4, 8, 2, 25, 7, 0 };
```

There are several ways to go about this. We will explore three approaches.

Selection Sort

Suppose we wish to sort this list into increasing order. The idea is simple.

- **Search** for the smallest item in the list (elements `0..(length-1)`).
- Once we have **selected** it, swap it with the item at the start of the list.
 - Now, the smallest item is in element `0`.
- Now, **select** the smallest item in elements `1..(length-1)`.
- Swap it with the item in element `1`.
 - Now we have the two smallest items in elements `0` and `1`, in the proper order.
- Continue until we have processed all elements.

We will represent the list at the start of this process (before any swapping) as:

```
| [ 5] ->-1 10 23 4 8 2 25 7 0
```

- The `|` indicates that everything to its LEFT is already sorted.
 - There is nothing to its left, since we are just starting out.
- The brackets around `'5'` indicate that it is our 'candidate' for the smallest value.
 - We have to start somewhere, so we start with the 1st element as the candidate

Figure 4. Algorithm Problem Example – Selection Sort

Bubble Sort

Bubble sort is similar, but uses a different approach. On each pass through the list, it looks at adjacent **pairs** of numbers. If they are out of order, they are swapped. This has the effect of the smaller values **'bubbling'** toward the start of the list and the larger values **bubbling** toward the end of the list.

Here is the same list as above, pictured for Bubble Sort, pass 1. In what follows, the brackets '[' are around the pair of numbers being checked.

[5 -1] 10 23 4 8 2 25 7 0

Here, 5 and -1 are out of order, so they are swapped, as shown in the next line.

-1 [5 10] 23 4 8 2 25 7 0

Now, the 5 is checked against the 10. They are in proper order, so no swap occurs (next line).

-1 5 [10 23] 4 8 2 25 7 0

Next the 10 and 23 are checked, and so on, until all pairs have been checked. When we find two out of order, we swap them.

-1 5 10 [23 4] 8 2 25 7 0

-1 5 10 4 [23 8] 2 25 7 0

-1 5 10 4 8 [23 2] 25 7 0

-1 5 10 4 8 2 [23 25] 7 0

-1 5 10 4 8 2 23 [25 7] 0

-1 5 10 4 8 2 23 7 [25 0]

-1 5 10 4 8 2 23 7 0 25

pass 1 complete

-1 5 10 4 8 2 23 7 0 25

Figure 5. Algorithm Problem Example – Bubble Sort

Insertion Sort

This sort takes an unsorted stream of numbers and sorts it by inserting each number in the location that it belongs. This requires an output list that is built, one element at a time. An `ArrayList` is especially good for this, as it has the method `list.add(index, value)` which places the value at the indicated index of the list. If there is already an element at that index, it is moved to `index+1`, and so on, to the end of the list.

The output for this is fairly simple. Each line may begin with a number in angle brackets (`<-1>`). This is the number to insert. The rest of the line is the list it will be inserted into. In the list of numbers, you will see a number in brackets `'[]'`. This number is the one being checked against the new number being added. When the number is inserted, the list is printed with the number in angle brackets moved to its correct place.

To Insert	List →	
	<code>< 5></code>	Initially the list is empty and the first number (5) is put at the start.
<code><-1></code>	<code>[5]</code>	
	<code><-1></code>	5
<code><10></code>	<code>[-1]</code>	5
<code><10></code>	<code>-1 [5]</code>	5
	<code>-1</code>	5 <code><10></code>
<code><23></code>	<code>[-1]</code>	5 10
<code><23></code>	<code>-1 [5]</code>	10
<code><23></code>	<code>-1 5 [10]</code>	
	<code>-1</code>	5 10 <code><23></code>
<code>< 4></code>	<code>[-1]</code>	5 10 23
<code>< 4></code>	<code>-1 [5]</code>	10 23
	<code>-1 < 4></code>	5 10 23

Figure 6. Algorithm Problem Example – Insertion Sort

These three course artifacts clearly indicated algorithmic problems in that a procedural sequence of manipulations is applied to similar sets of variables to produce a correct answer. Further, is the idea being imparted to students around when each searching and sorting algorithm was appropriate to use, and also aligning with the procedural nature of algorithmic problems. For example, as noted in Figures 8, 9 and 10 below, while bubble sorts were explained in the notes as more appropriate for small and mostly sorted data sets, insertion sorts were more appropriate for sorting data that is external to the program being written and use twice as much memory.

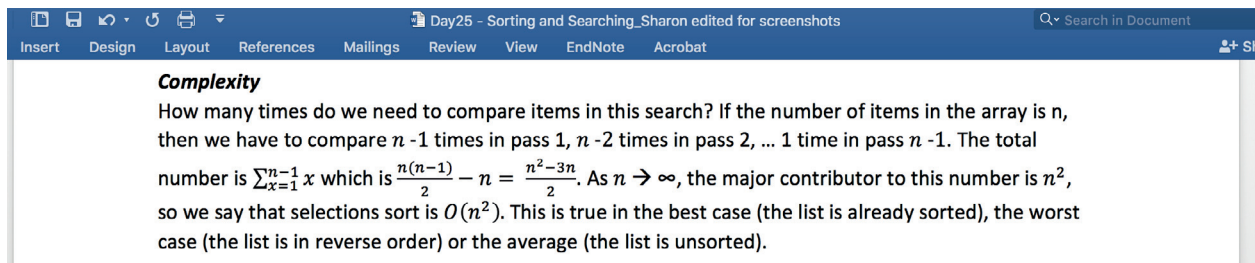


Figure 7. Selection sort complexity and use.

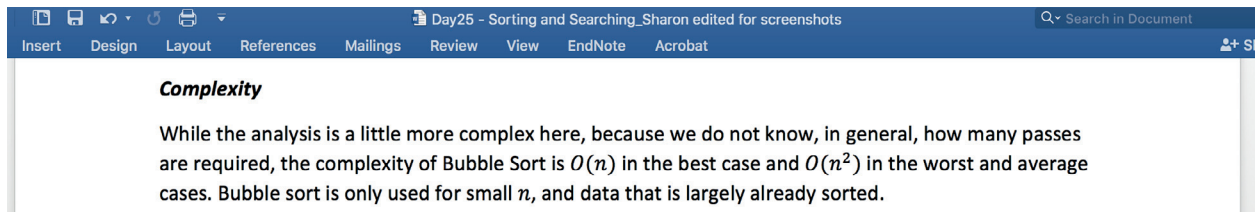


Figure 8. Bubble sort complexity and use.

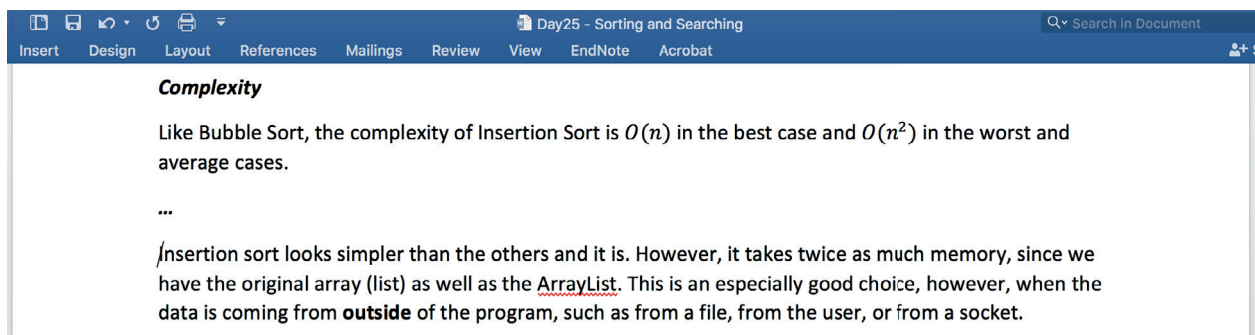


Figure 9. Insertion sort complexity and use.

This algorithmic problem example of sorting from Figures 5-10, which demonstrated the instructional content of searching through a list and swapping items in order to sort them appropriately, clearly illustrated the steps presented to students as a finite set of procedures and as inherent in algorithmic problems. Also as with algorithmic problems, this example elucidated inputs as the context of the problem that were abstract, formulaic and procedural in nature with an answer matching in value to an expected product, in this case a sorted list. Finally, this example presented an explanation of when it was appropriate to use one sorting algorithm over another, again reflecting the characteristic of procedural and formulaic inputs with expected or predictable products.

Another example of an algorithmic problem emerged from the artifacts from Sue's course. In this course, algorithmic steps and procedures were evident in the course artifacts used to present the content in the form of steps or procedures to students during a traditional lecture as well as the procedural reference sheets, the practice exercises and the homework that prepared students for similar exam questions. The slide presentation page in Figure 11 below identified a summary of the overarching steps involved in the lifecycle development for a database.

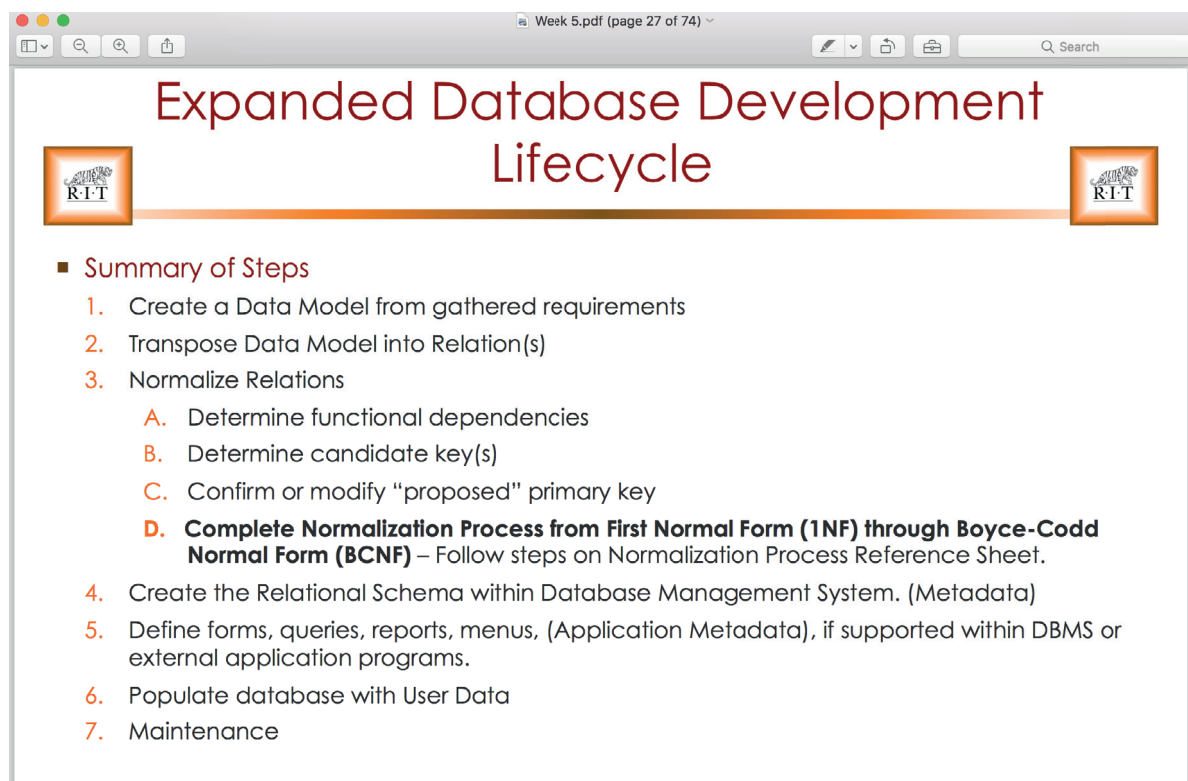


Figure 10. Slide presentation demonstrating procedural steps.

Each of these steps was further expanded in subsequent slides. Two noteworthy steps were (2) transpose data model into relations and (3) normalize the database. Each of these yielded yet another set of specific steps and is demonstrated respectively in Figures 12 and 13 below.

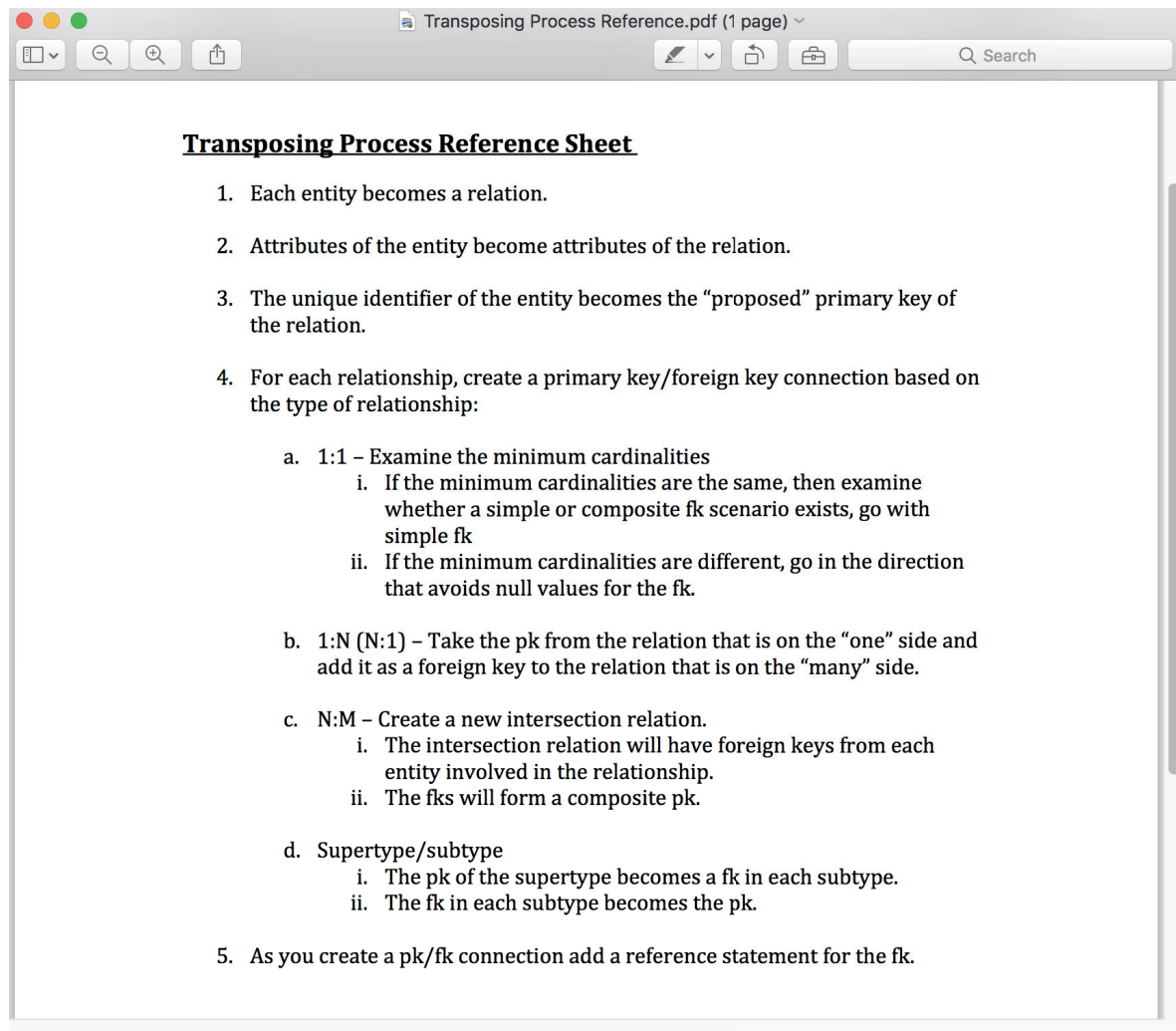


Figure 11. Slide presentation demonstrating expanded procedural steps.

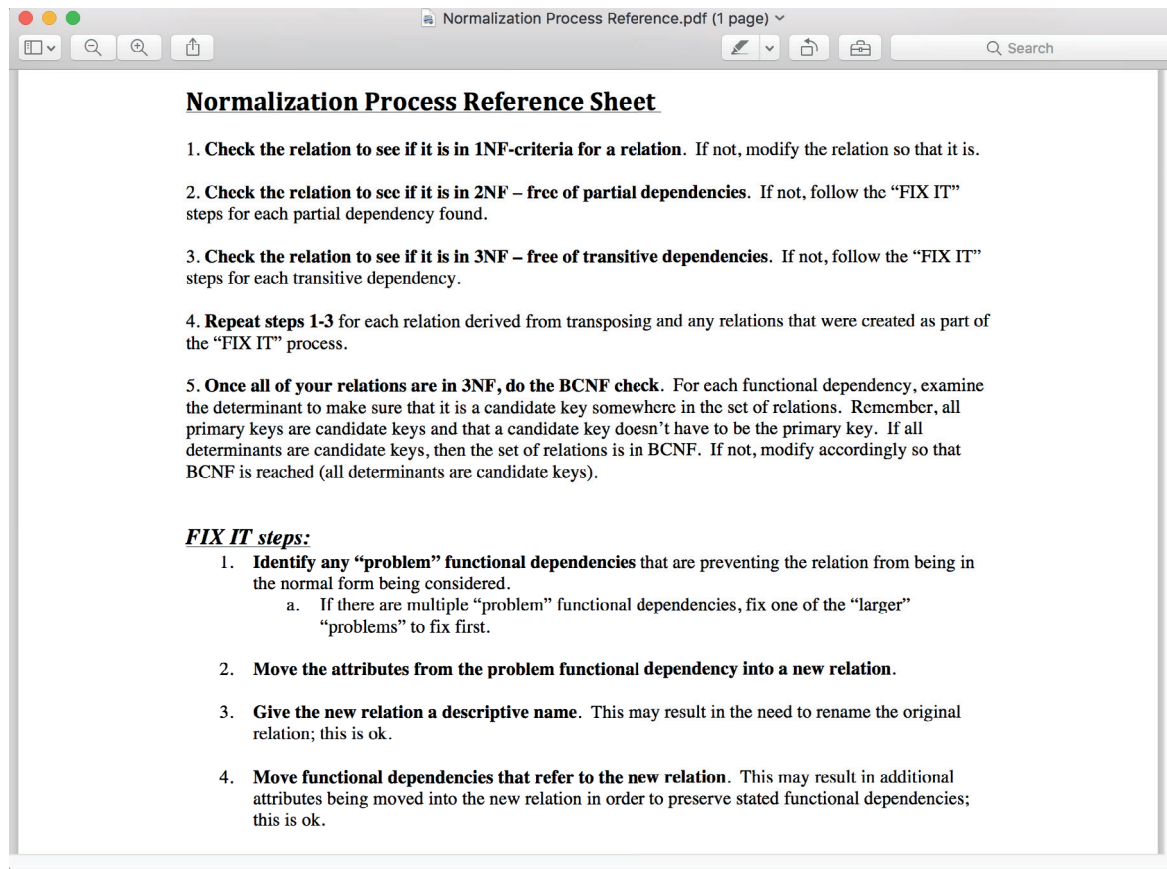


Figure 12. Slide presentation demonstrating expanded procedural steps.

To clarify, transposition is the process of migrating a conceptual database model to one that can be implemented in the system, while normalization is the process of organizing data in a database such that there is flexibility and consistency while eliminating redundancy. Both of these reference sheets clearly outlined each of the steps required for the transposition and normalization processes to be followed as part of the database development lifecycle. Again, the course artifact materials represented procedures contextualized as abstract and formulaic in nature and clearly leading to a predictable answer.

In continuing with the normalization artifacts from Sue’s course, the practice exercise represented in Figure 14 below carried the example further by demonstrating the

processes required as part of the database normalization as well as the expected correct outcome. The answer key, provided for instructors, clearly indicated a specific answer or product matching in value to that presented in the key, another key attribute noted as the success criteria for algorithmic problems.

PE5 [Compatibility Mode] Search in Document

Sign Layout References Mailings Review View EndNote Acrobat

Practice Exercise # 5 – Normalization through 2NF

Name: _____

For each problem below, given the original relation and functional dependencies, normalize the original and all resulting relations to 2NF. Be sure to use proper relational notation: RELATION(pkattr, attribute, fkattr). Include reference statements for foreign keys.

Problem #1

Emp_ID	Name	Dept_Name	Salary	Course Title	Date Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

EMPLOYEE2(Emp_ID, Name, Dept_Name, Salary, Course Title, Date Completed)

Functional Dependencies:
 Emp_ID, Course Title → Name, Dept_Name, Salary, Date Completed
 EmpID → Name, Dept_Name, Salary

YOUR ANSWER (Final set of relations normalized to 2NF):

Page Break

Figure 13. Exercise demonstrating required process with expected correct outcome.

Of particular note in both the sorting algorithm and the database normalization examples, is that while each concept is initially presented as a formulaic and abstract construct inherent in algorithmic problems, they were both later presented in the artifacts as specific examples, contextually grounded in a realistic scenario and inherent in rule-using problems.

In exemplifying rule-using problems, of note is that the course artifacts presented very detailed and clearly stated problem goals as the overarching learning activity with rules applied to produce constrained answers or products in the form of (a) required features and (b) detailed grading rubrics. One example of a problem that included these learning activity attributes of rule using problems was noted in Chris' course. This problem included a clearly stated goal and constrained purpose by requiring a highly-specified number of nodes as well as prescriptive error messaging. In the following semester project requirements document presented as Figure 15, students were instructed to accept a specific number of nodes and edges to be input to a program, and then print a specific error message before terminating if the number of nodes was not correct. Students were then instructed to read in specific values and test them while only supporting three-digit numbers (0-999).

Semester_Project

ences Mailings Review View EndNote Acrobat

Input Specification ¶

The graph will be provided to you on STDIN (i.e., as if a user were typing it in at the console). A general note on input, as a rule, the number of inputs provided to your program, and their types will be correct. The errors that you must detect are with the value of these inputs. ¶

The input will begin with the total number of nodes n as a single integer. Now normally a graph can have any positive number of nodes, but for our program, you only need to be able to handle graphs whose total number of nodes are in the range 1 to 20 (inclusive). **Note:** although we input an integer from 1-20 for the number of nodes, the nodes in our program will be labeled from 0-19. So, the first node will be node 0 and the last node (if there are all 20) will be node 19. If the number of nodes in the graph isn't within the legal range, then your program should print the following error message and terminate. ¶

```
Invalid number of nodes. Must be between 1 and 20. ¶
```

Error message 1 ¶

After your program has read in the number of nodes (assuming that number is legal) you should then read in a single integer m that represents the total number of edges in the graph. As our graph can only have a maximum of 20 nodes the maximum number of edges allowed is n^2 or 400 (self-loops are allowed) and the minimum is 0. If the number of edges in the graph isn't within the legal range, then your program should print the following error message and terminate. ¶

```
Invalid number of edges. Must be between 0 and 400. ¶
```

Error message 2 ¶

Next, your program will read in the source, destination, and weight for all m edges. The source and destination must fall in the range 0 to $(n-1)$ and the weight must be greater than 0. You program will not have to test weights against a maximum value. With that said, our output format can not support more than 3-digit numbers for edge weights, so we guarantee that we will not test your program with any edge weight greater than 999. ¶

If the source node is invalid you should, immediately after reading it, print the following error message and terminate. ¶

Figure 14. Semester project requirements demonstrating detailed goals.

As with both algorithmic and rule-using problems, the output in terms of success criteria for this project was also highly specified. As demonstrated in the figure 16, students were required to complete the project with an output that included a blank line, followed by the graph

which was to be formatted as an adjacency matrix with a visual example of the output provided, including specific spacing and blank line requirements for the output.

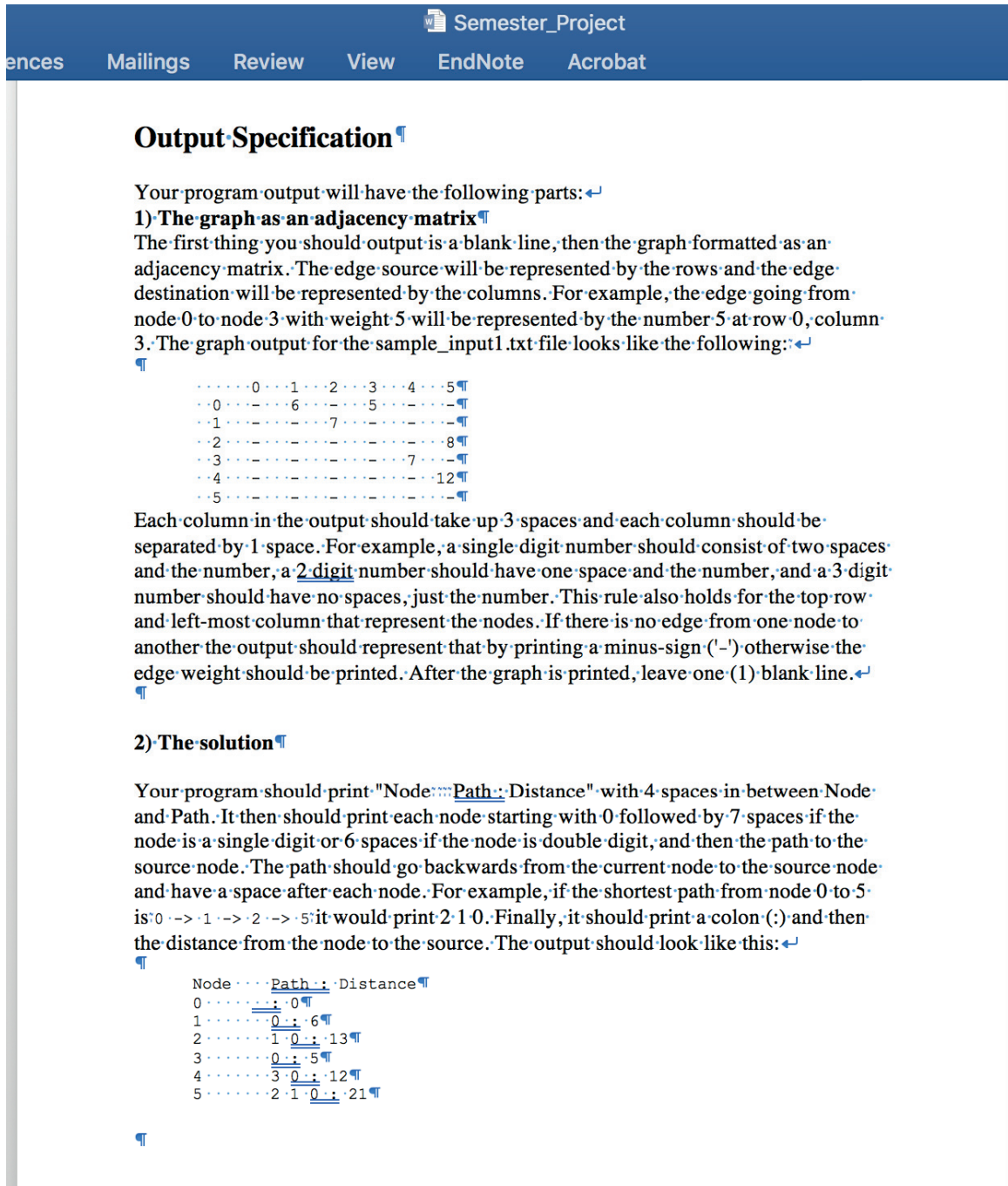



Figure 15. Demonstrating detailed output requirements as success criteria.

The following lab assignment artifact from Bill’s course also reflected the attributes of a rule using problem. Figure 17 below presented the highly prescriptive and specific requirements

for the battleship game that students were to develop. The learning activity focused on clear goals to be integrated as procedural processes and rules were specified regarding the digital game board, rules of play, graphical display, files to be included and network compatibility.

Lab 9 - BattleShip



Introduction

For this week's lab you will be creating a playable version of the battleship game. It will include the following features:

- Utilizes a 10x10 grid
- Each player has 5 ships of size {5, 4, 3, 3, 2}
- A graphical display of the game board for a single player
- A console on the GUI that is used for status messages
- Reads the initial board setup from a file
- All logic necessary to play a game of battleship
- Works over the internet to allow players on separate machines to play each other

Starter Code

A starter project has been provided for you. The GitHub link for it is specific to your section and will be provided by your instructor. Once you have the initial project you can proceed with setting up your file structure. Your source directory should be structured as follows (Red indicates provided files):

```

controller
  commands
  responses
  BattleShip.java
  Client.java
  CommsManager.java
  ConsoleWriter.java
  ModelActions.java
  
```

Figure 16. Demonstrating highly prescriptive goals while allowing for multiple solution paths.

Both of these examples presented a problem description with a highly-stated goal including a contextualized constrained purpose, while still allowing for multiple solution paths, procedures and methods to be used at the discretion of the problem solver.

As previously noted, grading rubrics were also apparent in the course artifacts as a mechanism to present the highly specific project requirements that produce system-constrained answers or products inherent in rule-using problems. The homework from Dave's course presented the following grading rubric in Figure 18, which dictated specific file names, error messages, and programming constructs. The rubric also allowed 10 points for the project

displaying and running as expected, indicating that a specific output was expected, an attribute matching both algorithmic and rule-using problems.

Grading topics	Max Pts	Pts Earned
Displays and runs as expected.	10	
Style:		
• JavaDocs for every class and method	5	
• Follow coding style: indentation, use of white space for readability – see Coding standards posted in MyCourses conference for this section	5	
• Variables of appropriate access (private)	3	
Other Extra credit items:		
Extra:		
• Menu controls: Restart, choosing number of racers...	+3...+10+	
• Usable controls: such as sliders for speed, etc...	+3+	
• Dynamic graphics drawn vs. icon	+3+	
• Animated (changing) 2D drawn graphic that moves	+5+	
Other items:		
Ways to lose points: (doing these will deduct indicated points)		
• Program <i>not</i> named Races.java	-5	
• Image file not named races.gif or races.jpg	-2	
• Using Timers vs. required Threads	-15	
• Incorrectly sending your work to dropbox	-3	
• Program does not compile – Grade of zero	-100	
• Program does not have adequate documentation	-5..-10	
• JavaDoc warnings	-2	
• Errors show up on command window	-1 to -5	
• Program shows a run time (execution) error	-5	
Total Points	100	

Note: JavaDocs are required for this homework
Additional Comments:

Figure 17. Grading rubric demonstrating required specific output.

Noteworthy is that this rubric presented in figure 18 also required students to display error messages in an output window. Error messages serve as a common form of debugging problems in computing, and as such present an aspect of troubleshooting problems. This will be examined further in the findings around troubleshooting problems.

Another example from Bill’s course artifacts exemplified the use of protocols in problem solving, as seen in Figure 19 below. The document highlighted that a protocol is a specification

for the format and content of messages sent via a network to allow to communication. In computing, these protocols serve as a set of rules for the communication, and as such explicitly

2 Problem Solving

Recall that a protocol is a specification for the format and content of messages that are sent from one computer to another over a network in order to allow the computers to communicate with each other.

Students will be organized into groups, and together with a problem solving team, will design and partially implement an instant messaging protocol. As you find the solution to each of the following problems, discuss your solution with your instructor or SLI.

2.1 The Protocol

On a whiteboard or using a pencil and paper, design and implement a protocol using plain text (ASCII characters). For each message described, consider some or all of the following:

- Which user is the message from?
- To whom is the message directed?
- Many messages will have multiple parts; how will you tell where one part ends and the next begins?
- How will you know when you have reached the end of a message?

Client-to-Server Messages

Description	Message Format
User 'Jodi' connects to the server	
User 'Joe' sends broadcast "Hi!" to server	
User 'Jane' sends direct message 'Hello!' to Joe	
User 'John' requests a list of currently connected users	
User 'Janet' disconnects from the server	

Figure 18. Exercise demonstrating procedural processes constrained by rules.

present problems where procedural processes are constrained by rules. As demonstrated in Figure 19 above, students were asked to design a protocol to meet the requirements set forth in an instant messaging program when a user connects to a server and then receives a response followed by a request for a list of connected users and a disconnection from the system. This instant messaging protocol example is also clearly a real-world example, noted in the context of rule-using problems, as protocols such as this one are required to enable computer network communications systems.

Another course artifact exemplifying rule-using problems is demonstrated in Figure 20 below. In this series of slides, the Dynamic Host Configuration Protocol (DHCP) was presented to students as a learning activity with a series of rules serving as part of a procedural process.

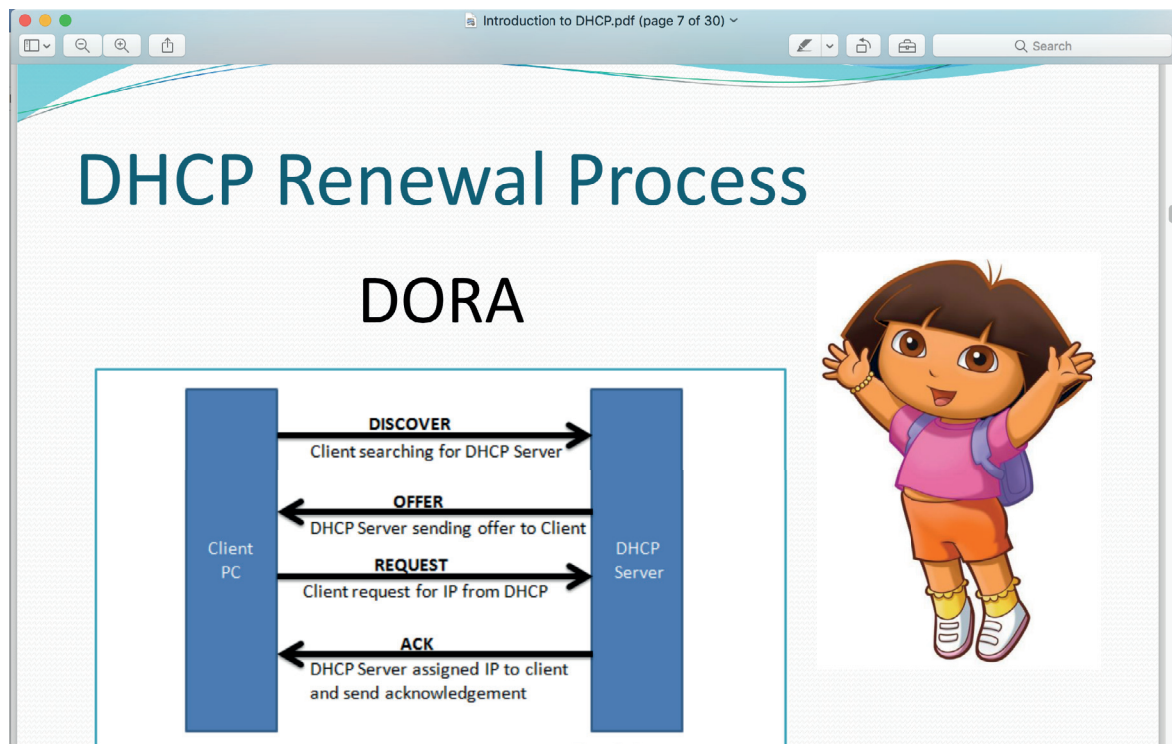


Figure 19. Course slides demonstrating rules serving a procedural process.

Here, a series of messages are expected from each device, named as either the client PC or the DHCP server, in a particular order. Each message has a specific purpose and the format of the

message, as outlined in the slide presented in Figure 21 below is of significant importance.

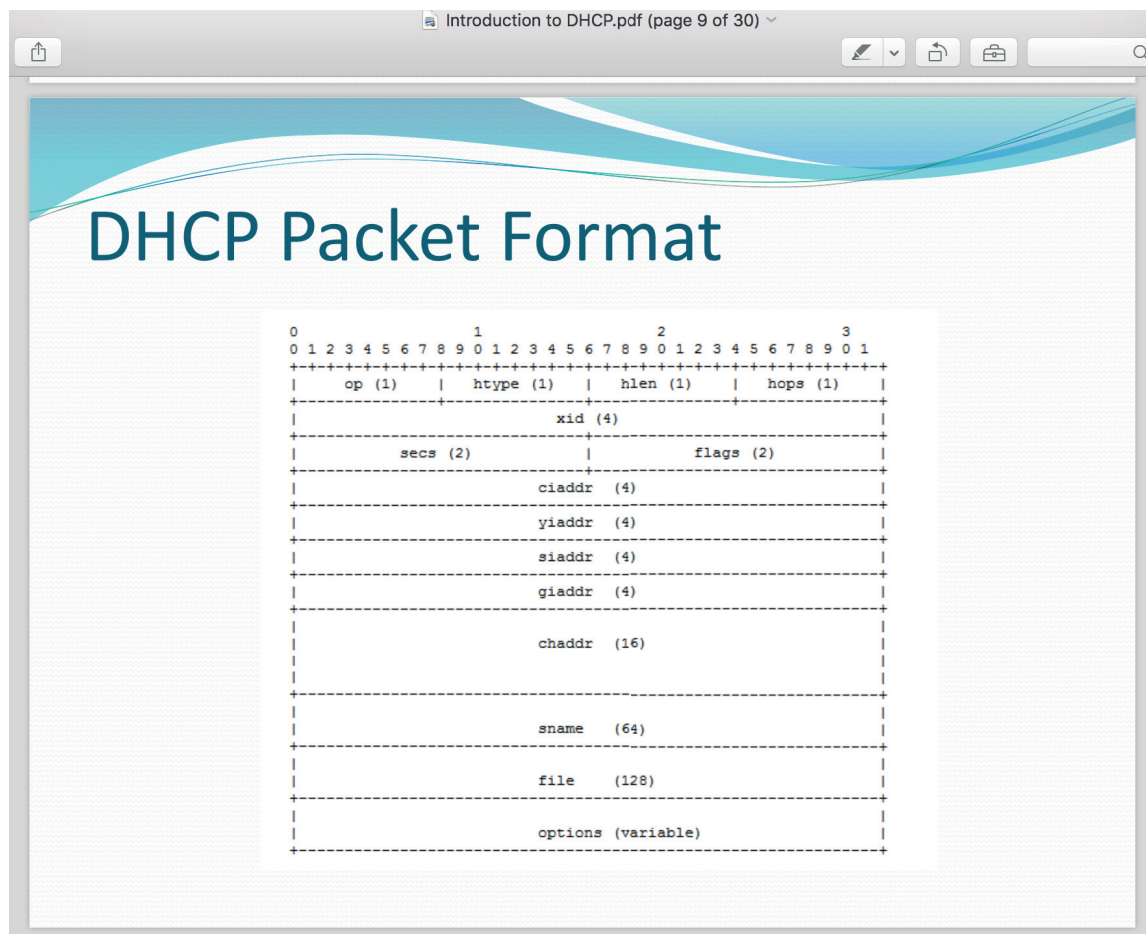


Figure 20. Course slide demonstrating protocol specific format as part of the rules.

The DHCP protocol, which is itself a series of rules for the communication between devices, stipulates the message format that includes each of the identifiable fields and the required sizes in parenthesis for each of the fields.

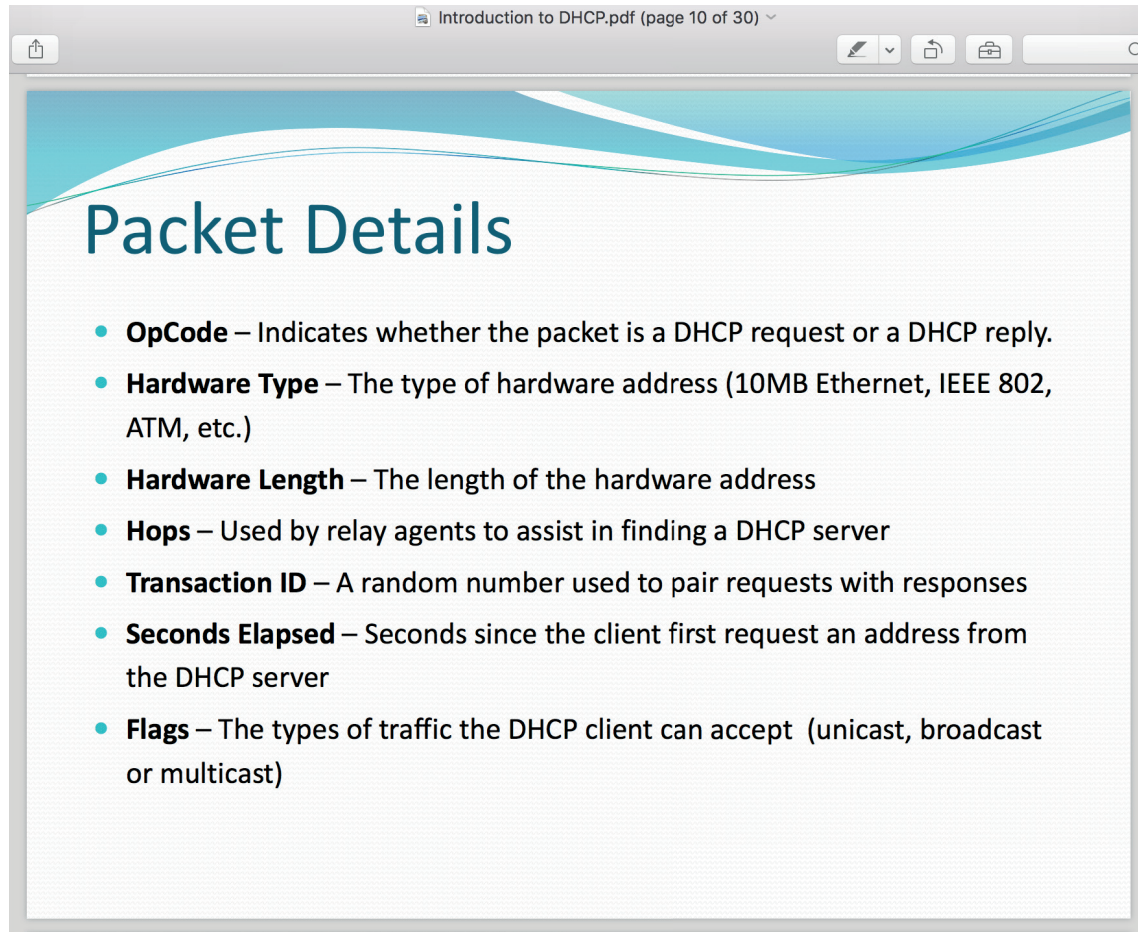


Figure 21. Course slide demonstrating protocol specific format as part of the rules.

As seen in Figure 22 above, each of the fields has a particular purpose and that is not to be deviated from when establishing a connection between the devices in order to produce a system with constrained answers or products, and as part of the learning activity. As with rule using problems, this contextually real-world scenario is used by nearly all devices connecting to any network anywhere in the world in order to establish communication.

Table 13 below summarizes the findings from the course artifact data in regards to the closely aligned algorithmic and rule-using problems.

Table 13
Course Artifacts - Algorithmic and Rule Using Problems

	Algorithmic problems	Rule-using problems
--	----------------------	---------------------

Learning activity	Procedural sequence of manipulations; algorithmic process applied to similar sets of variables; calculating or producing correct answer	Procedural process constrained by rules; select and apply rules to produce system-constrained answers or products
Findings	Artifacts: 7/7 courses supported	Artifacts: 7/7 courses supported
Success criteria	Answer or product matches in values and form	Productivity (number of relevant or useful answers or products), multiple solution paths, procedures or methods
Findings	Artifacts: 7/7 courses supported	Artifacts: 7/7 courses supported
Context	Abstract, formulaic	Purposeful academic, real-world, constrained
Findings	Artifacts: 7/7 courses supported	Artifacts: 7/7 courses supported

Troubleshooting problems in course artifacts. Evidence of troubleshooting problems in course artifacts focused on debugging tools. Through computer debugging, five of the seven courses' artifacts addressed the troubleshooting learning activity of examining a system, running tests, and evaluating results while hypothesizing and confirming fault states. The two sets of course artifacts that did not include troubleshooting, either explicitly or implicitly, were focused on logical data modeling rather than implementing programming code or computing systems. Of the five courses where troubleshooting was apparent, one course presented an entire experiment explicitly focused on debugging, while the others required students to inherently integrate debugging aspects such that the problems could in the end be completed or solved.

The success criteria attribute in troubleshooting problems is identified through fault identification and efficiency of fault isolation. This fault identification was primarily implicit in the course artifacts where computer programs and system implementations necessitated debugging in order to produce an answer or product matching in value or a number of relevant products as with algorithmic and rule-using problems. All five of the courses integrating troubleshooting into the artifacts did so through the use of real-world problems, as was also the case with rule-using problems.

Specifically, troubleshooting problems require the solver to engage in a learning activity that requires them to examine a situation, evaluate possible treatments, apply treatment(s) and

monitor the situation. The grading rubric previously presented in Figure 18 clearly indicated that students were required to inherently and continuously examine the situation while they were programming their assignment. Furthermore, this use of error messages was specifically intended for the students to be able to apply treatments to a problem situation by making corrections to their programming projects. This in turn implicitly required students to monitor the situation as part of the project to come to a final project solution.

None of the course artifacts revealed an emphasis on confirming fault states using specific strategies; however, one course artifact did explicitly walk students through the troubleshooting techniques. As demonstrated in Figure 23 below, Chris' course required students to work through troubleshooting in the form of a debugging exercise. The exercise began by explaining the primary goal and objective for the experiment, which was for students to build the necessary skills for troubleshooting their programs. A secondary goal was identified as familiarizing students with a series of errors that can be used for troubleshooting via debugging. Additionally, as indicated in the Overview, students were required to fix the bugs or errors. This aligns with the learning activity for troubleshooting problems whereby the systems are examined, tests are run, results are evaluated and faults states are hypothesized.

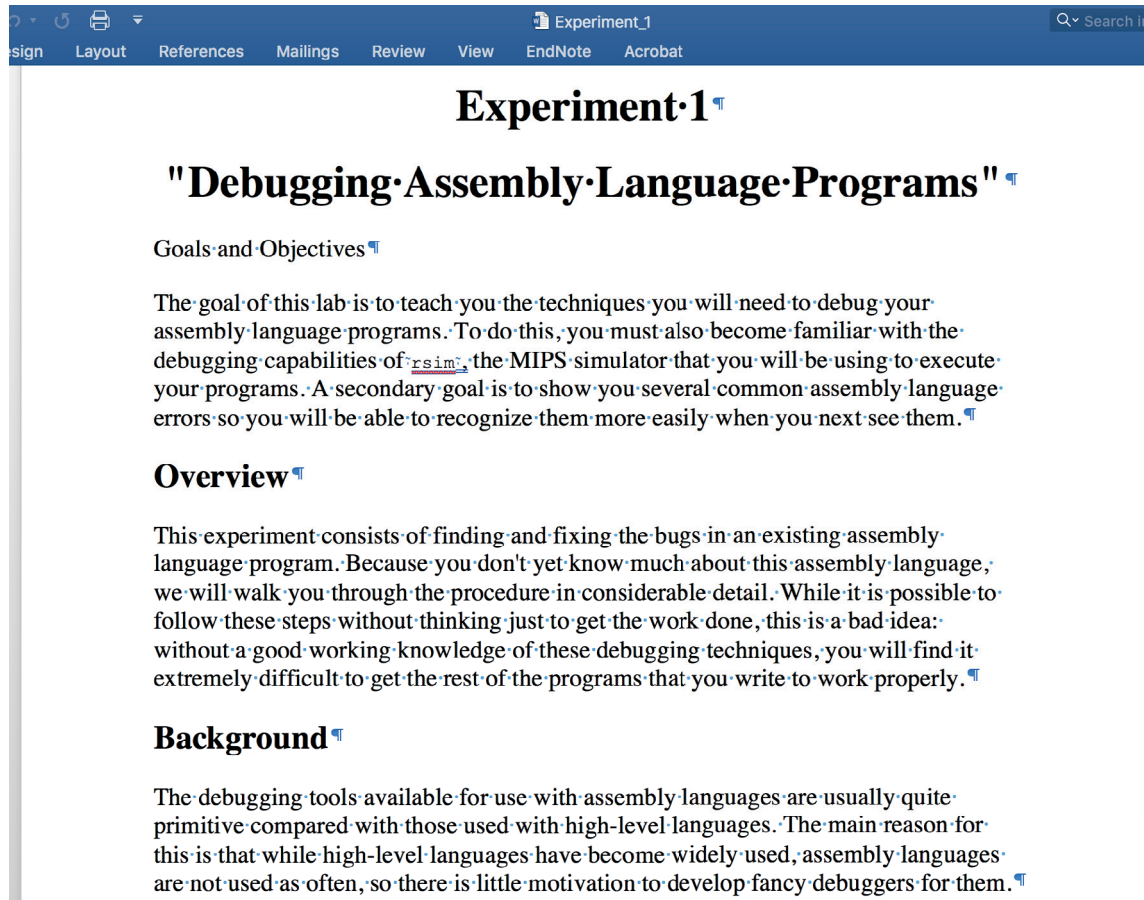


Figure 22. Course assignment demonstrating debugging as part of troubleshooting.

Another example from Bill's course and demonstrated in Figure 24 below, involved the class notes available to students. These notes demonstrated testing as a form of debugging and as a part of an overarching problem. In this case, testing was presented as an incremental action to be applied to the larger coding problem at hand. The document guided students through the process of testing and implementing a specific set of real-world (an aspect of rule-using problems as well) algorithms (algorithmic problems). While the troubleshooting activity was loosely identified as incremental testing, a formal strategy was not identified or applied, but rather an emphasis was placed on examining the system, running the tests and evaluating the results such that fault states could be confirmed.

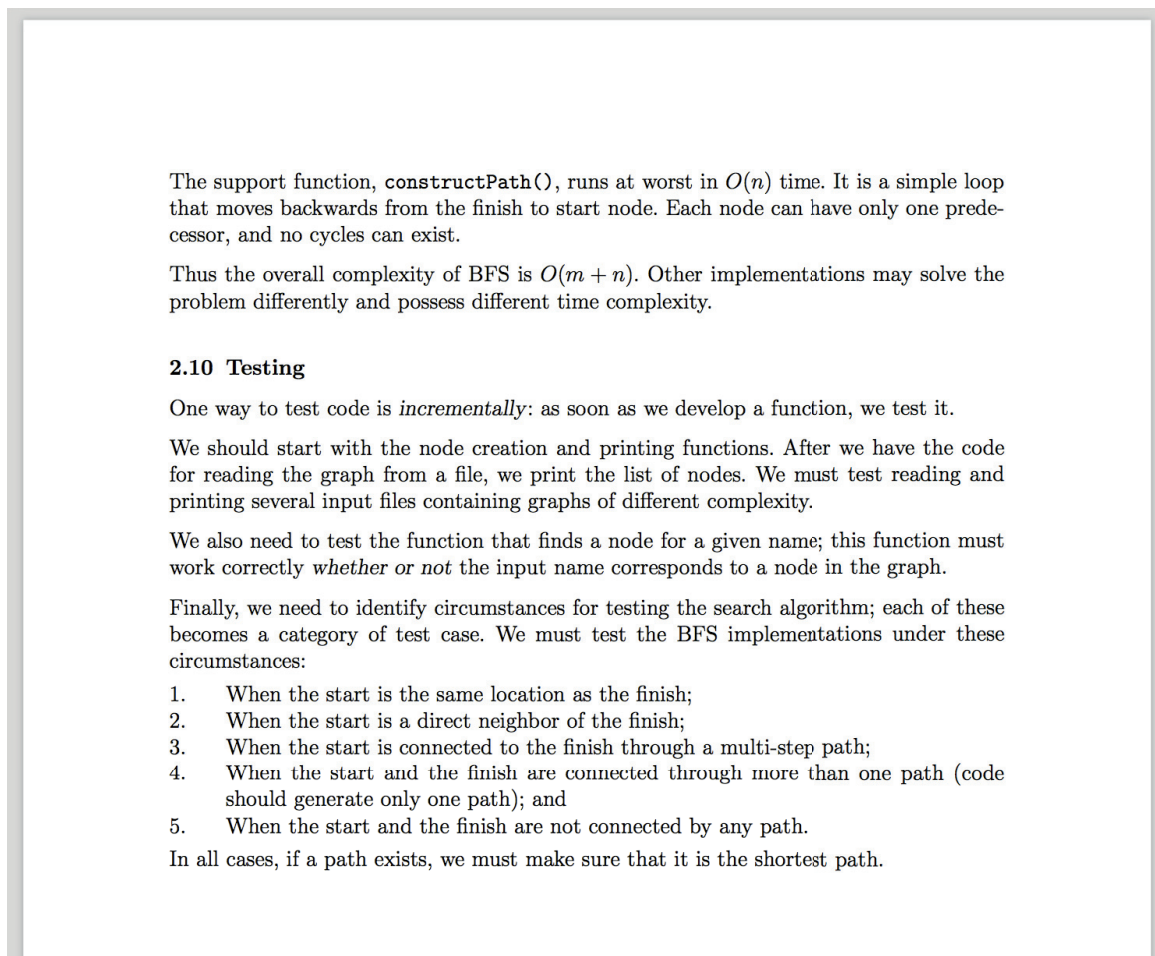


Figure 23. Course notes demonstrating testing as a part of debugging in troubleshooting.

Table 14 below summarizes the learning activity, success criteria and context attributes of the course artifact findings in regards to troubleshooting problems.

Table 14
Course Artifacts - Troubleshooting Problems

	Troubleshooting Problems
Learning activity	Examine system, run tests, evaluate results, hypothesize and confirm fault states using strategies (replace, serial elimination, space split)
Findings	Artifacts: 5/7 courses supported
Success criteria	Faults identification, efficiency of fault isolation
Findings	Artifacts: 5/7 courses supported
Context	Closed system, real world
Findings	Artifacts: 5/7 courses supported

Design problems in course artifacts. Design problems emerged in the course artifacts in all seven of the courses examined. However, this analysis revealed an incomplete understanding as the learning activity of “acting on goals to produce artifacts” was noted as an implicit rather

than explicit aspect of the course artifacts. Considering that a key goal of computing is producing artifacts through programs, databases, systems, etc., this presented itself as little surprise. The other learning activity aspects of design problems such as structuring and articulation, the success criteria of multiple undefined criteria with no right or wrong answers and the problem context that included complex and real-world scenarios with degrees of freedom, presented more insights into identifying design problems among the course artifacts. As such, the learning activity attribute of “problem structuring and articulation” was less noted and only presented itself in four of the seven sets of course artifacts. Noteworthy is that all four of these courses focused on programming. Of those four, two courses included explicit requirements for design while the other two were more intrinsic in nature, whereby the assignment could not be completed without consideration of problem structuring and articulation, yet no specific attention was drawn to this aspect as part of the assignment.

Bill’s course, which did explicitly require students to attend to the design problem learning activity of problem structuring and articulation, integrated a weekly ‘problem-solving session’ where students collaborated to answer questions focusing on design solutions for the lab. Figure 25 below, shows the document from the problem-solving session where a problem was posed requiring students to provide a design solution.

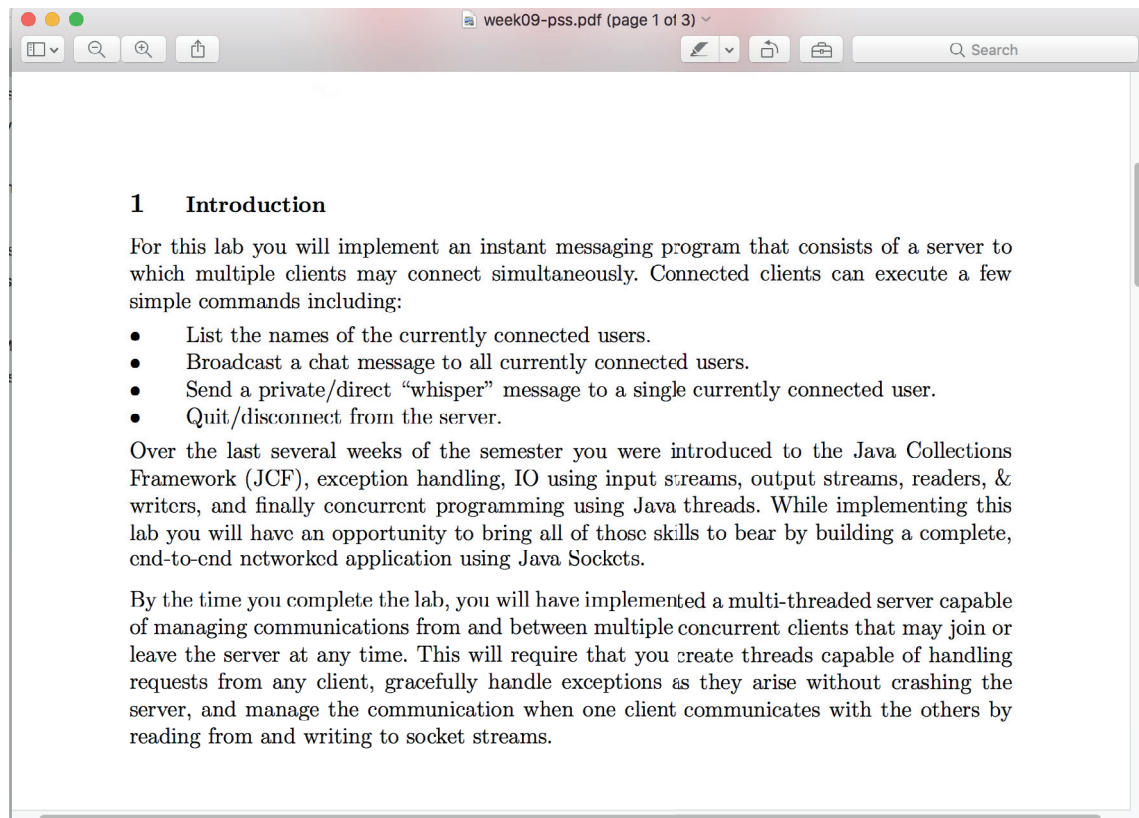


Figure 24. Exercise demonstrating design problem structuring and articulation.

Aligning with the learning activity of design problems, students were required to structure the problem to act on goals and determine a design solution. However, in contrast to design problems, students were presented with a context that included a significant amount of input and feedback in terms of the requirements. This idea of a detailed set of requirements runs contrary to the design problems specifying a vague goal statement with few constraints, and aligned more closely with the previously examined algorithmic and rule-using problems.

John’s was the single course primarily and explicitly focused on software design, and as such, the problems posed most closely aligned with Jonassen’s (2000) design problems. Because students were coding a software system, other problem type elements were also apparent such as algorithmic problems, rule-using problems, and troubleshooting problems, again indicating a departure from the pure design problems and an incomplete understanding. However, the problems in John’s course did present a focus on the design aspects where the learner was

required to act on goals to produce an artifact through problem structuring and articulation. The project also allowed for undetermined solution paths, noted in design problems, as the students were able to design a system meeting the requirements. Students were instructed to include design aspects for the course problem, including the following from a project requirements document (note that UML, or Unified Modeling Language, is specifically intended as a standardized method for conceptualizing and visualizing system design).

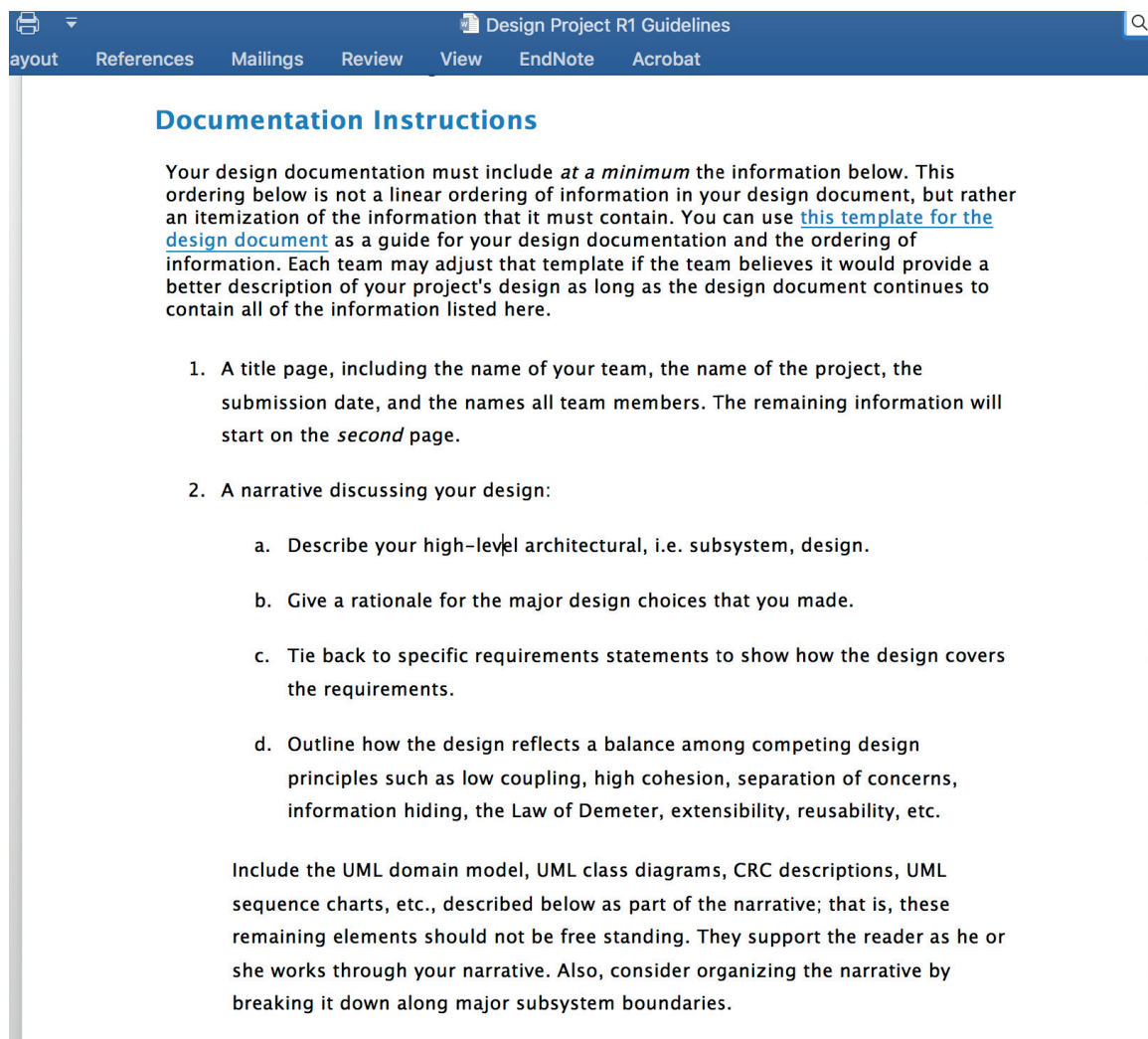


Figure 25. Course project demonstrating no specified criteria for success as part of a design problem.

As noted in 2b in Figure 26 above, the project required students to rationalize their design choices, indicating there was no specified criteria for success, no single right or wrong answer and that multiple solutions existed. This is further noted in the accompanying document in

Figure 27 below where the second paragraph clearly identified that there were no right or wrong design choices, only better or worse, explicitly aligning with the success criteria for design problems. However, the document also specified that a working program was expected, which lies in contrast to design problems, and more closely aligns with algorithmic and rule-using problems.

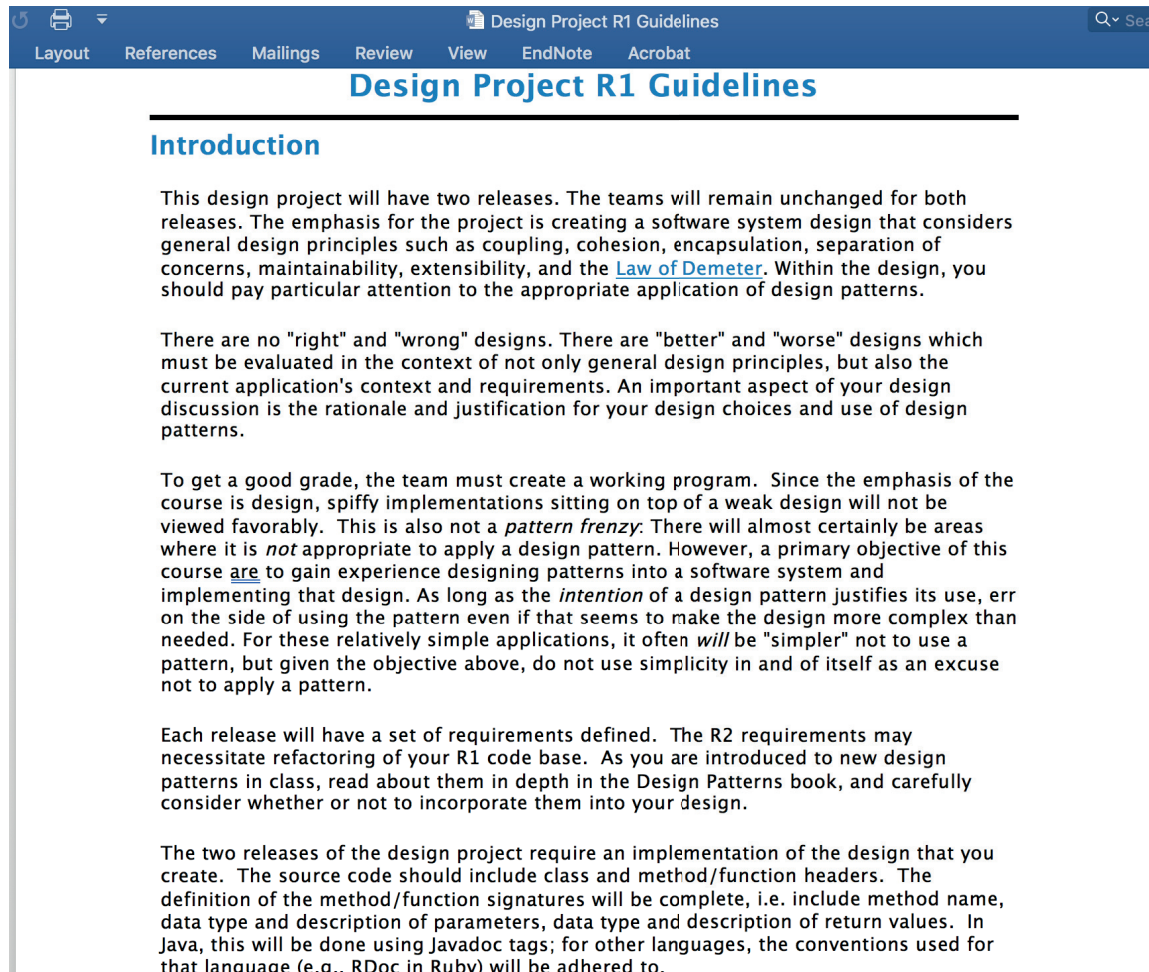


Figure 26. Project requirements demonstrating only better or worse design choices as in design problems.

In conclusion of the course artifact analysis around design problems, all four courses that included implicit or explicit design problems were woven around real-world and complex problems. Table 15 below summarizes the course artifact findings of design problems in regards to the learning activity, success criteria and context attributes.

Table 15
Course Artifacts - Design Problems

	Design problems
Learning activity	Acting on goals to produce artifacts, problem structuring and articulation
Findings	Artifacts: 7/7 courses required acting on goals to produce artifacts, 2/7 courses implicitly required problem structuring and articulation, 2/7 courses explicitly required problem structuring and articulation
Success criteria	Multiple, undefined criteria, no right or wrong, only better or worse
Findings	Artifacts: 2/7 courses explicitly supported
Context	Complex, real world, degrees of freedom, limited input and feedback
Findings	Artifacts: incomplete understanding; 7/7 courses supported real world, 4/7 presented degrees of freedom (2 implicit, 2 explicit), none presented limited input

Problem Types in Instructional Classroom and Laboratory Observations

Classroom and laboratory observations were consistent with examination of course artifacts and also revealed an overarching emphasis on four problem types: algorithmic problems, rule-using problems, troubleshooting problems and design problems.

Algorithmic problems in instructional classroom and laboratory observations. All seven faculty were observed presenting algorithmic problem learning activities through procedural sequences of manipulations that could be applied to similar sets of variables to produce a correct answer as well as contextually abstract or formulaic problems. Success criteria requiring an answer to match in value or form was observed with six of the seven faculty. Observations of Sue's class revealed her walking students through an example of a problem that was algorithmic in nature and she addressed the class by saying, "what we are going to do this week is focus in taking a relational diagram and transpose it ... you already know how to do these." Sue also directed the students to the reference sheet previously noted as Figure 12. Sue explained that they were going to follow the process and then she continued on to guide them through that process. She asked, "What would the primary key need to be for employee id?" followed by, "What about adding department number to employee?" She continued onto ask the class, "if we needed to determine functional dependencies what would we have? Does employee

id determine others? Is it a valid key? Do we have any other functional dependencies?” Students answered these questions and the instructor explained the rationale for each, indicating a correct answer or expected outcome as with algorithmic problems. Sue also demonstrated the idea of presenting the problem in an abstract context. She offered, “Now let’s go through some examples using A, B and C because I’m not feeling very creative today and because it forces you to focus on the process ...”. Table 16 below summarizes the findings in regards to the learning activity, success criteria and context attributes for algorithmic problems as supported by the instructional classroom and laboratory observation dataset.

Table 16
Instructional Classroom and Laboratory Observations - Algorithmic Problems

	Algorithmic problems
Learning activity	Procedural sequence of manipulations; algorithmic process applied to similar sets of variables; calculating or producing correct answer
Findings	Observations: 7/7 courses supported
Success criteria	Answer or product matches in values and form
Findings	Observations: 6/7 courses supported
Context	Abstract, formulaic
Findings	Observations: 7/7 courses supported

Rule-using problems in instructional classroom and laboratory observations. In

terms of rule-using problems, all seven faculty demonstrated learning activities that focused on procedural processes constrained by rules and applied those rules to produce answers or products. Six faculty demonstrated success criteria through a number of relevant answers or products (sometimes a single answer, as with algorithmic problems). All seven faculty presented problems that were contextualized in real-world scenarios. Bill presented his students with a programming assignment that was to implement a real-world battleship game. In regards to the rules and procedural processes that were part of the learning activity, Bill explained to the students that in programming the game boards they “need to know which one belongs to each player because they display a little differently.” Bill demonstrated the possibility of multiple

solution paths when he examined one student's programming solution and explained "I wouldn't have done it that way...[details on other options for implementation]... its awkward, but its simpler, I think ..." Table 17 below summarizes the instructional classroom and laboratory dataset for rule-using problems in regards to the three attributes, learning activity, success criteria and context.

Table 17
Instructional Classroom and Laboratory Observations - Rule-using Problems

	Rule-using problems
Learning activity	Procedural process constrained by rules; select and apply rules to produce system-constrained answers or products
Findings	Observations: 7/7 courses supported
Success criteria	Productivity (number of relevant or useful answers or products), multiple solution paths, procedures or methods
Findings	Observations: 6/7 courses supported
Context	Purposeful academic, real-world, constrained
Findings	Observations: 7/7 courses supported

Troubleshooting problems in instructional classroom and laboratory observations.

Troubleshooting problems were observed in four of seven courses. Of the four where troubleshooting was observed, all required students to engage in some hands-on implementation or design. The learning activities in these classrooms and labs engaging in troubleshooting problems integrated the examination of systems, running tests, evaluating results, hypothesizing about the fault states and then making corrections. Troubleshooting success was evident in the same four classrooms/labs as where the faults were identified, and all four classes' troubleshooting problems were also embedded in a real-world context. In observing Tim's lab section, troubleshooting was evident in the form of debugging by having students examine the system and evaluate the results when he made an announcement to the entire class that, "when running powershell, you don't have to generate an error, just figure out how you can get some type of information, a warning or whatever... Just try to figure out how to get some information from the event viewer." At another point in the lab, Tim again identified faults by examining the

system when he helped a student with, “were just going to look at all the system errors...” This was similarly noted in Dave’s class when he directed students to “test, test, test!” before assisting a specific student who was reporting an error that he needed help deciphering. Dave walked over to the student and asked the student to show him the problem. Dave continued to look over the student’s shoulder and explain that it was not actually an error but rather a warning. By working with the student to evaluate the results, Dave exemplified walking the student through a troubleshooting problem. Table 18 below summarizes the three attributes of learning activity, success criteria and context in regards to troubleshooting problems as found in the instructional classroom and laboratory observation dataset.

Table 18
Instructional Classroom and Laboratory Observations - Troubleshooting Problems

	Troubleshooting problems
Learning activity	Examine system, run tests, evaluate results, hypothesize and confirm fault states using strategies (replace, serial elimination, space split)
Findings	Observations: 4/7 courses supported
Success criteria	Faults identification, efficiency of fault isolation
Findings	Observations: 4/7 courses supported
Context	Closed system, real world
Findings	Observations: 4/7 courses supported

Design problems in instructional classroom and laboratory observations. The instructional classroom/laboratory findings echo the artifact examination findings in that they were overall incomplete in regards to design problems. Consistent with the course artifacts, design problems were apparent in all seven of the instructional classroom and laboratory observations specifically in regards to the learning activity of acting on goals to produce artifacts. However, the learning activity attribute provided limited understanding, and in further examining the three problem-type attributes, the problem structuring aspect of the learning activity, undefined success criteria with only better or worse designs and the contextualization of complex, real-world problems with limited input and feedback provided deeper insights into the

problems presented. As such, design problems emerged as apparent in three of the seven courses observed. All three of these courses focused on programming assignments as a primary deliverable. Two of the instructional classroom/laboratory observations revealed explicit use of design problems, while the third was implicit in nature where the project could not be completed without addressing the design aspects. While the success criteria attribute was not directly observed, some classroom discussion around ‘better or worse’ choices was evident. All the problems were presented as real-world, yet none presented limited input, and instead provided significant input in the form of requirements along with advantages and disadvantages in regards to design choices.

John’s class explicitly demonstrated characteristics of design problems. For example, when students broke out into small groups to work on semester projects, John engaged one group with the following dialogue, “So you’re at the conceptualizing ... design or implementation level? What patterns are you using, how is that going? What role does privacy play? ... There are several implementation possibilities, all are appropriate at some level.” This clearly exemplified that design characteristics were part of the problem as students clearly had options for various design patterns to be used in the project as well as several viable implementation possibilities. In another example in John’s class, design characteristics were again observed when John presented students with, “So this has to be an implementation inheritance... that is a design issue to consider ... what are the advantages of object adaptation? What are the advantages of class adaptation?” The design problem characteristic of better or worse solutions as reflected in the idea of advantages and disadvantages is clearly being presented to students for consideration. John continued on to describe a specific design pattern along with the definition of that design pattern, thereby presenting significant input and feedback to the context of the problem.

Bill also presented students with significant input regarding the context of the problem. During one lab session, he asked students, “how do you do a design so that you cannot impact the rest of the program very much?” While this approach did encourage student to articulate the program structuring as part of the learning activity, Bill continued on to guide students to a solution using a specific programming construct by providing significant input regarding the approach to use.

A similar observation was made in Dave’s class when he offered to students, that they might accomplish one aspect of the programming assignment by using either a vector or an array, but then added “I suggest a vector because vectors are thread safe.” Finally, a more detailed exchange between the instructor and the students regarding design occurred in Bill’s class:

Bill: Normally I don’t have people label things in a vbox or hbox, but its ok...

Student: can an alternative design be that we have two panes and on the bottom a text area?

Bill: Do you think that game would sell? Believe it or not, you’re not the first group in here to suggest that... what I prefer that you do is just say button, and indicate there are a bunch of them.

While the exchange indicated multiple acceptable means of accomplishing the task at hand, the instructor also specified a ‘better or worse’ solution with his explicit preference. Table 19 below summarizes the design problem findings in regards to the learning activity, success criteria and context attributes as part of the instructional classroom and laboratory observations.

Table 19
Instructional Classroom and Laboratory Observations - Design Problems

	Design problems
Learning activity	Acting on goals to produce artifacts, problem structuring and articulation

Findings	Observations: 7/7 courses required acting on goals to produce artifacts, 1/7 courses implicitly required problem structuring and articulation, 2/7 courses explicitly required problem structuring and articulation
Success criteria	Multiple, undefined criteria, no right or wrong, only better or worse
Findings	Observations: incomplete understanding regarding criteria, 3/7 courses supported no right or wrong, only better or worse
Context	Complex, real world, degrees of freedom, limited input and feedback
Findings	Observations: incomplete understanding, all were real world, 3/7 courses presented degrees of freedom (1 implicit, 2 explicit), 0/7 courses presented limited input

Problem Types in Faculty Interviews

Consistent with examination of course artifacts and instructional classroom/laboratory observations, faculty interviews also supported the dominance of the four problem types: algorithmic problems, rule-using problems, troubleshooting problems and design problems. However, faculty interview explanations of the problem types were more general and thus evidence of the specific attributes were not always identified. In contrast, the specificity of the attributes such as success and design criteria were more explicit during classroom enactments and in use of the artifacts. Still yet, the interview responses revealed that understandings of these problem types were incomplete.

Algorithmic problems in faculty interviews. Faculty emphasized algorithmic problems in terms of the overarching learning activity that presented a procedural sequence of manipulations, an algorithmic process applied to similar sets of variables, and calculating or producing a correct answer. In considering problems in the computing domain, Bill specifically described problem solving as working to “design algorithms and classes to help realize ... solutions” (Bill, personal interview, April 13, 2018). Chris also similarly described problem solving in the context of an algorithmic problem type when he explained it as “coming up with an algorithm to solve a problem,” and later on stated even more plainly that “the whole algorithm

solving is always really nothing more than problem solving” (Chris, personal interview, April 12, 2017). Bill further characterized algorithmic problems when he said,

in general we jump right into their list of specific things they have to solve, which is intended to guide them toward the answer. The next upcoming lab is scheduling jobs with several workers who can do the jobs. The first question is how would you determine that it’s even possible? Or is it job A needs job B done, and job B needs job A done, right? Stuff like that. General algorithms are given, then we ask them to apply it to specific problems. (Bill, personal interview, April 9, 2018)

In this sense, Bill described a learning activity where an algorithmic process was applied to produce a correct answer. Using the example of job A and job B also exemplified the abstract lens through which the instructor was considering the problem, as is contextually specified with algorithmic problems.

Three faculty described the attribute for success criteria in looking for correctness in the form of a product that matched in value and form to an expected result. Sue indicated that she expected correct outcomes in terms of homework using a particular procedure, “So each week, there tends to be a homework assignment, and the homework assignment is graded based on correctness” (Sue, personal interview, April 12, 2017). Chris also elucidated on the idea of producing a correct answer when he suggested that there is always more than one way of solving a problem, “... but, you know, if they don’t use those steps, then they can’t, then they’re obviously doing something that is not natural in the language. And it’s not that you can’t do it, it’s just definitely not optimal” (Chris, personal interview, April 12, 2017).

In regards to the abstract or formulaic context of algorithmic problems, only one faculty member explicitly described using abstract or formulaic context, although as previously noted,

Bill implicitly addressed it with his example. Sue specifically addressed her preference for abstraction in presenting the problem to students as a procedural sequence of manipulations. She explained,

oftentimes I try to use real world examples, but then I also use abstract examples. So I don't know if, in one of the classes you attended, I had letters and numbers examples? ...So the letters and numbers, that forces them to focus on the process. So, I developed [a] transposing reference sheet, um, that, again, helps guide them through the process. And then, I give them abstract examples, so that they can't make connections between the meanings of the words, and like, oh, well this seems like this should go together and this seems like that should go together, um, with letters and numbers they're forced to focus on the process ... and once they have the process engrained, then they can use that to solve any or transpose any ER diagram that they're, that they're given. But I know that the letters and numbers are abstract, so I mix that with real-world scenarios that they can understand. (Sue, personal interview, April 12, 2017)

Of note is that Sue also framed her statement with the notion that she also uses real-world examples, a characteristic of rule using problems. Table 20 below summarizes the interview findings around algorithmic problems.

Table 20
Faculty Interviews - Algorithmic Problems

	Algorithmic problems
Learning activity	Procedural sequence of manipulations; algorithmic process applied to similar sets of variables; calculating or producing correct answer
Findings	Interviews: 7/7 faculty emphasized
Success criteria	Answer or product matches in values and form
Findings	Interviews: incomplete understanding with only 3/7 faculty clearly emphasizing an answer matching in value
Context	Abstract, formulaic
Findings	Interviews: incomplete understanding with only 1/7 faculty clearly emphasizing abstract context

Rule-using problems in faculty interviews. All seven of the faculty described rule-using problem characteristics in the assignments they integrated into their courses. Specifically, faculty emphasized the overarching idea of the problem having a learning activity with a clear goal, resulting in a constrained purpose without restricting the solution to a single specific procedure or method, but rather allowing for multiple solution paths. Sue described a learning activity with a procedure constrained by rules as,

I give them a reference sheet to go off of that ...that goes through the processing steps, so it's, check for first normal form, check for second normal form. If it's not a second normal form, follow the fix it steps. So the fix it steps is where if they follow that... they'll get the right answer. (Sue, personal interview, May 8, 2017)

Notably, Sue also identified that that the information presented as part of a problem is expected to be helpful to students. She explained, "So where I might say that ... I lead them to derive the answers based off of the ... the details provided" (Sue, personal interview, May 8, 2017). These details presented a learning activity whereby a procedural process was applied to produce a set of constrained answers or products.

Six of seven faculty overwhelmingly articulated that multiple solution paths exist in terms of determining success for a problem. In rejecting the idea of a single correct approach or solution to a problem, faculty instead focused on ideal approaches or solutions and better or worse solutions, also notable in design problems. Tim emphasized that students may use differing paths to solve a problem, an aspect of rule using problems when he explained that students were required to put together a book that encapsulated their work for the semester. Tim described that he looked at the end result with a perspective of, "Are these steps right? Are these

procedures right? ... do they work for you? Are they right for you? ... I don't care about if they're gonna work for me. Do they work for you? You know?" (Tim, personal interview, April 11, 2017). Tim also described the labs from his course as using procedures on how to set up specific systems, yet he emphasized that students may be using a different set of steps to get to the solution. More important than using a specific set of steps, Tim explained that he wanted students to have a deeper grasp of those steps,

Understanding how things work within the, the protocol stack, the OSI stack. So that's really what I'm going for, not just that step by step, everybody does the same exact steps to get to the root cause. You know, I wanna, I wanna make sure that they're using the tools that they have and that they know how to use them.

(Tim, personal interview, April 11, 2017)

John specifically identified that there are "different design choices that are gonna lead down different paths and there are tradeoffs, and so there is no right and there's no wrong, there's just works better or works not as good." Joan reiterated this idea, although in more general computing education terms rather than specifically regarding to the course as part of this study. She said, "there isn't always one solution, one might be better than the other, there isn't just one right answer all the time ..." (Joan, personal interview, April 12, 2017).

Tim continued on to describe his technique in conveying to students that multiple approaches to solve a problem are appropriate. According to Tim, he tells students, "don't worry about how I think you should do [it]. You, you worry about how you think this is the best way to do it, and what's the most efficient way to do it." While also conveying the idea of multiple solution paths, Chris framed the success criteria by suggesting that students have choices in what

algorithms, programming languages, tools and technologies they use to get to a solution. When asked if there is typically one solution to a problem, Chris responded:

No. Obviously there's an infinite number of solutions to any problems. And so, you know, the tools don't necessarily dictate. I mean, there's nothing stopping you in C [programming language] from doing object-oriented programming.

There's nothing stopping in you in Java from running a non-procedural, writing a non-object-oriented program...(Chris, personal interview, April 12, 2017)

However, while both of the ideas that Chris outlined as solutions are possible, he was emphasizing that by executing such solutions, students would be forcing the language to do something that it was unintended for, or unnatural, thereby producing a solution that was not ideal, regardless of whether or not the correct output was produced. He continued on with, "...there are perfectly good viable solutions [by students] who just interpret the problem differently" (Chris, personal interview, April 12, 2017). Chris also explained that in addition to the correct output, faculty are working to emphasize good programming standards to students. He offered, "... we're trying to get them to sort of program to a standard. This is what the user said they want, do this. Don't do something else" (Chris, personal interview, April 12, 2017). Bill echoed this idea, but also explained the struggle with balancing the possibility of multiple problem solutions with latitude for student creativity in making design choices and computing best practices, "They [students] just got started, and eventually they'll get better. By the way another balancing act is letting them be creative and go their own way versus not letting them do something we know is bad practice" (Bill, personal interview, April 9, 2018). Similarly, Sue shared her approach to presenting ideal solutions in the context of pros and cons to each:

I pose realistic situations to the, to the students, um, and that I allow them an opportunity to think about potential solutions but also, because they're just learning in that domain, that I also cover potential solutions to the problem and discuss the pros and cons to each option and then, you know, if... given my experience in the domain, if I feel that there's one option that's clearly better than the other options, to relay that to them. (Sue, personal interview, April 12, 2017)

Joan further echoed this with, "So they have to understand that too, that there's never only one answer to everything. But sometimes there's better answers" (Joan, personal interview, May 1, 2017).

All seven of the faculty described using problems in a context that was real-world situated or emphasized their academic purpose. As previously noted in the algorithmic problems section, Sue specifically addressed using both real-world and abstract problems, "oftentimes I try to use real-world examples, but then I also use abstract examples" (Sue, personal interview, April 12, 2017). All of the remaining faculty expressly focused on using real-world problems or purposeful academic problems. Bill explained using purposeful academic problems such as games as well as real-world problems, yet he was conflicted that these problems were not in the wheelhouse of current students,

I don't think we have enough that they can connect with as having seen them in their own computer use, right? We're doing games. I guess yeah they can relate to games, but we do it too much. We do puzzles sometimes, which really don't have a point other than it's fun. Kind of like games. ... I did one this term with trying to do some of the work that ... would have to deal with cars coming on and off and charging and stuff. It's a little boring, but at least it's more real life. We need

more of those, but the problem is, I think, they're harder because either we need a lot of data to start with, and I think we can solve that. I think if I knew where to look I could find some. Or it's more real-time embedded stuff, which then requires an investment of time to put up some kind of simulation they would connect to and use. (Bill, personal interview, April 9, 2018)

Dave described setting up problems that served as purposefully academic in nature, while also integrating abstract aspects of algorithmic problems,

The basic problem there that we start with is how do you ... What are the elements of the language? If statements and while loops and methods and et cetera, et cetera. What we do is we try to set them problems that require that they use those features of the language to solve some new problem they hadn't quite thought of before. When we get into [current course title], the question of teaching currently, it's a different story altogether. We assume they have the elements of the language, with a few exceptions, and we're trying to teach them major new concepts that are ... I think there are four major concepts we try to teach them. In [course name], there's binary input and output, there's GUI programming, network programming, and threads. Those are the four major concepts. These are not exactly elements of the language. They come to you through the API, so they're more advanced than elements of the language, but they're very important concepts, and they are much harder to grasp, because they're much more advanced. It's not where do I put the parenthesis in a for loop, which is something ultimately you can just memorize. These are how do you take these particular tools, and there aren't that many of them, but how do you take these tools and use

them to solve this particular problem... to the point where they can solve that particular problem. It's not a rote memorization. Can I conceptualize this abstract problem and see the pattern of how I go about solving it and repeat it?" (Dave, personal interview, April 9, 2018)

Sue emphasized the importance of real-world problems when she offered, "I wanna make sure that, you know, I pose realistic situations to the, to the students" (Sue, personal interview, April 12, 2017). Bill explained the importance of the real-world application of the problems being posed to students with,

I feel I should point out that this problem-solving business is supposed to mean that they're solving practical problems in the sense they are not doing this simply because it's an exercise to learn something, but it should always be some kind of application. (Bill, personal interview, April 9, 2018)

Dave demonstrated a stronger emphasis on the academically purposeful problems when he explained how he develops problems for his course,

I know that this is the topic, I need a project. How do I come up with one? I think about things I've done in the past. Sometimes it's things someone else did in the past, earlier version of the course. They have similar labs, I'll take those and bend them to my needs. So, to come up with the idea of what it's gonna be is, it's just out of whole cloth largely. (Dave, personal interview, April 13, 2018)

Table 21 below summarizes the learning activity, success criteria and context attributes in regards to the interview findings for rule-using problems.

Table 21
Faculty Interviews - Rule-using Problems

	Rule-using problems
Learning activity	Procedural process constrained by rules; select and apply rules to produce system-constrained answers or products

Findings	Interviews: 7/7 faculty emphasized
Success criteria	Productivity (number of relevant or useful answers or products), multiple solution paths, procedures or methods
Findings	Interviews: 6/7 faculty emphasized
Context	Purposeful academic, real-world, constrained
Findings	Interviews: 7/7 faculty emphasized

Troubleshooting problems in faculty interviews. Five of the seven faculty described troubleshooting as an aspect of the problems being posed to students. Troubleshooting learning activities focus on examining systems, running tests, evaluating results to hypothesize solutions and using strategies. In the interviews, faculty specifically addressed the learning activities noted above in the form of debugging computing programs or systems, yet they omitted any discussion of specific success criteria in terms of identifying or isolating faults or examining troubleshooting in a closed or real-world system. While one might extrapolate these ideas around success criteria and real-world context from the other problem types presented, lack of specific data from the interviews yielded an incomplete understanding in regards to these two attributes for troubleshooting problems. Returning to the learning activity for troubleshooting problems, Chris explained,

I try not to let, to let them make me do the debugging for them... I am trying harder and harder to give them self-sufficiency on how to debug... I've basically implemented a policy where if you come in my office asking me about problems about your broken program, you have to show me the debugger running of it and tell me where it's blowing up and what's going wrong. (Chris, personal interview, May 4, 2017)

Dave keyed in on troubleshooting as an aspect of the problems by describing that,

...sometimes you just get stuck, and you can't see the forest for the trees. You're looking at this particular problem. It certain[ly] has happened to me many times.

You're banging your head against the wall. You can't. Why isn't this working?

You'll get a colleague or somebody to come in, take a look at it, fresh eyes. They take a look, and they say, "Oh, it's right," and you fix it, and by god, it's gone.

(Dave, personal interview, April 9, 2018)

Bill described that students had the option to troubleshoot course problems via an online forum where there was an opportunity to,

...ask questions. We let them post a few lines of their own code if they need to and say why isn't this working? Or more generally they say here's the approach I've taken, what do you think of it? And we answer them. (Bill, personal interview, April 9, 2018)

Tim described the nature of troubleshooting problems in his course as,

things are gonna break, and you gotta figure out how to fix them, but part of learning about it is breaking things. It's not just getting to the right answer. It's about understanding why things break, how they break, and how you go about troubleshooting to figure out how to fix it. (Tim, personal interview, April 11, 2017)

Tim continued on to describe a problem that he posed to students on an exam,

... on their midterm, I have a troubleshooting exercise in there, and I tell them, there's no right or wrong answer, I just want to see that you're using the tools that you, that you've learned in class and the methods that we learned, and applying them so that it leads you to the root cause of the problem. (Tim, personal interview, April 11, 2017)

Table 22 below summarizes the interview dataset findings around troubleshooting problems in regards to the learning activity, success criteria and context attributes.

Table 22
Faculty Interviews - Troubleshooting Problems

	Troubleshooting problems
Learning activity	Examine system, run tests, evaluate results, hypothesize and confirm fault states using strategies (replace, serial elimination, space split)
Findings	Interviews: 5/7 faculty emphasized
Success criteria	Faults identification, efficiency of fault isolation
Findings	Interviews: incomplete understanding
Context	Closed system, real world
Findings	Interviews: incomplete understanding

Design problems in faculty interviews. Design problems presented the most complex and incomplete understanding of all problem types as only some aspects of some attributes were identifiable. None of the faculty specifically described problems that required students to act on goals to produce artifacts or problem structuring as part of the problem as regards the learning activity attribute. However, similar in nature to rule-using problems, four of the faculty specifically described better or worse solutions to problems as it relates to success criteria.

Bill expressed what might be construed as discontent with the opposing attributes of highly prescriptive requirements as outlined in algorithmic and rule-using problems with the multiple solution paths enabled in rule using and better or worse solutions in design problems. He offered that the requirements and constraints posed in his problems are sometimes in conflict with allowing for varying solutions to those problems. This stemmed from his explanation that while students are sometimes allowed to work through design decisions initially, they are later required to use faculty specified designs in moving forward with a project, "... it's this tension between let their creativity come out, versus, we need to make sure they practice something we taught them, and also versus, the students are not as comfortable with letting their creativity come out" (Bill, personal interview, April 13, 2018). He also described,

Supposedly by the end of problem solving, they have some kind of idea of the design and they can start doing the implementation. In some cases like what I was describing, you would imagine that it's pretty nailed down because we sort of directed them that way. In some ways it's more open ended that we feel there are several ways to do it. We'll let the group decide. The worst thing that we do certainly more than a third of the time, that bothers me, is that in problem solving we tell them it's up to you how you would solve this, but then at the end of problem solving we say because we wanna make sure you learn that material that we're talking about this week, you will do it this way. I warn them, I say unfortunately some of these designs are throw-away, but I don't like that. (Bill, personal interview, April 9, 2018)

John reiterated the idea of throwing away design, but in a different sense. He focused more on the value of the design over the code,

Well, I, I try and teach them that, they're not, even if they throw away the code, they're not throwing away the, throwing away the knowledge they've gained. ...So their next implementation is gonna be faster to, to write. It's gonna work better. Um, it's gonna be easier to maintain in their next releases...So that, throwing away code is cheap compared to throwing away designs, throwing away requirements. (John, personal interview, May 8, 2017)

Bill also explained that the design aspect of the problem was valued in terms of the overall assignment, although still with a key focus on the instructor supplied design. He offered,

I will say in general, I think this is mostly me, we've been assigning more of the points on assignment to design. Either did they follow ours? Or is the one they

came up with reasonable? If not, why not? (Bill, personal interview, April 9, 2018)

In fact, Bill further emphasized that he saw problem solving in computing as “being given some requirements and trying to choose the appropriate tools for a solution. And in these course, design algorithms and classes to help realize those solutions” (Bill, personal interview, April 13, 2018). Interestingly, this statement introduces the idea that many of the problems posed by faculty integrated the aspects of several types of problems all at the same time, a topic that will be further explored in Chapter 5.

In regards to the context attribute in design problems, only one faculty member came close to specifically addressing degrees of freedom and limited input. John explained that students are “expected to think about open-ended problems” because the problems are “closed-ended from a customer requirement point of view,” but “they’re open-ended from a solution design point of view” (John, personal interview, April 25, 2017).

Table 23 below summarizes the findings around design problems from the faculty interview dataset, and as regards the learning activity, success criteria and context attributes.

Table 23
Faculty Interviews - Design Problems

	Design problems
Learning activity	Acting on goals to produce artifacts, problem structuring and articulation
Findings	Interviews: incomplete understanding
Success criteria	Multiple, undefined criteria, no right or wrong, only better or worse
Findings	Interviews: 4/7 faculty described better or worse solutions
Context	Complex, real world, degrees of freedom, limited input and feedback
Findings	Interviews: incomplete understanding

In summary, problems posed by computing faculty commonly ran across problem type boundaries as defined by Jonassen’s (2000) problem-solving typology, exemplifying characteristics of several problem types at once, yet primarily illustrating well-structured characteristics. While faculty presented all four problem types in the course artifacts and

instructed characteristics of them in their classroom enactments, during interviews, faculty clearly emphasized the value of real-world, domain contextualized problems over abstract and formulaic problems. During interviews, faculty also recognized that computing problems inherently support multiple solution paths that allow for student creativity in design choices as with rule-using problems, but better or worse solutions exist, as is notable ill-structured design problems. Table 24 below summarizes the four predominant problem types, problem-type attributes and findings across data three sources from the seven courses.

Table 24
Problem Type Findings Summary

	Problem Types 			
	Algorithmic problems	Rule-using problems	Troubleshooting problems	Design problems
Learning activity	Procedural sequence of manipulations; algorithmic process applied to similar sets of variables; calculating or producing correct answer	Procedural process constrained by rules; select and apply rules to produce system-constrained answers or products	Examine system, run tests, evaluate results, hypothesize and confirm fault states using strategies (replace, serial elimination, space split)	Acting on goals to produce artifacts, problem structuring and articulation
Findings	Artifacts: 7/7 courses supported Observations: 7/7 courses supported Interviews: 7/7 faculty supported	Artifacts: 7/7 courses supported Observations: 7/7 courses supported Interviews: 7/7 faculty supported	Artifacts: 5/7 courses supported Observations: 4/7 courses supported Interviews: 5/7 faculty supported	Artifacts: 7/7 courses required acting on goals to produce artifacts, 2/7 courses implicitly required problem structuring and articulation, 2/7 courses explicitly required problem structuring and articulation Observations: 7/7 courses required acting on goals to produce artifacts, 1/7 implicitly required problem structuring and articulation, 2/7 courses explicitly required problem structuring and articulation Interviews: incomplete understanding
Success criteria	Answer or product matches in values and form	Productivity (number of relevant or useful answers or products), multiple solution paths, procedures or methods	Faults identification, efficiency of fault isolation	Multiple, undefined criteria, no right or wrong, only better or worse

Findings	<p>Artifacts: 7/7 courses supported</p> <p>Observations: 6/7 courses supported</p> <p>Interviews: 3/7 faculty emphasized an answer matching in value</p>	<p>Artifacts: 7/7 courses supported</p> <p>Observations: 6/7 courses supported</p> <p>Interviews: 6/7 faculty supported</p>	<p>Artifacts: 5/7 courses supported</p> <p>Observations: 4/7 courses supported</p> <p>Interviews: incomplete understanding</p>	<p>Artifacts: 2/7 courses explicitly supported</p> <p>Observations: incomplete understanding regarding criteria, 3/7 supported no right or wrong, only better or worse</p> <p>Interviews: 4/7 faculty described better or worse solutions</p>
Context	Abstract, formulaic	Purposeful academic, real-world, constrained	Closed system, real world	Complex, real world, degrees of freedom, limited input and feedback
Findings	<p>Artifacts: 7/7 courses supported</p> <p>Observations: 7/7 courses supported</p> <p>Interviews: incomplete understanding with 1/7 faculty emphasizing abstract context</p>	<p>Artifacts: 7/7 courses supported</p> <p>Observations: 7/7 courses supported</p> <p>Interviews: 7/7 faculty supported</p>	<p>Artifacts: 5/7 courses supported</p> <p>Observations: 4/7 courses supported</p> <p>Interviews: incomplete understanding</p>	<p>Artifacts: incomplete understanding, 7/7 courses presented real world examples, 4/7 courses presented degrees of freedom (2 implicit, 2 explicit), none presented limited input</p> <p>Observations: incomplete understanding, 7/7 courses presented real world, 3/7 courses presented degrees of freedom (1 implicit, 2 explicit), none presented limited input</p> <p>Interviews: incomplete understanding</p>

Instructional Approaches to Problem Solving Enacted by Undergraduate Computing Faculty

This section is organized to address major findings for question two. More specifically, this section presents the main themes that address computing faculty instructional approaches to problem solving in their undergraduate computing courses. Across all three data sources, the findings indicated a focus on well-structured problem-solving steps, and more specifically, problem decomposition served as the predominant approach to problem solving whereby a larger problem was broken down into smaller problems or subsets of problems. Findings also revealed that faculty instructional approaches implicitly modeled problem decomposition rather than explicitly defining or identifying the approach. Below I address these findings based on instructional classroom and laboratory observations, faculty interviews, and course artifacts.

Decomposition as Problem-Solving in Instructional Classroom and Laboratory

Observations

A total of 22 observation hours were completed across seven courses. Courses varied in delivery format: two courses included a separate lecture and lab period (with one of those also including a recitation session), one course used a studio-type approach where both traditional lecture and lab exercises were integrated into the course time that was scheduled in a laboratory, one course split the scheduled course time between traditional lecture and group project work, and three courses followed a traditional lecture format.

Separate lecture and lab period (one course included a separate recitation period)	Tim's course Bill's course
Studio-type model with traditional lecture and lab exercises integrated into scheduled course time	Dave's course
Lecture and project group work integrated into scheduled course time	John's course
Traditional lecture format	Deb's course Sue's course Chris' course

Three of seven courses nominally exemplified the decomposition of problems to subproblems through the instructional classroom and laboratory observations. As previously noted in Figure 29, the use of the laboratory sign-off document was also observed in Tim's laboratory session as a means to decomposing problems into subproblems. This approach allowed students to work on and complete one subproblem before moving onto the next subproblem as defined by the sign-off sheet.

Another example came from Bill's class when he instructed students to divide the system into subcomponents as dictated by modular design practices during a class on programming practices. The overall idea of modular design practices inherently breaks an overarching project or problem into smaller pieces or modules that can be used independently, and as such exemplified the well-structured problem-solving approach of problem decomposition. At another point during the same lecture, and also in reference to programming practices, Bill eluded to subproblems when he suggested to the class that, "subsections are arranged this way" while also diagramming the concept of the subsections on the whiteboard. In this sense, Bill was demonstrating his expectation that the students decompose a larger problem through the modular subsections of the programming project.

Dave also enacted the use of subproblems during classroom instruction. Using a series of short exercises integrated into the instructional session, Dave emphasized specific learning goals via subproblems that would later be implemented in the context of a larger project. These exercises were presented to the class, and students were expected to then complete them in a timeframe of a few minutes while the instructor was available to answer questions. In this manner, the instructor presented a subproblem to the students that they brought to a state of completion before moving onto another topic or subproblem.

Each of these observed examples demonstrated faculty modeling the problem-solving approach of decomposing problems without expressly articulating it as such. Rather than explicitly instructing students to engage in the process of defining the subproblems or decomposing the larger problem into smaller problems, the faculty focused on presenting examples where this approach was inherently modeled as the best or preferred way of solving the problem. This means that faculty predominantly engaged in well-structured problem-solving approaches, with an emphasis on one specific aspect in the process: decomposing to subproblems. Although this overall approach of problem decomposition was modeled for students, it still allowed for differing solutions within the subproblems. However, because the emphasis predominantly focused on the decomposition solution strategy within the well-structured approach, the other aspects of well-structured problem-solving approaches such as defining and representing the problem as well as reflecting back to evaluate the effects of the solutions were largely ignored, as were ill-structured problem-solving approaches.

Decomposition as Problem Solving in Faculty Interviews

In contrast with classroom observations where problem-solving approaches were implicitly addressed and only observable in three faculty classrooms, faculty interviews revealed more explicit attention to problem decomposition as an approach to problem solving. Five of the seven faculty noted the importance of decomposing the larger problem into subproblems. John described that his approach was to, “throw em a big problem and say, you find the ... subproblems in there...” (John, personal interview, May 8, 2017). In a similar manner, Chris explained that he expects students to:

dissect the [assignment] and break it up into small parts... I do try to make it for the students to understand that when they program, they should follow the exact

same basic ideas they followed whenever they programmed in any language... go, okay, how do I break this up into multiple tasks? Where do I draw all my functions? What do they do? (Chris, personal interview, May 4, 2017)

In this sense, Chris correlated the idea of problem decomposition to dissecting the assignment and breaking it up into smaller pieces as a means of solving the problems he poses. He presented the idea that students should draw on their previous experiences of using programming constructs such as ‘functions’ as a means to break up the overarching problem into multiple tasks. Dave described that he tries “to teach them that you ought to think about, how could I write a piece of this and get the piece working and then incrementally add on more pieces” (Dave, personal interview, April 13, 2018). He elucidated further by explaining how important it is to incrementally divide the problem into subproblems by referencing his own expertise in the domain:

I’ll even tell them in some of the labs, if you read through them, or homeworks, it’ll even say, ‘If you’re going to do this, get this part working first then move on to this part.’ I try to teach them that incremental approach. (Dave, personal interview, April 13, 2018)

In his reflections, Dave seemed to suggest that students did not always address problem solving in this way. Instead, he suggested that students were resistant to this approach. Dave followed by explaining his own mode of tackling problems:

They’re very resistant, and I don’t know how to get them to be less resistant, that’s another mystery after 40 years I have not solved. It would be nice if I could. If I could get them to understand that if I can get a piece working, great, I know I’ve got that working now. Let’s just not screw that up and add this other piece in.

To me, as a programmer, what I do ... that's exactly how I do it. So, I didn't just make this up out of nowhere. That's how I solve these problems that are a little too big to hold in my head. I have developed an ability, which I'm trying to get them to develop, to see where the pieces are. How do I chunk this up? You can't just slice it anywhere. (Dave, personal interview, April 13, 2018)

In fact, Dave did enact this approach into his classes. As previously described, observations revealed Dave emphasizing learning goals via subproblems that were later implemented within a larger project. His use of short, in-class exercises were presented to the students and completed by the students before moving on to the next topic.

In another example, Bill also expressed his own efforts to get students to break up a problem into smaller pieces or subproblems, specifically by giving students ideas on how to work through a project. His notions aligned closely with Dave's thoughts on using an incremental approach to divide the project into subproblems:

we like to think that at least giving this advice for the projects helps them a little bit to cut it down to smaller pieces that don't seem so overwhelming. Just do this first. If you get that done, do something else. (Bill, personal interview, April 13, 2018)

While Sue still used subproblems, her approach diverged from other faculty; she specifically explained that she used assignment deliverables as subproblems within the context of a larger-scale project:

... I divide it up into deliverables or chunks. And so I'll have a deliverable that's due, and then, as a class, we'll go over the solution to that deliverable, and then

I'll post up the solution for that piece, and then they can use that as the starting point for the next deliverable. (Sue, personal interview, May 8, 2017)

In this regard, Sue described deliberately scaffolding students' problem-solving process and was intentional about ensuring that students would be able to tackle smaller problems. However, it was not clear how Sue helped students to understand that each subproblem contributed to a larger problem/project.

Interestingly, two faculty described frustration with the fact that students are unable to approach a problem by breaking it up into smaller subproblems. For example, Sue explicitly identified that the students have trouble with breaking down a problem from a larger set of requirements or deliverables to smaller subsets when she said:

... and so, ... they're seeing it modeled for them, but then when they go to do the homework, I don't know if it's, if it's overwhelming, if they just try to treat, if they're not able to essentially chunk up the problem...into smaller pieces, and they just try to treat it as one, you know, great big problem... (Sue, personal interview, May 8, 2017)

As previously noted with Sue, classroom observations revealed no clarity on how she helped students to understand that the problem should be decomposed to subproblems that contribute to the larger problem/project.

In fact, of the faculty that described problem decomposition during interviews and in the examples presented above, none specifically described explicit instruction around decomposing problems into smaller subsets of problem. Instead, faculty described modeling the approach of problem decomposition for students by 'throwing them a large problem' that they were expected to deconstruct or expecting students to 'dissect the

assignment.’ This aligned with the instructional classroom and laboratory observations where only three faculty implicitly modeled problem decomposition. Particularly noteworthy was Chris’ description of the idea of modeling a problem-solving approach for students by showing students good solutions and walking them through those solutions because this was the way that he was taught:

... as far as I know, problem solving is sort of taught, is we basically, give them examples, and we walk through examples with them, and then we basically send them away to go do their own. And, I mean, that’s the way I was taught how, um, that’s pretty much problem solving. And I know there’s obviously better ways and maybe new ways of doing this, but really, the bottom line, you basically show them what quite possibly is good solutions to problems and try to walk them through how they came about, and then from then, you hope that they can extend it. (Chris, personal interview, May 4, 2017)

Decomposition as Problem-Solving in Course Artifacts

Consistent with the faculty interviews, examination of course artifacts revealed evidence of decomposing problems to subproblems. Five of the seven courses’ artifacts demonstrated examples of the overarching problem being broken down into smaller subproblems. This was evident in Tim’s hands-on laboratory exercise, as seen in Figure 28 below, where the overall exercise was decomposed to a subset of subproblems and noted as Specifics numbered 1-4.

Lab Three: Active Directory Installation

Prerequisites:

Watch Video/Demo

Print out a hard copy of the Lab 3 signoff sheet and bring it with you to lab.

Update your site book with information pertaining to lab 2.

Objective:

At the end of this lab, you should be relatively comfortable navigating Windows 2012r2 Active Directory Domain Services and using the management console. Understand the value of using a remote access utility like RSAT (Remote System Administration Tool). Learn the different roles that individual users play in an enterprise environment as well as how they fit into the logical infrastructure of a large enterprise environment.

Specifics:

1. Install Active Directory and Domain Name Service (DNS) on Windows Server 2012r2.
2. Create an organizational unit (OU). Add a standard user and Enterprise Administrator account to the newly created OU.
3. Have the Windows 7 client join the newly created domain.
4. Install Remote Server Administration Tools (RSAT) on the Windows 7 client and remotely access Active Directory Users and Groups from that client using the Enterprise Administrator account.

Figure 27. Course exercise demonstrating problem decomposition.

Furthermore, the lab exercise in Figure 28 above was accompanied by the lab sign-off sheet in

Figure 29 below.

Lab 3 – Sign Off Sheet (Due at the beginning of the next assigned lab)	
(10 points) Site Book has been updated with Lab 2 information.	_____
(5 points) Active Directory Domain Services has been installed on Windows Server 2012r2.	_____
(10 points) Two users have been created, one with Administrator rights and a standard user.	_____
(5 points) Windows 7/8 client can successfully join the domain.	_____
(10 points) RSAT has been installed and successfully tested with the administrator account.	_____
Points earned for signoffs (Max 40)	_____

Figure 28. Lab exercise sign-off sheet demonstrating problem decomposition.

This sign-off sheet was used by the instructor to track students' accomplishments in the lab, and further demonstrated the overarching activity being decomposed to distinct subproblems, an approach that was consistent throughout all the lab exercises in Tim's course.

Another example centered on Bill's course where the breakdown of a problem was presented to students as the 'Plan of Attack'. As noted in Figure 30 below, the document for the semester project presented a suggestion for students to approach the problem, with a particular subset of problems including (1) updating the scoring algorithm for the player followed by (2)

working on the computer controlled players, etc.

Unit Testing Guidelines

As with Part 1, you are strongly encouraged to write unit tests for your code. Tests that you wrote previously may break as a result of some of the modifications that you will be making. This is a good thing! Making changes to your code that breaks tests shows that you have tests that covers that code that need to be updated. Modify your existing tests so that they pass using your new code, and add new tests as needed. If your tests are high quality, you may be able to earn back a limited number of lost functionality points.

Plan of Attack

While you are free to tackle the project in whatever order you would like, it is recommended that you implement the main project tasks in the following order:

1. Updated scoring algorithm for the players.
2. Computer controlled players.
3. Railroad Barons: Lonely Edition - 1 human vs. 3 computer players.
4. Railroad Barons: Classic Edition - this should continue to work (albeit with the new player scoring algorithm). If your Part 1 implementation included bugs, you may earn some points back (to a maximum grade of 75%...[see below](#)).

Figure 29. Course project demonstrating problem decomposition as a 'plan of attack'.

Although the instructions clearly indicated that students were free to approach the problem in any way they wanted, this recommendation presented a suggestion in regards to the problem decomposition. In this sense, through the course artifacts, faculty were implicitly modeling the problem-solving approach of decomposing a larger problem into smaller subproblems. The approach of breaking the problem into smaller subproblems was modeled to students through the suggestions as in the Lab 3 example in Figure 28 or even required through the use of the instructor sign-offs in Figure 29, yet it was never explicitly identified to students that the larger problem was being decomposed.

Table 25 below summarizes the findings in regards to problem decomposition as the predominant approach to problem solving as demonstrated by the data sources.

Table 25
Problem Decomposition Findings Summary

	Decomposition to Subproblems
Instructional Observations	3/7 courses implicitly modeled
Faculty Interviews	5/7 faculty explicitly described
Course Artifacts	5/7 courses implicitly modeled

The Nature of Collaborative Learning in Undergraduate Computing Education Courses

This section addressed how aspects of collaborative learning were realized in undergraduate computing courses. Collaborative learning was examined based on both instructor-student and student-student interactions. Analysis of instructional classroom and laboratory observations, faculty interviews and course artifacts indicated that collaborative learning was informal and unstructured in computing education courses. For student-student interactions, the findings revealed that collaborations were primarily determined by faculty's effort to (a) require student group work and thus was explicit as part of classroom enactment and assignments or (b) require individual student work resulting in prohibited or ignored group work due to the nature of the assignments. Findings also revealed that faculty fostered a classroom environment where they modeled the role of a facilitator to encourage instructor-student interactions. However, during interviews, faculty expressed a tension between the value of collaboration to students as they enter the workforce and the need for development of individual student skillsets. Faculty expressed that group work can sometimes sabotage development of individual content knowledge skills; that is, contributions to group work may not always be equitable and thus some students may rely on their groupmates to develop solutions as opposed to meaningfully engaging with the problem(s) themselves. This suggested that faculty held limited understanding of how effective group work can be structured such that students' contributions could reflect their individual content and skill strengths.

Instructional Classroom and Laboratory Observations and Course Artifacts

Instructor-student collaborative learning. Classroom observations revealed that all seven of the faculty modeled instructor-student collaborative learning by using social constructivist scaffolding techniques including instructor feedback, coaching, modeling,

questioning, procedural prompting, conceptual modeling, providing hints, presenting examples and stepping through problems. In this regard, faculty enacted their understanding of the value of collaboration, albeit through informal and unstructured means.

Faculty commonly used questioning techniques to encourage instructor-student interactions. Faculty also solicited and welcomed questions from students that they followed by providing thoughtful responses often serving as part of a larger classroom dialogue. This regularly occurred at the beginning of class and continued throughout the class sessions. All seven of the faculty engaged in the process of posing additional questions to students throughout the sessions to further encourage collaboration through instructor-student interactions. During one observation, Joan began the class by asking for questions. This was followed by a student asking for clarification on a topic by starting with, "I was kind of confused about ...". Joan addressed this student's concern when she provided coaching to the all students in the class by reminding them about what had been previously discussed in the last class and then using questioning to ask, "So how do those two relate to each other?"

Joan continued on to scaffold and model collaborative social constructivist approaches with the class by walking students through an exercise in a step-by-step fashion. She accomplished this by continuously asking questions and waiting for student responses before moving onto the next step in the process. For example, Joan questioned students about how to represent a specific and critical aspect of the project design when she asked, "how can you tell where is the primary key?" One student answered, and then Joan continued the instructor-student collaboration by asking if there was anything else 'we' should do in this example. In this example, Joan modeled collaborative learning by using questioning to coach the students to step through the problem.

Similarly, Bill fostered an environment of facilitating dialogue and collaboration when he began class with a question and answer exchange with students. One student asked, “so is lab 8 event driven?” to which Bill answered, “yes because you have events from the network.” The dialogue regarding these details of the project continued with Bill also asking questions of the students such as, “how do you do a design so that you cannot impact the rest of the program very much?” In this manner, Bill also coached the students to think through the problem for themselves by using questioning. John demonstrated a similar questioning approach when he began class by asking students, “so what does factory method imply to you? What might its intent be?” In this regard, faculty used their role as instructor to facilitate, engage in and model collaborative learning with and for students in the classroom environment. By using questioning, the faculty attempted to facilitate a classroom learning environment where students were encouraged to think through problem solutions and come to their own conclusions. However, while their engagements appeared thoughtful and intentional in regards to engaging students, interactions were primarily enacted informally and in an unstructured manner as part of the instructor-student exchanges.

Student-student collaborative learning. While all seven faculty consistently fostered a classroom environment that modeled collaborative learning through instructor – student interactions, classroom enactments for student-student collaborations varied with (a) three courses explicitly requiring student-student collaboration as a result of classroom enactments and course assignments and (b) four courses prohibiting or ignoring student-student collaboration as a result of classroom enactment and the nature of the coursework. For the classrooms where student-student collaborative learning was required and therefore enacted, students were

commonly on their own to work through collaborations, aligning with the informal and unstructured nature of the interactions previously noted in the instructor-student exchanges.

Required student-student collaborative learning. The three faculty intentionally fostering an environment of student-student collaborative learning explicitly set aside classroom time for the distinct purpose of group work that also enabled faculty interactions with the students. Of the three faculty who did this, two facilitated the required student-student collaborative learning on a regular weekly basis and the third was less regular as the group work was specific to two lab assignments and a final semester project. In this third case, student-student learning was also allowed, but not required, during weekly lab exercises.

While the collaborative learning was required and explicit in nature across these three courses, it was also primarily unstructured with students simply being left on their own to work as part of a group. Classroom observations and artifacts for the courses requiring student group work confirmed the unstructured nature of student-student collaborations through the omission of any specific collaborative learning instruction. In Bill's class, students were required to gather together for a weekly problem-solving session. Yet, observations revealed that no instruction was presented to students on how to engage with one another, either in-person or through the course artifacts. Instead, at the beginning of the session, students were simply assigned to groups and presented with a document regarding the technical details of the problem to be considered. Examination of the course artifacts confirmed this and Figure 31 below demonstrates the limited extent of guidelines for the collaborative work, whereby students are simply instructed to work in groups to answer the questions that follow.

2 Problem Solving (20%)

You will work in groups to answer the following questions on pen and paper. Make sure everyone in the group puts their name of the worksheet before handing it in.

1. Here is a table that lists the constants that are used by for the various heroes when attacking and taking damage.

Figure 30. Course exercise demonstrating limited guidelines for collaborative work.

John's class, which also required student-student collaboration, was similarly informal and unstructured in nature regarding collaboration. While John did enable group work during scheduled classes by specifically setting aside time, no instruction was presented to students on how to participate in the group work. Instead, John only informed students that it was time to break into groups to work on their projects. Similarly, course artifacts revealed that while collaboration was required, no instruction on how to work in groups was presented. Some documents merely referenced that the assignment required team submissions as an indicator of collaboration. This was similarly noted in the syllabus which explained group work with the text presented below in Figure 32. Here, the method of instruction simply indicated that students would work together as teams throughout the semester and that some class time would be used for the team project.

Method of Instruction:

This course will combine a small amount of lecture, student presentations, and many in-class activities. Team project work will exist throughout the entire term. Some class time will be used for team meetings and discussions between the instructor and teams.

Figure 31. Course syllabus demonstrating limited instruction on collaborative work.

In this manner, all three of the faculty requiring student-student collaboration fostered a classroom environment that enabled the collaborative interactions through time set aside for the work. Likewise, course artifacts revealed that assignments were to be completed and submitted

as groups. Yet students were primarily left on their own when working together, resulting in informal or unstructured collaborative learning experiences.

Prohibited or ignored student-student collaborative learning. In four courses, faculty either prohibited or ignored student-student collaborations. Three of the four faculty enacted a classroom environment that did not allow for student-student group work and required individual submission of all assignments. These classrooms enacted a traditional teacher-centered environment with the faculty leading the classroom lessons. The fourth faculty member allowed for group work by ignoring student-student interactions, yet specifically required individual assignment submissions, fostering an environment that encouraged individual student work.

The 'ignored' collaborative learning experiences took the form of informal student groupings to discuss and work through exercises or project work during lab sessions, and emerged in observations as overlooked by the instructor. Students in Tim's hands-on laboratory session were observed informally engaging with one another as they worked through the lab exercise. Students asked each other questions and discussed potential solutions and next steps in working through the problems. During the lab period, Tim neither encouraged nor discouraged this practice and continued to work with students in a one-on-one fashion as they raised their hands with questions or for lab sign-offs (as previously described). Yet artifacts revealed that no group submissions were allowed for any assignments, emphasizing the importance of individual work.

The absence of student interactions in classroom observations revealed that classroom enactments in three courses did not foster an environment supporting student-student collaborations. In these courses, faculty members conducted their classes in a traditional teacher-centered fashion, while facilitating instructor-student interactions as previously described.

Students primarily engaged with the instructor in asking and answering questions, but no peer-to-peer interactions were observed. Course artifacts supported this finding as individual student submissions for assignments were required. In one case, course artifacts were even more extreme to indicate that any student collaboration on coursework was forbidden. Joan's and Sue's course explicitly outlined in their syllabi the expectation for individual assignment submissions as noted in Figure 33 below.

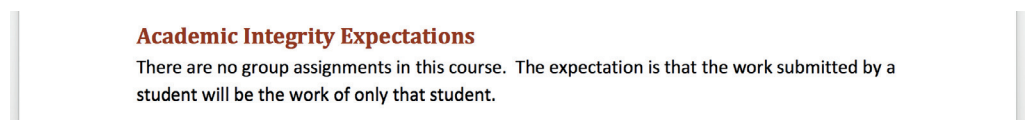


Figure 32. *Syllabus emphasizing individual work.*

Chris' syllabus, seen in Figure 34 below, framed the requirement for individual work within the context of academic conduct, making clear that any student-student collaboration of any nature, even discussing possible problem solutions, was forbidden. Although a distinction was made in regards to discussion of the assignment versus the solution, the syllabus clearly indicated that violations of this policy would be considered academic dishonesty.

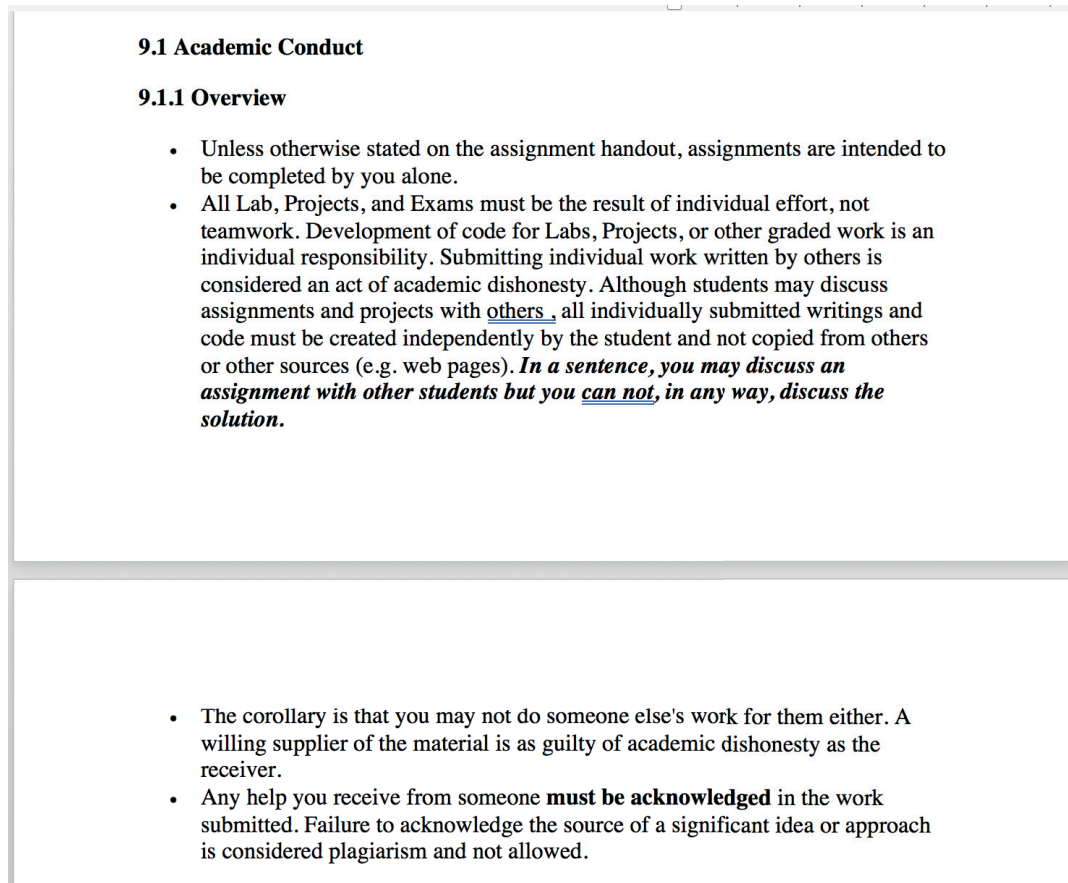


Figure 33. Syllabus demonstrating that collaboration is forbidden.

Faculty Interviews

Instructor-student collaborative learning. In describing their instructor role in working with students as part of the problem-solving process, six of the seven faculty indicated that they viewed their roles as instructors in terms of a coach, guide, counselor, facilitator, and mentor. Note that collaborative learning in this case was based on the role of the faculty and how they perceived engaging in the classroom as a social space. More specifically, these roles were teacher-centered, focusing on actual instruction, and aligned with the instructional classroom observations. In this regard, six of the seven faculty embraced the idea of setting the classroom up as a collaborative learning environment to foster collaboration. Bill described himself in working with students as, “I see myself as the facilitator, going around, do you understand and

all this” (Bill, personal interview, April 9, 2018). Sue explained her role in providing feedback and presenting examples to students as being “... like a tour guide. So, I kind of approach misdirections but then I also, divide the deliverables up or, or chunk things in such a way that they’ll have repetition” (Sue, personal interview, May 8, 2017). Dave saw his relationship with students as emulating the master-apprentice model, particularly while modeling problem solving by presenting problems and stepping through examples. He explained:

...it’s more of an apprentice kind of approach where I’m going to do it. Watch me do it. Now, I want you to do it a little bit. I’m going to watch you do it. Then, it’s just plain, okay, now do it without me. We try to sort of ease them into it a little differently. (Dave, personal interview, April 13, 2018)

Joan expressed her view on her role as a coach and stepping through problems:

So, I’m sort of a coach, like, ... I can tell when they’re really stuck and they’re just not they really can’t get the answer. So, I’ll go back to the beginning. Sometimes we just go back to the fundamentals. That’s always a good thing to do. (Joan, personal interview, May 1, 2017)

Sue also explained that she likes to use questioning in facilitating collaborative learning with the class:

and then I sit, and I wait, and the general, uh, culture that I have in the classroom is that when I pose a question to the class, it’s open for the entire class to answer. I’m not looking for individuals to raise their hand, who I then call on and they give the answer. I want multiple people to state what they feel the answer is, cause then I can kinda hear the predominance of the correct answer versus an incorrect answer. (Sue, personal interview, May 8, 2018)

She continued on to add, “I get feedback from them based on the interactions that we have during class while we’re discussing the solution” (Sue, personal interview, May 8, 2018). John described visually representing a system through conceptual modeling (in the form of the Unified Modeling Language (UML) diagrams), to walk students through a problem-solving project during class:

We kinda walk them through the steps. So, the first thing they develop is what we call a domain model, which ... gets captured as a, as a UML diagram, but it’s trying to capture it in a form that they can then talk to the customer about understanding the requirements. (John, personal interview, May 8, 2017)

In this example, John viewed his role as a coach to model a particular problem-solving approach to students within the social space of the classroom, albeit through an informal and unstructured means as the distinct approach was not delineated.

In a similar manner, Dave explained how he coaches students to work together informally on problem solving during in-class programming sessions. While Dave perceived a clear intent of having students share their experiences in the social space of the classroom, his explanation also indicated his intent may have been unclear to students:

I’ll tell them, when I go around to working on a particular problem, and I’ll say, ‘Who thinks they’ve got this solved?’ Then, I’ll come and I’ll look and see how they’re doing. If the guy next door didn’t put his hand up, I’ll say, ‘Why don’t you show him what you did?’ I try to encourage that. It’s successful up to a point. Some students who need the help just don’t want it because they feel ... I don’t know what they feel. I would guess that they feel that’s some sign of failure on their part, where the real failure is to not learn what you’re not doing right. That’s

the real failure, but they feel ... Some students seem to feel that if they can't do it on their own, they've failed, so they want to do it on their own. (Dave, personal interview, April 9, 2018)

Student-student collaborative learning. Faculty interviews were consistent with instructional classroom observations and artifacts in revealing that any student-student collaborations occurring were informal and unstructured in nature. Additionally, despite the finding that only three faculty supported student-student collaborative learning through classroom enactments and artifacts, interviews revealed that six of the seven faculty expressly recognized the value of students engaging in collaborative learning through group work. However, six faculty also perceived collaborative learning or group work to be at odds with developing individual student skillsets. As such, these varying perceptions of collaborative learning in the context of problem solving revealed faculty perceptions of an overarching tension between student collaboration and the building of individual domain-centered skillsets as students prepare to enter the workforce.

Required student-student collaborative learning. As previously noted, in three courses, collaborative learning manifested as a result of course assignments and was supported by faculty enabled class time for group work. During interviews, the faculty requiring these student-student collaborations emphasized the informal and unstructured nature of the collaborations. More specifically, through the informal implementation of collaborative learning, faculty identified their own shortcomings in preparing students to work together. Bill who required weekly collaborative problem-solving sessions, offered that students “probably need more experience and more training in how to work on a team than we give them” (Bill, personal interview, April 9, 2018). John, who allocated weekly time for group project work and meetings, confirmed the

idea of informal collaborative learning when he described the broader approach of how the faculty enable students to work in groups. He explained a hands-off approach where faculty simply allow students to work through problems by putting “them in the situation where they have to work it out... You know, but it’s, it’s more of, you know, let them have the experience... and hope it’s a good experience for them, at least a... supervised experience” (John, personal interview, May 8, 2017). John further emphasized the hands-off approach in allowing students to learn to work together as part of a group:

as an experiment in a controlled environment... I’m not gonna say, hey, have you thought about... doing this? Or have you thought about doing that? You know, if there is I see an issue, I’ll give them some hints as to what to do... (John, personal interview, May 8, 2017)

In this example, John emphasized the unstructured approach to student-student collaborative learning. By allowing students to independently work through the experience of being part of a group, John acknowledged that group experiences may or may not prove positive for all students.

Dave, who also required group work, explained that while the goals are to simulate a work environment where group work is necessary, the implementation may be ineffective:

The group dynamics are, this is probably what we’re the worst at, is to know how to really make them function as a group. It’s supposed to be, theoretically, a simulation of a work environment, but it really isn’t ... if you’re working for a company, and they’re paying you, and that helps keep you alive, and you just say, ‘I don’t think I’m going to go to work today,’ there’s a real cost to that. Whereas, if you’re working on a group project, and you think, I don’t want to go to that group project meeting today, I’m just not going to go, what’s going to happen?

They're going to say, 'This guy was a bum,' at the worst. Usually they don't. It's amazing. They won't turn on each other too badly, no matter how badly they behave. (Dave, personal interview, April 9, 2018)

In this sense, Dave began to reflect on the inadequacies of implementing collaborative learning through group work, yet he stopped short of indicating the overarching benefits of the collaborations. Even Joan, who did not facilitate student-student collaborative learning in her course expressed frustration in how group work was managed in other courses. She explained, "We just throw them into groups and ... hope for the best" (Joan, personal interview, May 1, 2017). She continued to offer that some instructors do give more direction on how to work in groups, but explained that it is not part of the curriculum. In this sense, even faculty exhibiting some appreciation of the value of collaboration demonstrated a lack of understanding as to the overall benefits of the collaboration or the means to realize those benefits.

Despite the informal or unstructured nature of the group work, the three faculty members requiring student collaboration also recognized its importance in the workforce, although with varying degrees. Bill distinctly articulated a strong sense of the value of collaborative group work in the context of problem solving when he said, "the goal of the problem solving is for them to share ideas and come up with a mutually satisfactory solution" (Bill, personal interview, April 9, 2018). He continued on to highlight the importance of collaboration when students enter the workforce, yet contrasted this notion by emphasizing the significance of individual work:

We have tried very hard to draw a line in a reasonable place, and that is hard, in that we recognize, and we tell them, this is not a career, or I should say a profession, where you work by yourself. However, we need to assess how well you individually have learned the material. Basically, we say don't copy code,

and we allow almost anything else... One person does this part, the other does this part, but you're going to fail if you're not intimately familiar with each other's parts. (Bill, personal interview, April 13, 2018)

Dave also recognized the value of individual skillsets in contrast to collaboration. In doing so, he emphasized collaborative work more as a form of collegiality specific to troubleshooting in the context of problem solving than as an overarching valuable skill to enhance problem solving. He offered:

Well, if you're going to be a good problem solver, you have to be able to function as an individual up to a point, but I think sometimes, I really do believe this, sometimes you just get stuck, and you can't see the forest for the trees. You're looking at this particular problem. It certain[ly] has happened to me many times. You're banging your head against the wall. You can't. Why isn't this working? You'll get a colleague or somebody to come in, take a look at it, fresh eyes. They take a look, and they say, 'Oh, it's right [there], and you fix it, and by god, it's gone. I think good problem solving has to be to a certain extent collaborative or at least, maybe collaborative isn't the right word, but at least supported by collegial interaction. (Dave, personal interview, April 9, 2018)

Prohibited or ignored student-student collaborative learning. Aligning with the classroom observations and artifact examinations, all four of the faculty who prohibited or ignored student-student collaborative learning emphasized the importance of individual domain-centered student skillsets. However, in contrast to this, faculty interviews revealed varying perceptions of prohibiting or ignoring student-student collaborations. Of the four faculty members disallowing or ignoring student collaboration, one faculty member ignored the idea of

students working together yet emphasized individual work, one faculty member explained how she encouraged students to work together informally outside of class on assignments, one faculty member expressed that she allowed students to work together only on ungraded assignments, and one faculty member viewed collaboration as a sign of academic dishonesty. Furthermore, three of the four faculty expressed perceptions around the positive value of collaboration to the workforce that appeared incongruous with these overall notions emphasizing individual skillsets over collaboration.

Tim, who allowed students to work together informally during lab sessions and afterward by ignoring the interactions, offered, "... I know that some students tend to pair up. So... they will feed off each other...in pairs," implying a positive perception of the collaborations (Tim, personal interview, April 11, 2017). Yet, in contrast, Tim also emphasized the importance of individual student skillsets by explaining that when he sees students informally pairing up in the lab, he is sure to note when one student might be sitting as a backseat driver and letting the other student take the lead. Tim described that he explains to students that they need to step up and also be in the lead or they will have trouble when it comes to doing the hands-on laboratory exam. He expressed this in terms of the importance of being a good or exceptional student by developing an individual skillset in the form of self-reliance,

one thing I've observed with students, with good students is they are self-reliant.

And I know there's other students, there's some students that are out there that just aren't, just don't wanna do it. But then there's students in the middle that I think if we force them more and more to be more self-reliant, it's, it's like anything, you know, it's like a sport, okay? It's... the more you train, the more efficient you become at it, the easier it becomes for you, and I think, given this,

you know, this four-year window of when they're being educated, I think at some point, they've just gotta become... I think these middle students that become more self-reliant will get there much quicker to being exceptional students. (Tim, personal interview, May 2, 2017).

Tim further echoed the value of individual skillsets by also emphasizing independent learning.

He offered that if possible, he would:

force these students to, to develop the habits to learn things on their own ... When I was an undergrad, I almost wish that somebody had said to me, look, you've gotta learn this stuff on your own, you've gotta, and I mean, you know, in a, to a certain extent, you know, when you write a paper, you research it on your own, but, I just, I, if it was up to me, I'd be like, okay, you're gonna go into this lab, I'm gonna leave, and you're on your own. (Tim, personal interview, May 2, 2017)

Joan, who disallowed collaborative assignments, also expressed the importance of individual skillsets in regards to the assignments. She explained that projects in her class are required to be:

... individual. I don't like them working in groups because when they work in groups, there's one person ... doing all the work. That's my experience... and one person is not doing the work, or not enough work, and I want to make sure that ... every one of them have ... gotten to a certain point in the course in order to give them a grade. And I can't tell that with group work. (Joan, personal interview, April 12, 2017)

Inconsistent with this, Joan also explained that homework assignments in her course were individual,

but they, they work together a lot...and I don't mind it. They don't have the answers and, and they have to work out the problem, so they sit together, um, and usually a couple students will come in at the same time and have similar questions, but they're working with each other. (Joan, personal interview, April 12, 2017)

In further contrast with her previous assertions of the importance of individual skillsets, Joan explained that she encouraged students to work together outside of class, and described that some of the students come into office hours as a group because they had obviously been working on a problem together. She expressed her support of this approach and welcomed students to her office hours to work through the problem as a group.

Sue, who also did not facilitate student collaborations in her course, offered that over the years, she had moved toward making all of the assignments individual because she, "wanted to make sure that everybody was trying to do everything." She continued to say that their chances of practicing every skill were less if they were part of a group where they could "skirt their way through." Sue further explained that she did not use group assignments because ... "different people have different strengths, the work would need to be divided, and so there's no guarantee that everybody had... contributed to every portion of the assignment" (Sue, personal interview, May 8, 2017). Sue had in fact shifted the assignments over the course of her teaching to be individual because she "wanted to make sure that everybody was trying to do everything." In this sense, Sue emphasized the value of individual skillsets as critical to students in her course. In contrast to this, Sue expressed an understanding that there is value in having students work together:

It's just been kinda something that, through the years, I've just kind of moved away from. I know that working as part of a group is important and it's a critical skill to have, um, but at the same time, I need to make sure that they know the topics that they need to know in the course. (Sue, personal interview, May 8, 2017)

Sue also offered that there were optional homework assignments in her course that were not submitted for credit, and for those assignments students were welcome to pair up together. This notion of working together on ungraded assignments further supported Sue's previous perceptions regarding the critical importance of ensuring individual student skillsets, while also indicating her sense of the value of collaboration in preparing students for the workforce.

Chris, who did not facilitate student collaborations in his classroom enactments and explicitly forbade interactions through his syllabus, emphasized his rationale with the idea that collaboration was indistinguishable from cheating:

My syllabus says strictly, my problem is collaboration really comes down to cheating. What's the difference, the students and, lots of people have a problem with the difference between collaboration and cheating. Working together to the solution versus handing the solution to somebody else, or having somebody explain the solution line by line to you. And so, in my syllabus, I am very draconian. I tell them that they're not even allowed to talk about the solution with another person. (Chris, personal interview, April 12, 2017)

A distinct friction between collaboration and individual skillsets emerged as Chris explained that collaboration was indistinguishable from cheating while also recognizing the value of collaboration to the workforce. Chris offered:

I really think it's very, very important, because students, obviously, when they get in the real world, nobody's gonna be working alone. I find it really something important, now the problem comes in is, when you have a very well-defined assignment, it becomes impossible to distinguish between cheating and collaboration. (Chris, personal interview, April 12, 2017)

Chapter 5

Discussion

The purpose of this naturalistic exploratory case study was to examine the ways in which computing faculty are preparing students to become domain problem solvers and effective team members. Central to this understanding were the three research questions:

- R1. What kinds of problems are posed by computing faculty in their process of teaching and learning in first and second-year computing courses?
- R2. What instructional approaches do undergraduate computing faculty use as a mechanism to facilitate teaching and learning problem solving in first and second-year computing courses?
- R3. In what ways are aspects of collaborative learning realized in undergraduate computing faculty instructional approaches to problem solving in first and second-year computing courses?

This chapter includes a discussion of the three major findings as they relate to the literature on problem types, problem-solving approaches and collaborative learning in the context of social constructivist theory and STEM education followed by the implications of these findings for post-secondary computing education professionals. This chapter concludes with a discussion of the limitations of the study and areas for future research.

Summary of Findings

First-year and second-year instructional classroom and laboratory observations, in-depth faculty interviews and course artifacts served as the main sources of data for this study. A problem-solving typology (Jonassen, 2000) and problem-solving literature alongside social constructivist theory, served as the framework to examine the above practices. Recalling the

workforce demands of problem solving, teamwork, and communication as articulated by computing employers (Archer & Davison, 2008; Robles, 2012; Vivian et al., 2016), the three key findings are critical in considering the future of computing education: (1) problems presented to students in course artifacts and classroom instruction were predominantly well-structured in nature, with faculty emphasizing during interviews the importance of real-world problems and allowing for multiple solution paths while focusing on better or worse solutions, (2) consistent with course artifacts, instructional approaches to problem solving were also well-structured, with faculty implicitly modeling problem decomposition to students and (3) collaboration was informally modeled through instructor-student interactions in the classroom along with some limited and unstructured student-student group work in solving problems, with faculty focusing on building individual student skillsets.

Problem Types: Computing Faculty Predominantly Enacted Well-structured Problems

The first major finding for this study revealed that computing faculty primarily posed and emphasized well-structured problems to first-year and second-year computing students. Newell et al. (1958) explained that well-structured problems are grounded in generalizable information processing theory and as such reflect more traditional problem types in computing. In converse, ill-structured problems address everyday life and are grounded in constructivism with domain specificity using authentic contexts (Jonassen, 2000). In fact, the well-structured problems noted in this study focused on the use of algorithmic, rule-using, troubleshooting and design problems. The enactment of these four problem types is consistent with the nature of computing, and the emphasis on algorithmic and rule using problem characteristics in this study aligns with two key constructs integral to computing: algorithms and protocols. A computer algorithm is a set of operations that are executed to accomplish an objective or accomplish a task (Fincher & Petre,

2004). For example, one of the problems used in Dave's course focused on sorting algorithms, a key construct in the computing domain where data sets are ordered according to specific criteria. This example reflects the idea that computer algorithms are considered fundamental to all of computing, and as such are one of the first topics studied in the domain, as mastery of this construct facilitates further study of other computing subdomains. Additionally, a computing protocol is an agreement between two parties on how communication is to proceed, specifically around the rules that will control the system (Tanenbaum & Wetherall, 2014). Protocols govern the communications for computing systems that facilitate information transfer and processing. As such, rule using problems inherently encompass instruction around protocols. An example of this was found in Tim's course with the focus on the DHCP protocol (as in Figures 20 and 21), that governs the setup of the computer's communication information. Computing systems also inherently require troubleshooting in the configuration, building, securing, managing and daily operations as well as the design before-hand. Thus, the four problem types that emerged as dominant in the findings are consistent within the context of the computing domain and computing education. This suggests that the faculty in this study emphasized problems that are traditional and developmental in this domain.

The above finding is important for computing education because of the limited research in this domain. The focus on well-structured problem-type characteristics suggests that first-year and second-year students are being prepared to engage with well-structured problems. This may prove appropriate as faculty work to prepare students in building their domain skillsets and problem schemas at a fundamental level (Chi, Feltovich, & Glaser, 1981; Heyworth, 1999).

More specifically, computing faculty clearly recognized the importance of authentic and real-world problems in the computing domain. In this sense, computing faculty emphasized an

understanding of preparing students for the applied nature of the computing domain by presenting them with authentic and real-world problems. Indeed, faculty focused on real-world problems in terms of well-structured problems. Again, this faculty approach of presenting real-world computing problems within a well-structured context may be entirely appropriate given the developing skillsets of the first-year and second-year students. In fact, this approach leaves room to explore more authentic problems as ill-structured in nature during advanced third-year and fourth-year curriculum as domain specific schemas become more developed. Jonassen's (2000) problem-solving typology, taxonomic in nature, was designed with the expectation of well-structured problems serving as a prerequisite to ill-structured problems, and as such, exploring well-structured computing domain problems prior to ill-structured problems may prove an effective curricular approach (Jonassen, 2000).

In presenting well-structured real-world contextualized problems, faculty did not single out one particular problem type in the problems posed to students, but rather they emphasized characteristics across four problem types. This may suggest that faculty did not have a clear understanding of the differing problems types, but more strongly suggests that faculty posed problems to students that they understood to convey the most relevant aspects of the contextualized domain content they were instructing. As such, the characteristics in the problems presented must not be ignored and furthermore should be considered as critical to learning and understanding in the domain.

As Jonassen's (2000) problem-solving typology was designed with the expectation that categories were neither absolute, nor discrete, nor independent, nor mutually exclusive of each other and that additional problem types and categories may emerge, a new computing problem type may be appropriate given this study. The fact that Jonassen (2000) intended fluidity across

the typology categories in the sense of considering new problem types supports the notion of explicitly combining the salient aspects of algorithmic problems, rule-using problems, troubleshooting problems and design problems to form a new computing problem type. More specifically characteristics across the three emphasized attributes of learning activity, success criteria and context that emerged across all the data sources and as noted in Table 24 are relevant. For example, in regards to learning activity, a new computing problem-type might integrate both procedural processes as with algorithmic problems and producing a correct answer as with rule-using problems. Furthermore, the problem would likely also include troubleshooting learning activities where the system would be examined, tested and evaluated, all while integrating the design problem learning activity of acting on goals to produce artifacts.

Similarly, the success criteria attribute for a new computing problem type would likely include characteristics across the four emergent problem types such as: an answer matching in value as with algorithmic problems, multiple solutions paths as with rule-using problems, fault identification and isolation as with troubleshooting problems and better or worse solutions as with design problems. In regards to the context attribute, a new computing problem type could exhibit both abstract characteristics as with algorithmic problems alongside real-world characteristics of rule-using problems, troubleshooting and design problems. Figure 35 offers a visual of prospective characteristics for inclusion in a new computing problem or set of problem types.

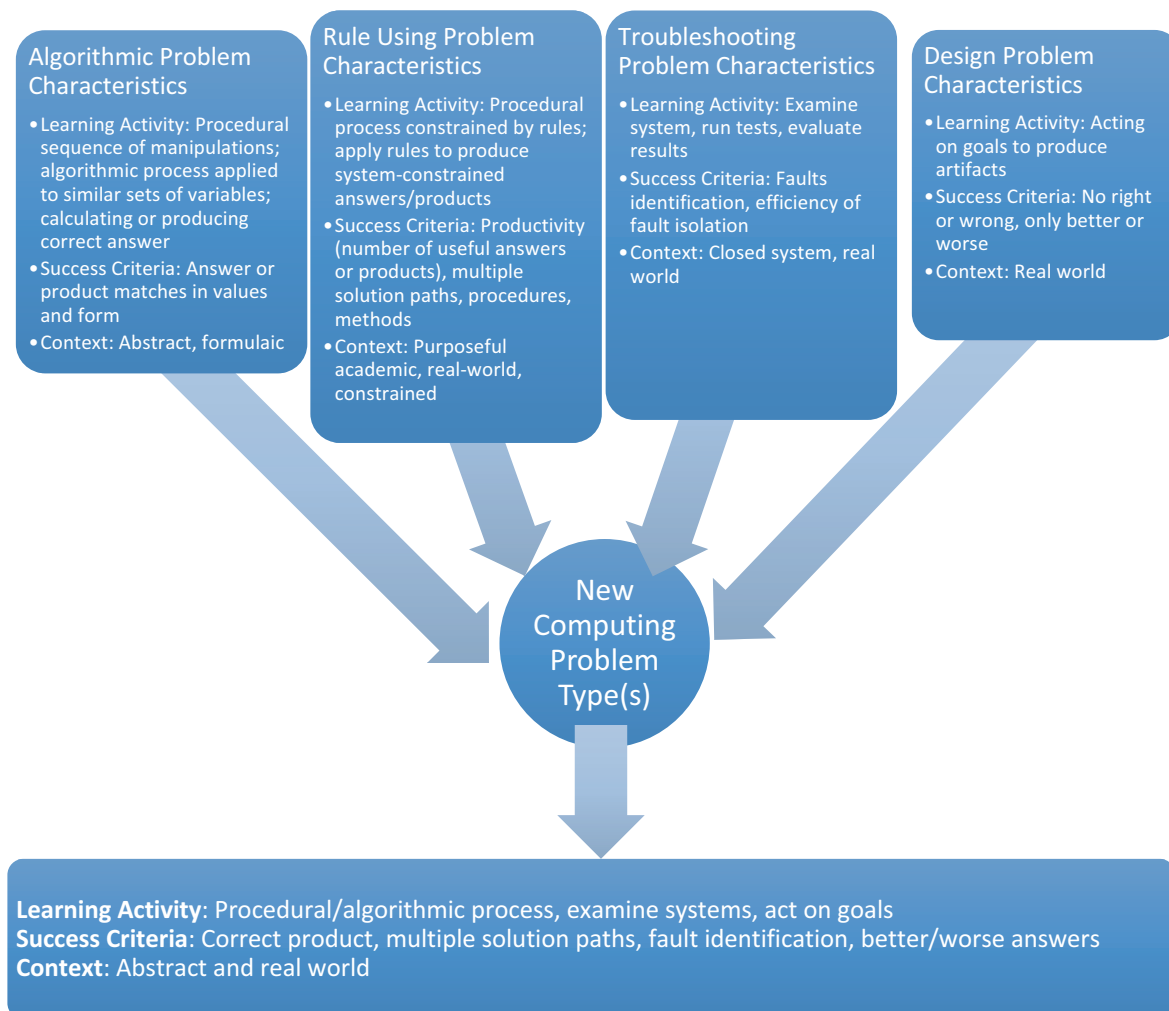


Figure 34. Prospective new computing problem type characteristics.

Specifying a new computing problem type is important to the computing education domain because it recognizes the complete set of the most relevant problem aspects considered as critical to learning and understanding by educators in the contextualized domain. Moreover, a new computing problem type explicitly identifies these aspects in a manner that can be attended to by computing faculty in terms of problems posed and instructional enactment, rather than the potential for unintentionally omitting particular elements or implicitly addressing them. While further examination is needed, this potential new computing problem type emphasizing the

salient aspects of problems in the computing domain may prove particularly relevant when considering the distinction that well-structured problems versus ill-structured problems call on differing design models and differing skillsets and may be considered in regards to problem-solving approach and collaborative learning (Jonassen, 1997).

Problem-Solving Approach: Computing Faculty Implicitly Decomposed Problems as a Mechanism to Facilitate Problem Solving

This study revealed that faculty instructional approaches to problem solving focused on modeling problem decomposition, and this involved a larger problem being broken down into smaller problems or subsets of problems. However, faculty implicitly modeled this approach rather than explicitly defining or identifying it for students. Decomposition as a well-structured problem-solving approach is grounded in information processing models for problem solving, and as such is aligned with the overall well-structured nature of the problems posed to students. More specifically, problem decomposition is noted as one of the strategies for exploring possible problem solutions in the previously described problem-solving steps of (1) defining and representing the problem, (2) exploring possible solution strategies (e.g., recall analogical reasoning; means-end analysis; decomposing to subproblems; and generate and test), and (3) implementing strategies and reflecting back to evaluate the effects of the solutions (Gick, 1986; Jonassen, 1997). In considering the exploration of possible problem solution strategies as with step 2, it must be noted that because students generally lack extensive background knowledge and experience to suitably engage with potential solutions, specific strategies must be implemented. Problem decomposition serves as one of these strategies whereby a problem is repeatedly divided into smaller problems until they are small enough that a solution becomes apparent.

For this study, the decomposing to subproblems strategic approach was implicitly, rather than explicitly modeled for students by faculty. All of the faculty, as domain experts, implicitly enacted problem decomposition by breaking down the problem into parts, presenting hints and suggestions to students as well as laboratory sign-offs, referencing project subsections, and providing incremental assignment goals. Bill described his efforts in providing advice to students in breaking the projects into smaller pieces for students and Sue explained using project deliverables to ‘chunk’ up the assignment into pieces serving as the starting point for the next deliverable. In this respect, faculty valued the role of discrete steps and procedural approaches to problem solving, but faculty did not explicitly identify that their approach involved problem decomposition. Even more important, they did not explicitly identify the specific steps to solving these problems. So, given that problem decomposition was not explicitly named and the specific steps with smaller problems were not identified, it begs the question, how did students engage with their process of problem solving? This question is important because Jonassen (2000) explicated that problem-solving competencies are distinct from component skills or skills specific to a domain. This suggests that development of these skills would thus require more explicit instruction that help students to visualize and understand their process in finding solutions to computing problems.

Indeed, computing education research has noted that intentional rather than implicit approaches to teaching problem solving by delineating general strategies into more discrete steps as subgoals improve learning (Atkinson & Derry, 2000; Dhillon, 1998; Morrison et al., 2015; Peterson & Treagust, 1998; Schraw et al., 2006). These findings are also noted in the closely aligned STEM domain of mathematics (Catrambone, 1996, 1998). What remains unclear is whether or not the implicit nature of modeling problem decomposition is a comprehensive and

effective instructional approach. While more work has yet to be done in this area, the current study emphasized faculty understandings underscoring problem decomposition as an effective instructional approach for problem solving in computing education.

Collaborative Learning: Computing Faculty Emphasized Informal and Unstructured Collaborative Learning Among Students

All seven faculty in this study embraced social constructivist approaches through scaffolding and modeling faculty-student exchanges in their classroom environments. They scaffolded students by using a support system of questioning and dialogue that assisted the students as learners in internalizing the knowledge and ultimately enabling them to solve the problem at hand (Vygotsky, 1930-1934/1978). In this way, faculty demonstrated their understanding of the requirements and stages through which students travel on their journey to reach understanding in a socio-cultural space (Adams, 2006; Ormrod, 1995; Vygotsky, 1930-1934/1978). Indeed, via interviews, six of the faculty further embraced social constructivism through their perceived and self-described roles as coach, guide, counselor, facilitator and mentor. In contrast, only three of seven faculty embraced student-student collaboration in their courses and intentionally allocated class time for group work, indicating an overall lack of commitment to the effort. So, while faculty articulated some of the tenets of social constructivism such as redirecting the instructor's attention toward creating a safe learning environment where knowledge co-construction and social mediation are critical, there were missed opportunities with facilitating student-student collaboration.

Yet regardless of whether enacted as instructor-student interactions or student-student teamwork, collaborative learning was informal in nature in first-year and second-year computing courses. Both the lack of intentional student-student collaboration in four of seven courses, as

well as the informal or unstructured nature of the collaborations that did exist, lead one to infer that faculty understandings of the value of a collaborative learning environment fell short of realizing potential benefits to students.

By omitting or limiting collaboration in a course, educators risk losing an opportunity for students to engage in the process of observing varied thinking strategies, a key construct of social constructivist learning. This may very well limit students' independent and individual problem-solving skill development, identified by faculty in this study as critical to graduates, and as noted in the aforementioned ACM curriculum guidelines and workforce expectations as being at the core of all the computing domains. In fact, albeit well-intentioned, faculty in this study revealed a misunderstanding of collaborative student-student learning, noting the tension between collaboration and ensuring development of individual skillsets. Sue expressed that group work did not ensure that every student had practiced every skill and Joan echoed this by explaining that group work facilitated one person doing all the work. Yadin and Or-Bach (2010) echoed this idea of the value of individual learning by asserting that the, "current emphasis on the benefits of collaborative learning belittles the importance of individual learning processes and reduces the opportunities to require and assess individual learning within IS [Information Science]." Yet more commonly, research emphasizes the value of collaboration as resulting in individual mastery of the material, better problem solutions and increased individual problem-solving performance as students request explanations and justifications from one another to share their conceptual and procedural knowledge in jointly constructing problem solutions (Heller, 1992; Webb, 1989). Regardless, collaborative learning and problem solving is an essential component of the workplace. In this sense, faculty's informal, unstructured and absent enactment of collaborative student learning along with the emphasis on individual student learning in this

study may have limited overall student learning.

Furthermore, limiting student collaboration prevents the interactive processes of negotiation, discussion, and information sharing that promotes learning and results in knowledge construction in a social-cultural context (Wang et al., 2009). Engaging in these interactive processes of group dialogue through discussion, negotiation, questioning, information sharing or debate can serve as an opportunity for problem solvers to iteratively refine their own problem representations in approaching a problem solution (Jonassen, 1997). Social discourse through peer collaboration presents opportunities for students to work through all steps of the problem-solving process; framing a problem, considering alternative aspects of a problem, identifying possible alternative solutions, and assessing the viability of the solutions, as well as monitoring and adapting solutions and presenting insights and solutions from multiple perspectives that might not otherwise be apparent. The end result of social constructivist problem solving enables students as members of groups to move beyond being more than a convenient way to accumulate the knowledge of the group members. Instead they work together toward synergistically accruing insights and solutions that might otherwise not emerge (Brown et al., 1989).

Moreover, if students simply learn collaboration from peers without purposefully structured and guided instruction on how to effectively collaborate, educators may exacerbate the risk that students will learn ineffective methods of collaboration, thereby potentially limiting their success in the workforce. Indeed, faculty in the current study explicitly noted that students were put into groups with no intentional instruction on how to effectively contribute as a team member. This could have long-lasting implications on the efforts to diversify the field of computing to become more inclusive of a broad range of styles inherent in prospective future computing professionals.

The value to students in successfully engaging in teamwork as they enter the workforce not only lies in their success in their profession, but in their ability to effectively learn in order to solve problems in the domain. By omitting student collaborations entirely or limiting them to informal and unstructured experiences, faculty missed opportunities for students to engage more deeply in problem solving as unguided by social-constructivist theories, potentially limiting student opportunities to prepare for effective collaboration in the workforce as well as more effectively solve the computing problems presented to them.

A Preliminary Collaborative Problem-Solving Framework for Computing Education Research

Altogether, this study suggests that computing faculty would benefit from professional development that not only addresses problem types as per Jonassen's (2000) problem-solving typology, including the approaches emphasized in the study, but also the pedagogical approaches and strategies that complement these types of problems. In this regard, this study expands Jonassen's (1997; 2000) problem-solving typology and work on instructional problem-solving approaches to include collaborative approaches alongside structured and explicit strategies. Figure 36 illustrates the multifaceted nature of this endeavor and highlights the various dimensions at play in computing education as a preliminary collaborative problem-solving framework for computing education research.

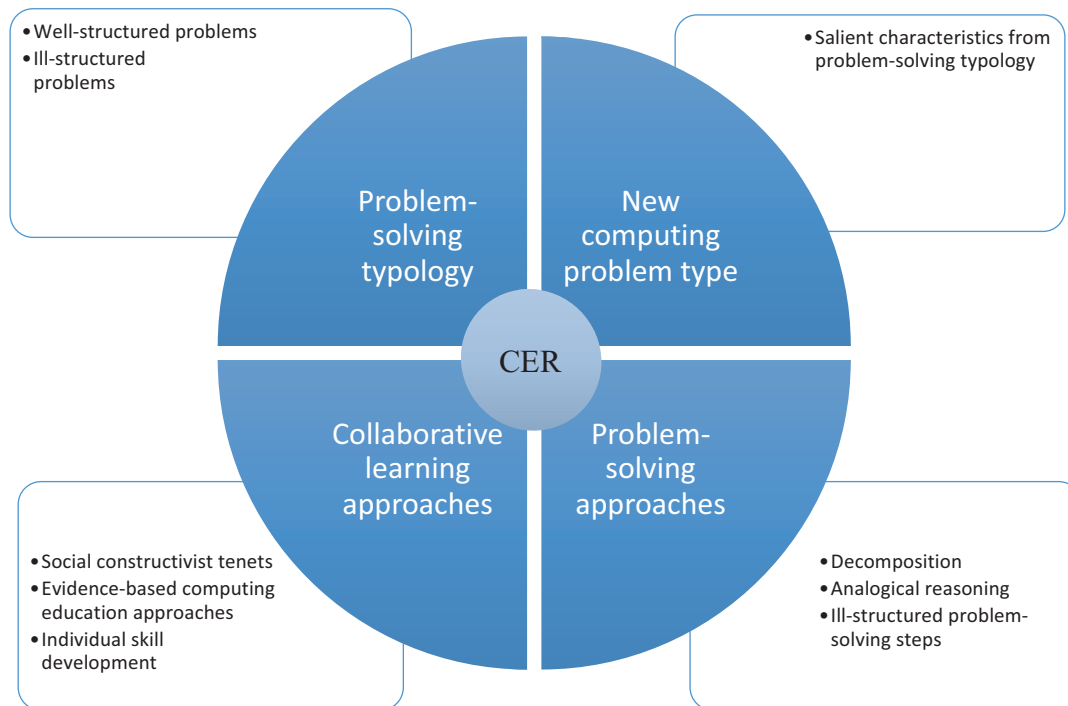


Figure 35. Preliminary collaborative problem-solving computing education research framework.

Implications for Practice

The findings of this study have implications for future instructional practices in terms of collaborative learning in the context of problem solving in computing courses. These implications center on the notion that more attention is required for computing faculty in evidence-based pedagogical practices. Specifically, opportunities exist for faculty to engage in more explicit and structured instructional experiences for students in posing problems, presenting problem-solving approaches and fostering a collaborative learning environment. Being intentional about these practices may prove beneficial to addressing areas that inadvertently go unnoticed by faculty in their daily instructional and classroom practices. This is not to suggest that simply drawing attention to these areas will enable faculty to adopt instructional practices that integrate evidence-based approaches into their instruction. In fact, Barker, Hovey & Gruning (2015) asserted that despite the wealth of evidence-based effective teaching practices, many of these practices remain unknown to faculty until they intentionally seek them out to solve a

problem, or they are presented through funded initiatives, conferences and journals, or from colleagues. As such, this set of implications for practice serves as a first step in making our faculty colleagues aware of these instructional approaches (Barker, Hovey, & Gruning, 2015).

Types of problems posed. In the current study, faculty primarily posed and emphasized well-structured problems to first-year and second-year computing students. Faculty also highlighted the importance of real-world problems and allowing for multiple solution paths with better or worse solutions. In considering the implications for practice, this study elucidated the salient aspects of problems posed by computing faculty. As such, an opportunity exists for faculty to explicitly recognize the characteristics of the problems they pose to ensure that the relevant aspects of the domain content are being presented. Given the limited research of problem solving in computing education, questions arise about the nature of problem solving and pedagogical training for computing education. The findings of this study suggest that faculty would benefit from professional development that explicitly highlights the characteristics of various problems and how their instruction may include or exclude important features along the continuum of problem types. Understanding each problem type and its associated characteristics could inform how faculty pose problems to students and enact their approaches to problem solving in a classroom environment. Even more importantly, using Jonassen's (2000) problem-solving typology, faculty can visualize where their instructional emphasis lies, all at the same time identifying potential means to expand their instructional approaches.

While the emphasis on well-structured problems may be warranted for first-year and second-year computing students because of their limited problem-solving skills and domain content background, there is also value in introducing ill-structured problems during instruction. Ill-structured problems provide opportunities for students to apply their skills and knowledge and

engage in real-world problems that are less constrained by classroom limitations. An opportunity exists here to develop faculty pedagogical understanding of the continuum from well-structured to ill-structured problems and to address scaffolding strategies to engage in full spectrum of problem characteristics across the continuum.

Problem-solving approaches. In this study, faculty implicitly focused on problem decomposition as a means to solving problems. This approach serves as one specific strategy in the second step of the three-step problem-solving process: (1) defining and representing the problem, (2) exploring possible solution strategies and (3) implementing strategies and reflecting back to evaluate the effects of the solutions (Gick, 1986; Jonassen, 1997). By focusing course instruction on implicitly modeling problem decomposition from step 2, the other problem-solving steps of defining and representing the problem and reflecting back to evaluate the effects of the solutions were left unattended. It is in this first problem-solving step that the goal is extracted from the problem statement with questions such as “what do I need to produce” and “what does an appropriate solution look like” are asked. Here, problem solvers create a problem space with their understanding of the problem that includes the attributes, goals and possible solutions (Gick, 1986; Greeno, 1977). In the third problem-solving step, problem solvers test solutions, adjust processes and generate new hypotheses (Jonassen, 1997).

Additionally, alternative problem-solving strategies from step 2, such as the recall analogical reasoning heuristic, were left unexplored. The recall analogical reasoning heuristic uses similarities between the current problem state and previous problem states along with recollection of the solution. This has been successfully enacted by computing faculty using Pattern Oriented Instruction (POI) in introductory programming with students demonstrating better problem-solving competence, more competent problem decomposition and solution

construction (Muller, Ginat, & Haberman, 2007). Regardless of the problem-solving approach to be used, delineating general strategies into more discrete and explicit steps (Dhillon, 1998; Peterson & Treagust, 1998; Schraw et al., 2006), and specifically outlining a general process of analysis, design, exploration, implementation and verification through structured instruction (Schoenfeld, 1980) have been demonstrated to improve learning.

Thus, the current study's implicit focus on one specific problem-solving strategy by faculty suggests that faculty may not be engaging in the full complement of previously outlined effective problem-solving approaches. Furthermore, lack of structured or explicit approaches to problem-solving learning may limit opportunities for deeper student learning. As such, guiding computing faculty in both a broader and deeper understanding of explicit and effective problem-solving instructional approaches may prove beneficial to enhancing student problem solving in the computing domain.

Collaborative learning. In the current study, faculty focused on building individual student skillsets while fostering a limited informal and unstructured student-student collaborative learning environment. As such, an opportunity exists for faculty to augment their understandings of the evidence in computing education research that supports the notion of increased student learning through structured collaborations. For example, Sabin (2008) specifically noted that an effective collaborative learning model requires “teaching students effective team work practices.” Using a structured approach, Ge and Land (2003) emphasized the importance of guiding and monitoring peer interactions in a formalized and structured manner, as with Vygotsky's (1930-1934/1978) ZPD. Faculty in the current study missed opportunities to realize the benefits of formal and structured collaborations which include students submitting more working programs with increased correct and required features (Hanks, et al., 2010), increased

likelihood of female student persistence (Lewis et al., 2012), students attempting and debugging more problems (Murphy, et al., 2010) and improved student passing rates (Vihavainen et al., 2014).

Faculty in the current study also neglected to integrate effective and more structured collaborative learning approaches utilizing social constructivist scaffolding techniques such as question prompts, compelling students to articulate the steps they have taken and their rationale for actions (Lin, et al., 1999) or reciprocal teaching to engage in summarizing, questioning, clarifying, interpreting and predicting in order to enhance learner comprehension and monitor their own understanding (Palincsar & Brown, 1984). As evidenced, faculty in the current study did not appear to recognize the critical impact of peer interactions on learning. Augmenting classroom practices with the aforementioned approaches presents opportunities for students to engage in collaborative learning as part of the problem-solving process whereby they can articulate problem-solving steps and present their rationale for actions along with summarizing, questioning, clarifying, interpreting and predicting problem-solving facets.

In fact, misguided and incorrect understandings of collaborative learning by faculty in this study may limit the potential of community learning which is critical to computing in the workplace. Without these experiences, students are positioned at a significant disadvantage when compared with programs that promote more structured approaches to problem solving and collaborative learning. By formally integrating structured and evidence-based collaborations in classroom enactments, faculty may in fact realize the benefits of collaborative student learning.

Study Limitations

This study was limited in that it centered on a single large private technical university in the northeast. Future investigations may include additional study sites such as smaller, liberal

arts colleges with computing programs as well as larger public universities to provide a broader and more robust dataset. While this study did examine computing education across three distinct computing programs, additional study of computing subdomains may also prove important in examining specific problems as well as approaches to problem solving and collaborative learning.

According to Spiro, et al. (1988), introductory learning in a domain differs from advanced knowledge acquisition, also known as expertise, in many ways. This study was limited in examining computing curriculum and instructional enactment in first-year and second-year computing courses. As such, future studies may examine third and fourth-year computing programs as they further prepare students for entry to the workforce after establishing a baseline of computing knowledge and expertise. This breakdown of early curriculum and more advanced curriculum may also present an opportunity for comparing problems posed to students, problem-solving approaches and collaborative learning.

Recommendations for Future Research

In terms of research, this exploratory case study informed several new directions for future work. Specifically, there are numerous research gaps related to educational computing theory around problem solving and collaborative learning in the domain. First, continued research on problems posed to students is warranted. Jonassen (2000) contended a gap exists between the problems presented to learners in a school environment and the problems encountered in a professional context. Given the workforce demands previously noted, the question arises as to whether or not the problem types emerging as part of this study truly are reflective of the workforce environment or whether more ill-structured problems or problem aspects are warranted, and at what level of the curriculum. For example, it is conceivable that

first-year and second-year curriculum should focus on well-structured problems with third-year and fourth-year curriculum focusing on problems more ill-structured in nature. Thus, the next steps here warrant a deeper understanding of the problem types required in the computing domain such that graduates become successful contributors to the workforce in the context of problem solving.

Second, additional research regarding student understanding of problem solving is necessary. Faculty valued the role of problem solving and enacted the importance of implicitly modeling practices in order to guide students to acquire skills to engage in problem solving. Indeed, all of the faculty emphasized problem decomposition and its role in enabling students to define their processes. Despite these benefits, implicit modeling of the process may have resulted in an abstract process. The next steps here would thus involve understanding of how students understand and enact these practices in their own approaches to problem solving. This understanding has reciprocal implications for practice, as Barker, et al. (2015) indicated that student feedback on instructional practices was the single and most critical kind of evidence in supporting faculty decisions to routinize practices in their classrooms.

Finally, future research into the impact of specific instructional approaches around collaborative learning in the context of problem solving in the computing domain and assessment of these approaches is needed. In particular, examining informal and unstructured approaches would be important when considering underrepresented subpopulations in computing. As students are largely left to find their own way when engaging with collaborative practices, the risk ensues that students will learn ineffective methods of collaboration, thereby potentially limiting efforts to diversify the computing field. As such, the next steps here include better understanding the impact that guided and structured collaborative learning may have on diverse

problem solving in the computing domain. In summary, the findings from this study in consideration with potential future work could inform development of an inclusive framework for computing education grounded in the context of problem solving and social constructivism.

Future research questions motivated by this study include:

1. How do computing students approach problem solving?
2. In what ways do computing students experience collaborative learning enacted in the context of problem solving?
3. In the context of problem solving, how does the learning environment in computing education promote a diverse and inclusive collaborative culture?
4. Is a new computing-specific problem type with distinct instructional problem-solving approaches warranted?
5. What structured approaches for collaborative problem solving in the computing domain are most effective for various levels of computing curriculum?

References

- ACM. (2017). Association for Computing Machinery. Retrieved from <http://www.acm.org/>
- Adams, P. (2006). Exploring social constructivism: Theories and practicalities. *Education*, 34(3), 243-257.
- Anderson, J. R. (2005). *Cognitive psychology and its implications*: Macmillan.
- Apple. (2017). Apple Education. Retrieved from <https://www.apple.com/education/teaching-code/>
- Archer, W., & Davison, J. (2008). Graduate Employability: what do employers think and want? Council for Industry and Higher Education. In.
- Ardis, M., Budgen, D., Hislop, G., Offutt, J., Sebern, M., & Visser, W. (2014). Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. *joint effort of the ACM and the IEEE-Computer Society*.
- Atkinson, R. K., & Derry, S. J. (2000). Computer-Based Examples Designed to Encourage Optimal Example Processing: A Study Examining the Impact of Sequentially Presented, Subgoal-Oriented Worked Examples1.
- Bailey, M., Mason, S., Marchetti, C., Dell, E., Clayton, L., Rommel, A., & Valentine, M. (2015). *Institutional Transformation Guided by a Multi-Frame Organizational Analysis Approach*. Paper presented at the 2015 American Society for Engineering Education Annual Conference and Exposition, Seattle, WA.
- Baker, D. P., Day, R., & Salas, E. (2006). Teamwork as an essential component of high-reliability organizations. *Health services research*, 41(4p2), 1576-1598.
- Balliet, R. N., Riggs, E. M., & Maltese, A. V. (2015). Students' problem solving approaches for developing geologic models in the field. *Journal of Research in Science Teaching*, 52(8), 1109-1131.
- Barker, L., Hovey, C. L., & Gruning, J. (2015). *What influences CS faculty to adopt teaching practices?* Paper presented at the Proceedings of the 46th ACM Technical Symposium on Computer Science Education.

- Barr, R. B., & Tagg, J. (1995). From teaching to learning—A new paradigm for undergraduate education. *Change: The magazine of higher learning*, 27(6), 12-26.
- Begel, A., & Simon, B. (2008a). *Novice software developers, all over again*. Paper presented at the Proceedings of the Fourth international Workshop on Computing Education Research.
- Begel, A., & Simon, B. (2008b). *Struggles of new college graduates in their first software development job*. Paper presented at the ACM SIGCSE Bulletin.
- Bereiter, C. (2014). Principled practical knowledge: Not a bridge but a ladder. *Journal of the Learning Sciences*, 23(1), 4-17.
- Bhattacharyya, G., & Bodner, G. M. (2014). Culturing reality: how organic chemistry graduate students develop into practitioners. *Journal of Research in Science Teaching*, 51(6), 694-713.
- Bransford, J. (1993). Who ya gonna call? Thoughts about teaching problem solving. *Cognitive perspectives on educational leadership*, 171-191.
- Bransford, J., & Stein, B. L. (1984). *The IDEAL problem solver: A guide for improving thinking, learning, and creativity*. NY: WH Freeman and Company.
- Bransford, J., Sherwood, R., Vye, N., & Rieser, J. (1986). Teaching thinking and problem solving: Research foundations. *American psychologist*, 41(10), 1078.
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2), 77-101.
- Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational researcher*, 18(1), 32-42.
- Cannon-Bowers, J. A., Tannenbaum, S. I., Salas, E., & Volpe, C. E. (1995). Defining competencies and establishing team training requirements. *Team effectiveness and decision making in organizations*, 333, 380.
- Catrambone, R. (1996). Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 22(4), 1020.

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General*, 127(4), 355.

Chi, Feltovich, & Glaser. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive science*, 5(2), 121-152.

Chi, & Glaser. (1985). Problem solving ability. In R. J. Sternberg (Ed.), *Human abilities: An information processing approach*. New York: W.H. Freeman.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive science*, 5(2), 121-152.

Chi, M. T., Glaser, R., & Rees, E. (1981). *Expertise in problem solving*. Retrieved from

Chidanandan, A., Russell-Dag, L., Laxer, C., & Ayfer, R. (2010). *In their words: student feedback on an international project collaboration*. Paper presented at the Proceedings of the 41st ACM technical symposium on Computer science education.

CollegeBoard. (2017). AP Computer Science Principles. Retrieved from <https://apstudent.collegeboard.org/apcourse/ap-computer-science-principles>

Cooperrider, D. L., Barrett, F., & Srivastva, S. (1995). Social construction and appreciative inquiry: A journey in organizational theory. *Management and organization: Relational alternatives to individualism*, 157-200.

Crabtree, B. F., & Miller, W. F. (1992). A template approach to text analysis: developing and using codebooks.

Creswell, J. W. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches*: Sage publications.

Crotty, T. (1994). *Integrating distance learning activities to enhance teacher education toward the constructivist paradigm of teaching and learning*. Paper presented at the Distance learning research conference proceedings.

CSTA. (2017). Computer Science Teachers Association. Retrieved from <http://www.csteachers.org/>

- Daly, J., Kellehear, A., & Gliksman, M. (1997). The public health researcher: A methodological approach. In: Melbourne, Australia: Oxford University Press.
- Dhillon, A. S. (1998). Individual differences within problem-solving strategies used in physics. *Science Education*, 82(3), 379-405.
- Driscoll, M. P. (2000). Psychology of learning. *Boston, Allyn and Bacon*.
- Elio, R., & Scharf, P. B. (1990). Modeling novice-to-expert shifts in problem-solving strategy and knowledge organization. *Cognitive Science*, 14(4), 579-639.
- Elstein, A. S., Shulman, L. S., & Sprafka, S. A. (1978). Medical problem solving an analysis of clinical reasoning.
- Fayer, S., Lacey, A., & Watson, A. (2017). *STEM Occupations: Past, Present, And Future*. Retrieved from <https://www.bls.gov/spotlight/2017/science-technology-engineering-and-mathematics-stem-occupations-past-present-and-future/pdf/science-technology-engineering-and-mathematics-stem-occupations-past-present-and-future.pdf>
- Fereday, J., & Muir-Cochrane, E. (2006). Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International journal of qualitative methods*, 5(1), 80-92.
- Fincher, S., & Petre, M. (2004). *Computer science education research*: CRC Press.
- Fogler, H. S., LeBlanc, S. E., & Rizzo, B. R. (1995). *Strategies for creative problem solving*: PTR Prentice Hall Englewood Cliffs, NJ.
- Fosnot, C. T., & Perry, R. S. (1996). Constructivism: A psychological theory of learning. *Constructivism: Theory, perspectives, and practice*, 2, 8-33.
- Gergen, K. J. (1985). The social constructionist movement in modern psychology. *American psychologist*, 40(3), 266.
- Gick, M. L. (1986). Problem-solving strategies. *Educational psychologist*, 21(1-2), 99-120.
- Given, L. M. (2008). *The Sage encyclopedia of qualitative research methods*: Sage publications.

- Glesne, C., & Peshkin, A. (1992). *Becoming qualitative researchers: An introduction*. Longman White Plains, NY.
- Google. (2017). Google Computer Science Education. Retrieved from <https://edu.google.com/cs/index.html>
- Greeno. (1977). Process of understanding in problem solving. In N. J. Castellan, D. B. Pisoni, & G. R. Potts (Eds.), *Cognitive theory* (Vol. 2, pp. 43-83). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Greeno. (1978). *Natures of problem-solving abilities*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Greeno, J. (1978). *Natures of problem-solving abilities*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Greeno, J. G. (2006). Learning in Activity. In R. K. Saywer (Ed.), *The Cambridge handbook of the learning sciences* (pp. 79-96). New York, NY: Cambridge University Press.
- Gupta, G. K. (2007). Computer science curriculum developments in the 1960s. *IEEE Annals of the History of Computing*, 29(2).
- Hanks, B. (2008). Problems encountered by novice pair programmers. *Journal on Educational Resources in Computing (JERIC)*, 7(4), 2.
- Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). *Program quality with pair programming in CSI*. Paper presented at the ACM SIGCSE Bulletin.
- Hazzan, O. (2003). Computer science students' conception of the relationship between reward (grade) and cooperation. *ACM SIGCSE Bulletin*, 35(3), 178-182.
- Helminen, J., Ihantola, P., Karavirta, V., & Malmi, L. (2012). *How do students solve parsons programming problems?: an analysis of interaction traces*. Paper presented at the Proceedings of the ninth annual international conference on International computing education research.
- Heppner, P. P., & Krauskopf, C. J. (1987). An information-processing approach to personal problem solving. *The Counseling Psychologist*, 15(3), 371-447.

- Heyworth, R. M. (1999). Procedural and conceptual knowledge of expert and novice students for the solving of a basic problem in chemistry. *International Journal of Science Education*, 21(2), 195-211.
- Hogan, K. (1999). Sociocognitive roles in science group discourse. *International Journal of Science Education*, 21(8), 855-882.
- Hogan, K. (2002). Small groups' ecological reasoning while making an environmental management decision. *Journal of Research in Science Teaching*, 39(4), 341-368.
- ICER. ACM International Computing Education Research (ICER). Retrieved from <https://icer.acm.org/>
- IEEE. (2017). Institute of Electrical and Electronics Engineers. Retrieved from https://www.ieee.org/index.html?WT.mc_id=head_bm
- ISTE. (2017). International Society for Technology in Education. Retrieved from <https://www.iste.org/>
- ITiCSE. ACM Innovation and Technology in Computer Science Education conference (ITiCSE). Retrieved from <http://sigcse.org/sigcse/events/iticse>
- Jonassen. (1997). Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational technology research and development*, 45(1), 65-94.
- Jonassen. (2000). Toward a design theory of problem solving. *Educational technology research and development*, 48(4), 63-85.
- Jonassen, D., Davidson, M., Collins, M., Campbell, J., & Haag, B. B. (1995). Constructivism and computer-mediated communication in distance education. *American journal of distance education*, 9(2), 7-26.
- Jonassen, D., Strobel, J., & Lee, C. B. (2006). Everyday problem solving in engineering: Lessons for engineering educators. *Journal of engineering education*, 95(2), 139-151.

- Joy, M., Sinclair, J., Sun, S., Sitthiworachart, J., & López-González, J. (2009). Categorising computer science education research. *Education and Information Technologies*, 14(2), 105-126.
- Kennedy, D., Hyland, A., & Ryan, N. (2009). Learning outcomes and competences. *Introducing Bologna Objectives and Tools*, 2.3-3.
- Kim, J. (1981). Processes of Asian American identity development: A study of Japanese American women's perceptions of their struggle to achieve positive identities as Americans of Asian ancestry.
- Kinnunen, P., Meisalo, V., & Malmi, L. (2010). *Have we missed something?: identifying missing types of research in computing education*. Paper presented at the Proceedings of the Sixth international workshop on Computing education research.
- Kitchner, K. S. (1983). Cognition, metacognition, and epistemic cognition. *Human development*, 26(4), 222-232.
- Kozulin, A., Gindis, B., Agseyev, V. & Miller, S. (Eds.). (2003). *Vygotsky Educational Theory in Cultural Context*. Cambridge: Cambridge University Press.
- Largent, D. L., & Lüer, C. (2010). *You mean we have to work together!?!: a study of the formation and interaction of programming teams in a college course setting*. Paper presented at the Proceedings of the Sixth international workshop on Computing education research.
- Larkin, McDermott, Simon, & Simon. (1980). Expert and novice performance in solving physics problems. *Science*, 208(4450), 1335-1342.
- Larkin, & Reif. (1979). Understanding and teaching problem-solving in physics. *European Journal of Science Education*, 1(2), 191-203.
- Larkin, J., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, 208(4450), 1335-1342.

- Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*: Cambridge university press.
- Lewis, C. M., Titterton, N., & Clancy, M. (2012). *Using collaboration to overcome disparities in Java experience*. Paper presented at the Proceedings of the ninth annual international conference on International computing education research.
- Lin, X., Hmelo, C., Kinzer, C. K., & Secules, T. J. (1999). Designing technology to support reflection. *Educational Technology Research and Development*, 47(3), 43-62.
- Lin, X., & Lehman, J. D. (1999). Supporting learning of variable control in a computer-based biology environment: Effects of prompting college students to reflect on their own thinking. *Journal of research in science teaching*, 36(7), 837-858.
- Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry* (Vol. 75): Sage.
- Lingard, R., & Barkataki, S. (2011). *Teaching teamwork in engineering and computer science*. Paper presented at the Frontiers in Education Conference (FIE), 2011.
- Loftus, C., Thomas, L., & Zander, C. (2011). *Can graduating students design: revisited*. Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer science education.
- Luft, J. A., & Pizzini, E. L. (1998). The demonstration classroom in-service: Changes in the classroom. *Science Education*, 82(2), 147-162.
- Malmi, L. (2014). Theory---what is it for? *ACM Inroads*, 5(4), 34-35.
- Malmi, L., Sheard, J., Bednarik, R., Helminen, J., Kinnunen, P., Korhonen, A., . . . Taherkhani, A. (2014). *Theoretical underpinnings of computing education research: what is the evidence?* Paper presented at the Proceedings of the tenth annual conference on International computing education research.
- Martella, R. C., Nelson, J. R., Morgan, R. L., & Marchand-Martella, N. E. (2013). *Understanding and interpreting educational research*: Guilford Press.
- Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). *Subgoals, context, and worked examples in learning computing problem solving*. Paper presented at the Proceedings of

the eleventh annual international conference on international computing education research.

Muller, O. (2005). *Pattern oriented instruction and the enhancement of analogical reasoning*. Paper presented at the Proceedings of the first international workshop on Computing education research.

Murphy, L., Fitzgerald, S., Hanks, B., & McCauley, R. (2010). *Pair debugging: a transactive discourse analysis*. Paper presented at the Proceedings of the Sixth international workshop on Computing education research.

NACE. (2016). *JOB OUTLOOK 2016: THE ATTRIBUTES EMPLOYERS WANT TO SEE ON NEW COLLEGE GRADUATES' RESUMES*. Retrieved from <http://www.naceweb.org/career-development/trends-and-predictions/job-outlook-2016-attributes-employers-want-to-see-on-new-college-graduates-resumes/>

Newell, A., Shaw, J. C., & Simon, H. A. (1958). Elements of a theory of human problem solving. *Psychological review*, 65(3), 151.

Niss, M. (2007). The concept and role of theory in mathematics education. *Relating practice and research in mathematics education. Proceedings of Norma*, 5, 97-110.

Oracle. (2017). Oracle Academy. Retrieved from <https://academy.oracle.com/en/solutions-summary.html>

Ormrod, J. E. (1995). *Educational psychology: Principles and applications*: Merrill.

Pais, A., Stentoft, D., & Valero, P. (2010). *From questions of how to questions of why in mathematics education research*. Paper presented at the the Sixth International Mathematics Education and Society Conference.

Palincsar. (1998). Social constructivist perspectives on teaching and learning. *Annual review of psychology*, 49(1), 345-375.

Palincsar, & Brown. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and instruction*, 1(2), 117-175.

Pathak, S. (2017). Artificial intelligence 38 Backward Reasoning in ai, goal driven reasoning.

- Pears, A., Seidman, S., Eney, C., Kinnunen, P., & Malmi, L. (2005). Constructing a core literature for computing education research. *ACM SIGCSE Bulletin*, 37(4), 152-161.
- Peshkin, A. (1982). The researcher and subjectivity: Reflections on an ethnography of school and community. *Doing the ethnography of schooling*, 48-67.
- Peshkin, A. (1985). Virtuous subjectivity: In the participant-observer's I's. *Exploring clinical methods for social research*, 267, 281.
- Peshkin, A. (1988). In search of subjectivity—one's own. *Educational researcher*, 17(7), 17-21.
- Peterson, R. F., & Treagust, D. F. (1998). Learning to teach primary science through problem-based learning. *Science Education*, 82(2), 215-237.
- Pieterse, V., Thompson, L., Marshall, L., & Venter, D. M. (2012). *Participation patterns in student teams*. Paper presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education.
- Pollock, L., & Harvey, T. (2011). *Combining multiple pedagogies to boost learning and enthusiasm*. Paper presented at the Proceedings of the 16th annual joint conference on Innovation and technology in computer science education.
- Polson, P., & Jeffries, R. (2014). Instruction in general problem-solving skills: An analysis of four approaches. In *Thinking and Learning Skills: Volume 1: Relating Instruction To Research*, 417.
- Potosky, D. (2014). The Instructor's Toolbox: A Meaning-Centered Framework for the Social Construction of Experiential Learning. *Developments in Business Simulation and Experiential Learning*, 33.
- Powell, K. C., & Kalina, C. J. (2009). Cognitive and social constructivism: Developing tools for an effective classroom. *Education*, 130(2), 241-251.
- Radermacher, A., & Walia, G. (2013). *Gaps between industry expectations and the abilities of graduates*. Paper presented at the Proceeding of the 44th ACM technical symposium on Computer science education.

- Randolph, J. J., Julnes, G., Bednarik, R., & Sutinen, E. (2007). A comparison of the methodological quality of articles in computer science education journals and conference proceedings. *Computer Science Education, 17*(4), 263-274.
- Rice, J., & Rosen, S. (1994). History of Computer Science from Purdue University. Retrieved from <https://www.cs.purdue.edu/history/history.html>
- Robles, M. M. (2012). Executive perceptions of the top 10 soft skills needed in today's workplace. *Business Communication Quarterly, 75*(4), 453-465.
- Romm, T. (2017). Amazon, Facebook and others in tech will commit \$300 million to the White House's new computer science push. *Recode*.
- Rosenshine, B., & Meister, C. (1992). The use of scaffolds for teaching higher-level cognitive strategies. *Educational leadership, 49*(7), 26-33.
- Sabin, M., Alrumaih, H., Impagliazzo, J., Lunt, B. M., Tang, C., Zhang, M., . . . Viola, B. (2017). *ACM/IEEE-CS Information Technology Curriculum 2017: Final Draft*. Retrieved from <http://www.acm.org/binaries/content/assets/education/it2017.pdf>
- Sahami, M., Danyluk, A., Fincher, S., Fisher, K., Grossman, D., Hawthorne, E., . . . Thompson, A. (2013). *Computer Science Curricula 2013*. Retrieved from New York, NY: <https://www.acm.org/education/CS2013-final-report.pdf>
- Salas, E., Dickinson, T. L., Converse, S. A., & Tannenbaum, S. I. (1992). *Toward an understanding of team performance and training*: Ablex Publishing.
- Saldaña, J. (2015). *The coding manual for qualitative researchers*: Sage.
- Schoenfeld, A. H. (1980). Teaching problem-solving skills. *The American Mathematical Monthly, 87*(10), 794-805.
- Schoenfeld, A. H. (1983). Beyond the purely cognitive: Belief systems, social cognitions, and metacognitions as driving forces in intellectual performance. *Cognitive science, 7*(4), 329-363.

- Schraw, G., Crippen, K. J., & Hartley, K. (2006). Promoting self-regulation in science education: Metacognition as part of a broader perspective on learning. *Research in science education*, 36(1-2), 111-139.
- Schraw, G., Dunkle, M. E., & Bendixen, L. D. (1995). Cognitive processes in well-defined and ill-defined problem solving. *Applied Cognitive Psychology*, 9(6), 523-538.
- Schön, D. A. (1990). The Design Process. In V. A. Howard (Ed.), *Varieties of thinking: Essays from Harvard's philosophy of education center* (pp. 110-141). New York: Routledge.
- Seidman, I. (2012). *Interviewing as qualitative research: A guide for researchers in education and the social sciences*: Teachers college press.
- Shackelford, R., McGettrick, A., Sloan, R., Topi, H., Davies, G., Kamali, R., . . . Lunt, B. (2006). *Computing curricula 2005: The overview report*. Paper presented at the ACM SIGCSE Bulletin.
- Shulman, L. (1987). Knowledge and teaching: Foundations of the new reform. *Harvard educational review*, 57(1), 1-23.
- SIGCSE. ACM Special Interest Group on Computer Science Education (SIGCSE). Retrieved from <https://dl.acm.org/sig.cfm?id=SP927>
- SIGITE. ACM Special Interest Group for Information Technology Education (SIGITE). Retrieved from <http://www.sigite.org/>
- Silver, E. A., & Marshall, S. P. (1990). Mathematical and scientific problem solving: Findings, issues, and instructional implications. *Dimensions of thinking and cognitive instruction*, 1, 265-290.
- Simon, & Simon. (1978). Individual differences in solving physics problems.
- Simon, D. P., & Simon, H. A. (1978). Individual differences in solving physics problems.
- Simon, H. A. (1973). The structure of ill structured problems. *Artificial intelligence*, 4(3-4), 181-201.

- Sims, D., Salas, E., & Burke, C. (2004). Is There a 'Big Five' in Teamwork? 19th Annual Meeting of the Society for Industrial and Organizational Psychology. 4. *Chicago, IL*.
- Spiro, R. J., Coulson, R., Feltovich, P., & Anderson, D. (1988). Cognitive Flexibility Theory: Advanced Knowledge Acquisition in Ill-Structured Domains. Technical Report No. 441.
- Spiro, R. J., Vispoel, W. P., Schmitz, J. G., Samarapungavan, A., Boerger, A., & Britton, B. (1987). Cognitive flexibility and transfer in complex content domains. *Executive control processes in reading*, 177-199.
- Spradley, J. P. (1980). *Participant observation*. New York Holt, Rinehart and Winston.
- Starks, H., & Brown Trinidad, S. (2007). Choose your method: A comparison of phenomenology, discourse analysis, and grounded theory. *Qualitative health research*, 17(10), 1372-1380.
- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2), 257-285.
- Tanenbaum, A. S., & Wetherall, D. (2014). *Computer networks*: Harlow, Essex: Pearson.
- Taylor, K. L., & Dionne, J.-P. (2000). Accessing problem-solving strategy knowledge: The complementary use of concurrent verbal protocols and retrospective debriefing. *Journal of Educational Psychology*, 92(3), 413.
- Tekumru Kisa, M., & Stein, M. K. (2015). Learning to see teaching in new ways: A foundation for maintaining cognitive demand. *American Educational Research Journal*, 52(1), 105-136.
- TOCE. ACM Transactions of Computing Education (TOCE). Retrieved from <https://toce.acm.org/>
- Tracy, S. J. (2010). Qualitative quality: Eight "big-tent" criteria for excellent qualitative research. *Qualitative inquiry*, 16(10), 837-851.
- Vihavainen, A., Airaksinen, J., & Watson, C. (2014). *A systematic review of approaches for teaching introductory programming and their influence on success*. Paper presented at

the Proceedings of the tenth annual conference on International computing education research.

Vivian, R., Falkner, K., Falkner, N., & Tarmazdi, H. (2016). A method to analyze computer science students' teamwork in online collaborative learning environments. *ACM Transactions on Computing Education (TOCE)*, 16(2), 7.

Voss, J. F., Greene, T. R., Post, T. A., & Penner, B. C. (1983). Problem-solving skill in the social sciences. In *Psychology of learning and motivation* (Vol. 17, pp. 165-213): Elsevier.

Voss, J. F., & Post, T. A. (1988). On the solving of ill-structured problems.

Vygotsky. (1930-1934/1978). *Mind and Society: The development of higher mental process* (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman Eds.). Cambridge, MA: Harvard University Press.

Vygotsky, L. S. (1962). Language and thought. *Massachusetts Institute of Technology Press, Ontario, Canada*.

Wang, Q., Teo, T., & Woo, H. L. (2009). An integrated framework for designing web-based constructivist learning environments. *International Journal of Instructional Media*, 36(1), 81-92.

Webb, N. M. (1989). Peer interaction and learning in small groups. *International journal of Educational research*, 13(1), 21-39.

Wiggins, G. P., & McTighe, J. (2011). *The understanding by design guide to creating high-quality units*: ASCD.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

Woo, Y., & Reeves, T. C. (2007). Meaningful interaction in web-based learning: A social constructivist interpretation. *The Internet and higher education*, 10(1), 15-25.

Wood, P. K. (1983). Inquiring systems and problem structure: Implications for cognitive development. *Human Development*, 26(5), 249-265.

Yadin, A., & Or-Bach, R. (2010). The Importance of Emphasizing Individual Learning in the "Collaborative Learning Era". *Journal of Information Systems Education*, 21(2), 185.

Yin, R. K. (2013). *Case study research: Design and methods*: Sage publications.

Appendix A - Faculty Interview Protocol

Guiding Faculty Interview Protocol: Problem Solving in Computing Education

Time of Interview:

Date:

Place:

Interviewer: Sharon Mason

Interviewee:

Position of the interviewee:

Faculty Interview 1 Guiding Questions	
Question	Rationale
<p>1. When you think about ps in computing (CS, IST, SE), what comes to mind? (how do you define ps?)</p>	<p>RQ1, RQ2, RQ3 This interview question explores problem solving across computing subdomains, part of the broader naturalistic study and introduced as part of the workforce demands and the lack of problem-solving distinction from the ACM/IEEE curriculum guidelines. It may also reveal information about expert vs novice approaches as part of ill and well-structured problem-solving approaches and social learning in courses.</p>
<p>2. How does this apply to your course?</p> <ul style="list-style-type: none"> a. Can you give an example of a problem in the context of your teaching? b. Why did you choose this problem? c. Is this typical? d. What other types might you pose? 	<p>RQ1, RQ2, RQ3 This question explores the broader issues of the workforce demands, and the lack of problem-solving distinction from the ACM/IEEE curriculum guidelines. It may also reveal information about expert vs novice approaches as well the kinds of problems being posed (ill and well-structured problem characteristics) and social learning in courses.</p>

<p>3. How do you typically pose a problem to students?</p> <p>a. What is presented to students as a problem? (guidelines, steps, expectations, outcomes?)</p> <p>b. Why do it this way?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question explores problem types as well as social learning, but also may reveal details around expert vs. novice approaches for problems.</p>
<p>4. What do you see as your role when you pose a problem-solving scenario or activity to your students?</p> <p>a. Can you give an example?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question explores expert vs. novice approaches for problem types as well as social learning.</p>
<p>5. What are your expectations when students approach a problem?</p> <p>a. Can you give an example?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question primarily explores approaches to ps (expert vs. novice) as well as social learning approaches in the context of problem types.</p>
<p>6. In what ways do your approaches to problem solving prepare students for work in their respective fields?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question explores all aspects of ps – how faculty perceive it, the problem types and the social learning (this question is also asked of students and serves as a triangulating aspect of the study).</p>
<p>7. Do you see ps as an individual process or a collaborative team effort? Why?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question primarily explores RQ3 around social learning, but is also in relation to approaches and problem types.</p>

Faculty Interview 2 Guiding Questions	
Question	Rationale
<p>1. Let's consider a problem that you had students working on in class last week (from classroom observations).</p> <p>a. Can you tell me about this problem? (What is the rationale and goal for this problem?)</p> <p>b. Why did you choose this problem and approach?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question explores problem types and approaches (novice vs. expert) and social learning.</p>
<p>2. What are your observations about the students approaches to solving this problem?</p> <p>a. Can you break down into steps that you observed?</p> <p>b. What do you expect students experienced?</p> <p>c. How do you want students to approach the problem?</p>	<p>RQ1, RQ2, RQ3</p> <p>This question primarily explores student experiences of ps, but inherently explores approaches to solving specific problem types.</p>
<p>3. What might you do differently?</p> <p>a. Why? (in the context of the course?)</p>	<p>RQ1 RQ2, RQ3</p> <p>This question reveals details about the problems that are being posed (types and approaches), what the faculty expect students are experiencing and how students are being prepared for the workforce demands.</p>

Appendix B – Background in Computing Education Curriculum

During the 1960s, universities focused on defining course curriculum and therefore indirectly defining the field itself (Rice & Rosen, 1994). Yet interest in computing from academics began even earlier. The Association for Computing Machinery (ACM) was established in 1947 during a meeting at Columbia University after a series of meetings and symposia earlier in the year at Harvard University, Massachusetts Institute of Technology and the New York Chapter of the American Institute of Electrical Engineers amid a growing interest in digital machinery. Today, the ACM, whose website tagline reads, “We see a world where computing helps solve tomorrow’s problems – where we use our knowledge and skills to advance the profession and make a positive impact” is noted as “the world’s largest educational and scientific society, uniting computing educators, researchers and professionals to inspire dialogue, share resources and address the field's challenges” (ACM, 2017). Another notable computing organization, the Institute for Electrical and Electronics Engineers (IEEE), currently serves as the “largest technical professional organization for the advancement of technology” (IEEE, 2017). The IEEE’s roots date back to 1884 as electricity was becoming a key influence in society and the telegraph was establishing the groundwork for data communications around the globe (IEEE, 2017).

With the advent of the computer science discipline in the 1960s, and amidst much debate over the focus and core aspects of the discipline, the ACM Curriculum Committee on Computer Science produced the first set of preliminary curriculum guidelines in 1965 after three years’ effort (Gupta, 2007). Since then, the ACM has worked with professional and scientific communities to provide computing curriculum recommendations and updates regularly. Upon establishing a joint task force in the 1980s, the ACM and the IEEE took the lead in developing

the flagship general computing curricula (CC) guidelines for undergraduate degree programs in computing. A growing number of computing degree programs in the 1990s, along with the significant breadth of topics covered, led to a need for curriculum guidelines for various computing subdisciplines.

While acknowledging the expectation for continued emerging computing disciplines in the future, the joint ACM/IEEE task force responded with a set of curriculum guidelines focused on five areas of computing as prominent in 2005 and since updated as such: Computer Engineering Curricula 2016 (CE2016), Computer Science Curricula 2013 (CS2013), Information Systems Curricula 2010 (IS2010), Information Technology Curricula Final Draft 2017 (IT2017) and Software Engineering Curricula 2014 (SE2014) (ACM, 2017; Shackelford et al., 2006). The specific computing discipline guidelines provide a valuable resource to prospective computing students and computing educators in developing academic programs as well as providing a blueprint for accrediting agencies, but do not serve as standards and are therefore not mandatory (ACM, 2017). Each was developed with significant input from the breadth of the computing community including academia, industry and government through surveys, online systems, academic conference presentations and discussions and requests for direct review (Ardis et al., 2014; Sabin et al., 2017; Sahami et al., 2013; Shackelford et al., 2006). Also, recognizing the natural confusion that emerges from having a variety of computing programs available, the ACM and IEEE responded with a now arguably outdated 2005 Computing Curricula Overview Report (CC2005) that distinguishes each of the programs from the other as having its own identity and pedagogical traditions while also outlining the commonalities among the programs (Sabin et al., 2017; Sahami et al., 2013; Shackelford et al., 2006).

For purposes of this case study, I focus on the Computer Science (CS), Information Technology (IT) and Software Engineering (SE) domains and define each as following:

- “Information Technology is the study of systemic approaches to select, develop, apply, integrate, and administer secure computing technologies to enable users to accomplish their personal, organizational, and societal goals” (Sabin et al., 2017).
- Computer science spans the range from theory through programming (Shackelford et al., 2006). Varying definitions of CS are determined by a particular scholars’ emphasis, but for the purposes of this work, Hazzan, Lapidot, and Ragonis’ (2015) definition is most articulate, “computer science is a multifaceted field that encompasses scientific and engineering aspects, which are manifested in algorithmic problem-solving processes, for which computational thinking skills (Wing, 2006), and sometimes also artistic and creative thinking, are required” (p. 24).
- Software engineering is “the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software” (Ardis et al., 2014; IEEE, 2017).

Outcomes, bodies of knowledge and problem spaces serve as a focus for distinguishing and defining each of the computing domains within the reports. While these points are critical to understanding the overarching domain-specific content that serves as a foundation for a computing program, of interest to this study is the common thread of *problem solving* as an indicator of metacognition in regards to obtaining and processing information along with the ability to analyze quantitative data as highlighted by employer demands. This is specifically of interest within a *collaborative learning* setting in the form of teamwork, and as such, is examined within the context of the seminal ACM/IEEE curriculum guidelines.

Problem Solving in Computing Curriculum Guidelines

What might commonly be referred to as content areas in a general academic sense, hold specific names within each of the computing domain ACM/IEEE curriculum guidelines. In computer science, the content areas are noted as Knowledge Areas (KAs), within Information Technology they are IT Domains, and within Software Engineering they are Knowledge Areas further divided into more granular *units* and *topics* for clarity. Distinctions among the computing programs focus on how each “is somewhat different from its siblings in the emphasis, goals, and capabilities of its graduates” (Shackelford et al., 2006). This specifically pertains to the Knowledge Areas, Domains, Units and Topics. For example, when considering operating systems, coverage is domain specific and a computer science program may focus on *design and implementation* while an information technology program may focus on *configuration and use* (Shackelford et al., 2006).

The CC2005 Curriculum Overview Report presents the concept of solving problems as a key facet of *all* computing with the “kinds of problems and issues they address” serving as a distinction among the computing domains (Shackelford et al., 2006). Rather than providing an explicit definition for problem solving within each domain, the CC2005 task force attempted to differentiate problems among the computing domains through a series of examples. For instance, the report offers that computer scientists, “develop effective ways to solve computing problems” exemplified by the idea that using their breadth from theory to programming, computer scientists “develop the best possible ways to store information in databases, send data over networks, and display complex images” (Shackelford et al., 2006). The CC2005 report further grounds problem solving within the discipline of computer science by identifying three career paths for computer scientists: (1) Designing and implementing software, (2) Devising new ways to use computers

and (3) Developing effective ways to solve computing problems (Shackelford et al., 2006). Information Technology programs are cited as “programs that prepare students to meet the computer technology needs of business, government, healthcare, schools, and other kinds of organizations,” by solving any computer-related problems that may arise. Software engineering is noted as “the discipline of developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them” (Shackelford et al., 2006).

The specific domain curriculum guidelines present little additional clarity around the idea of problem solving. The CS2013 guidelines repeatedly cite the ideas of understanding the nature of a problem and designing and analyzing a solution or algorithm for a problem. The CS2013 task force also offers that “graduates need to understand how to apply the knowledge they have gained to solve real problems, not just write code and move bits,” noting the significant impact their problem solutions have on people’s lives and the requirement to move beyond a purely technical activity. (Sahami et al., 2013). The IT2017 guidelines indicate the importance of solving problems for the IT professional, particularly the idea of ill-defined problems as inherent in problem-based learning, but then relegate the topic of problem-solving strategies to the software fundamentals subdomain (Sabin et al., 2017). The SE2014 guidelines present that “the professional practice of software engineering encompasses a range of issues and activities, including problem solving, project management, ethical and legal concerns, written and oral communication, teamwork, and remaining current in a rapidly changing discipline” (Ardis et al., 2014). More details include the idea that, “software engineering problem solving should be taught as having multiple dimensions,” and that,

students should learn to think about and solve problems such as analysis, design, and testing that are related directly to solving the clients' problem. They also need to address meta-problems, such as process improvement, the solutions of which will facilitate product-oriented problem solving. Finally, the curriculum should address areas such as ethical problems that are orthogonal to the other categories. Problem solving is best learned through practice and taught through examples.

(Ardis et al., 2014)

While the goal of the CC2005 Overview Report serves to provide clarity among and between the computing programs, at best it presents a now currently outmoded attempt to distinguish the ideas of *problem solving* among the computing disciplines. The more recent CS, IT and SE guidelines, in their attempts to further distinguish the computing domains offer little additional clarity in regards to the differences. The IT2017 guidelines task force echoes this assertion in its own report by offering that, "it is important to realize that there is a long history of overlap, misunderstanding, and sometimes even contention among the disciplines. This intermingling can follow many dimensions that might separate the disciplines" (Sabin et al., 2017). Arguably, the CC2005 Overview Report as well as the specific domain guidelines, in their attempts to differentiate problem solving among the disciplines, inadvertently present a stronger case for considering more deeply the commonalities among them.

Commonalities among all computing programs identified in the CC2005 Overview report include: common underpinnings of computing such as computer programming, understanding the possibilities and limitations of computing technology, understanding technology lifecycle management and the study of advanced and emerging developments in computing (Shackelford, 2005). Of particular interest is the common topic of professional values and principles which

includes, “the identification and acquisition of skill sets that go beyond technical skills. Such skill sets include interpersonal communication skills, team skills, and management skills as appropriate to the discipline. To have value, learning experiences must build such skills (not just convey that they are important) and teach skills that are transferable to new situations” (Shackelford et al., 2006). Each of the specific computing domain curriculum guides provides varying degrees of detail on the professional values and principles commonalities laid out in the CC2005 Overview Report. Examining the details of each domain’s guidelines provides insights into the perspectives of each task force’s emphasis at the time and perceived value for each within the respective curriculum.

Teamwork Skills in Computing Curriculum Guidelines

General consensus among the research literature explicates that “a team consists of two or more individuals, who have specific roles, perform interdependent tasks, are adaptable, and share a common goal” (Baker, Day, & Salas, 2006; Salas, Dickinson, Converse, & Tannenbaum, 1992) baker p, 1579. To effectively function as a team, members must encompass the knowledge, skills and attitude to be able to have knowledge of and monitor their own and each other’s performance and have a positive attitude to work as part of a team. (Baker et al., 2006; Cannon-Bowers, Tannenbaum, Salas, & Volpe, 1995; Sims, Salas, & Burke, 2004).

The CS2013 guidelines classify teamwork, verbal and written communication, time management, problem solving and flexibility as “soft skills,” specifically emphasizing the value of these skills in preparing students for professional practice. The guidelines couple this with the importance of personal attributes including risk tolerance, collegiality, patience, work ethic, identification of opportunity, sense of social responsibility, and appreciation for diversity (Sahami et al., 2013). Both general college experience as well as specific curriculum delivered

through a combination of required courses along with short modules in other courses is recommended. Although the CS2013 curriculum guidelines specify that a “Computer Science program encourages the development of soft skills and personal attributes,” little evidence of this is discerned throughout the document (Sahami, 2013). Rather than an overarching Computer Science skill, or a skill threaded throughout the entire curriculum, teamwork is explicitly designated as a part of the Software Engineering Knowledge Area under Project Management within CS2013 (p. 15). Instead, the content topics of ethics, privacy, civil liberties issues and economics of computing included in the Social Issues and Professional Practice (SP) KA are distinctly identified, and the exclusion of teamwork and problem solving along with other ‘soft skills’ appear as a considerable omission from the overarching curriculum.

In contrast, the IT2017 guidelines task force extends their approach to incorporating professional practice into the curriculum by offering that both IT courses as well as courses outside of the IT degree program, along with cooperative education requirements, can offer “learning experiences that emphasize team work, authentic projects, outside clients, relevant aspects of IT work, employers’ direct involvement and use of professional tools and platforms” Also grounded earlier in the guidelines, the task force claims that, “the IT graduate is a collaborative problem-solver” with a specific educational outcome for an IT graduate to “function effectively on teams and employ self- and peer-advocacy to address bias in interactions, establish goals, plan tasks, meet deadlines, manage risk, and produce deliverables” (Sabin et al., 2017).

The SE2014 guidelines initially recognized the importance of ‘soft skills’ by citing the aforementioned National Association of Colleges and Employers (NACE) 2013 survey emphasizing communication skills, ability to work in a team, problem-solving skills, planning

and organizational skills, ability to obtain and process information, and ability to analyze quantitative data, as important to employers, with the same skills appearing in the 2016 survey report (Ardis et al., 2014; NACE, 2016). In addition to the curriculum content in the form of KAs, the task force clearly recognized the importance of curriculum design when producing students with these valued skillsets by noting its impact on professional practice factors such as teamwork, communication, and analytic skills. Teamwork is specifically identified as a critical factor in developing and delivering quality software. Yet the task force also recognized the importance of individual work and contribution and specifically attend to this when offering that, “for group work, students should be informed about the nature of groups and of group activities and roles as explicitly as possible. This must include an emphasis on the importance of such matters as a disciplined approach, adhering to deadlines, communication, and individual and team performance evaluations” (Ardis et al., 2014).

Instructional Approaches in Computing Curriculum Guidelines

Examining each of the computing domain guidelines’ development of curriculum delivery methods and approaches provides yet another insight into the value placed by each task force on professional or ‘soft skills.’ While most attention in the general CC2005 is devoted to identifying the specific outcomes of the various computing programs as well as key aspects, a noteworthy commonality around instructional approaches is identified as the “exposure to an appropriate range of applications and case studies that connect theory and skills learned in academia to real-world occurrences to explicate their relevance and utility” (Shackelford et al., 2006).

The CS2013, IT2017 and SE2014 guidelines each included aspects highlighting the importance of real-world examples and content within the curriculum. The CS2013 guidelines

repeatedly, in both the KA guidelines as well as a series of exemplar course outlines, referenced the mapping of real-world problems to curricular aspects such as selecting a specific programming language or algorithmic solutions.

The IT2017 guidelines highlighted, “Problem based assignments, real world projects, and laboratory activities with workplace relevance are examples of curriculum elements that focus on developing skills” (Sabin et al., 2017). Hands-on activities were correlated with real-world experience through workforce collaborations, particularly in the form of internships, mentoring, and part-time jobs. Yet hands-on exercises are primarily relegated to being noted as critical components of work experiences (Sabin et al., 2017).

The SE2014 task force expressly identified pedagogical approaches such as problem-based learning, just-in-time learning, learning by failure and technology-enhanced learning. Curriculum Guideline 14 specifically highlighted that, “the curriculum should have a significant real-world basis. Incorporating real-world elements into the curriculum is necessary to enable effective learning of software engineering skills and concepts” (Ardis et al., 2014). Case studies, where students are exposed to real systems and project-based activities where typical industry projects are modeled including group work, presentations and formal reviews are highlighted as examples.

While conceivable that the CS2013 guidelines task force considered the issues of teamwork, problem solving and other ‘soft skills’ to be more relevant in regards to curriculum delivery than Knowledge Areas, the document provides little support for such an argument. Considerable attention is afforded to providing the reader with clarity around details regarding the flexibility and recommendations of the CS2013 task force, including consideration for institutional context, resources, faculty focus and differing core and elective course requirements.

Details regarding providing computer science to diverse audiences is also to be noted, with a focus on the multiple pathways to computing and explanation of both the benefits and the tradeoffs. Yet, little attention is devoted at all to the method of curriculum delivery, with the most explicit being the mention of the ‘lecture hour’ defined simply as a universal unit of measurement to indicate time expected on a topic, with a clear caveat that the report is not endorsing the traditional lecture format (Sahami et al., 2013).

The 518 page CS2013 report focuses on the KAs with exemplar curricular examples serving as nearly half the report. The sheer volume of exemplar coursework indicates a noble effort on the part of the task force to “identify and describe existing successful courses and curricula to show how relevant knowledge units are addressed and incorporated in actual programs” (Sahami et al., 2013). While the effort appears successful in its intent to promote ongoing engagement among computing educators by sharing ideas of new curricula, the impact would be strengthened with some elucidation as to how courses were determined to serve as exemplars or what the idea of a ‘successful’ course indicates. While not specifically noted as an instructional approach, the idea of using a ‘hands-on’ delivery method is referenced in 13 of the 88 exemplar courses outlined and student reflection as a curricular or assessed component is identified in four of the exemplar courses.

In contrast, the IT2017 guidelines devote an entire section to “A Performance Perspective on Learning.” Teacher-focused didactic teaching is distinguished from student-focused learning, highlighting active learning, where “students themselves create meaning and develop understandings with the help of appropriately designed learning activities” (Barr & Tagg, 1995; Sabin et al., 2017). The task force explicitly embraces this paradigm shift in educational approach of “providing instruction dominated by a passive lecture-based learning environment”

to “producing and creating experiences in which students are active participants in the learning process” (Barr & Tagg, 1995; Sabin et al., 2017).

Diverging from the previous IT2008 guidelines that used a content driven approach to framing curriculum, the IT2017 guidelines integrate an Understanding by Design (UbD) framework to focus on a *competency-based* IT curriculum framework. In this framework, content mastery is a means rather than the end to achieving long-term achievements for graduates (Kennedy, Hyland, & Ryan, 2009; Sabin et al., 2017; Wiggins & McTighe, 2011). As demonstrated in *Figure 1*, the IT task group operationally defined the IT competencies of knowledge, skills and dispositions by linking the curriculum within a professional context that includes what are labeled as ‘soft skills’ such as collaboration, working on authentic problems, reflective practice, teamwork and problem solving.

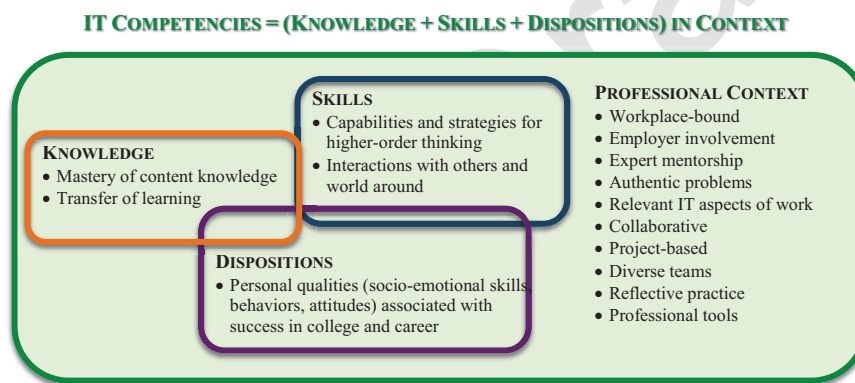


Figure 36. *IT competencies* (Sabin, et al., 2017, p. 29).

This IT2017 guidelines emphasize that there is value in not simply presenting these skills to students in a single assignment or course, but rather developing real skills in graduates to address the industry imperatives (Sabin et al., 2017). Although undefined, ‘experiential learning’ is explicitly identified as highly valued alongside technical knowledge and soft skill acquisition. (Sabin et al., 2017). Hands-on experience is implied to constitute experiential learning and is encouraged to mitigate students’ lack of experience before entering the workforce. Internships,

work-study programs and part-time work experience are cited as productive opportunities for students to have a constructive experience. Live lab exercises, synonymous with hands-on and experiential activities in the report are also identified as highly valued in enabling students to ‘learn by doing’(Sabin et al., 2017).

The SE2014 task force also recognized the shifting paradigm of teaching to learning, specifically noting that,

Software engineering education needs to move beyond the lecture format and to consider a variety of teaching and learning approaches. The most common approach to teaching software engineering material is the use of lectures, supplemented by laboratory sessions, tutorials, and so on. However, alternative approaches can help students learn more effectively. Central to designing learning activities for software engineering is recognition of the need for students to participate in time-limited, iterative development experiences. In addition to reflecting common industry practice, iterations are important to motivating student learning. Iterating on prior work helps students see the deficiencies of their efforts in prior iterations and provides an opportunity for reflection and improvement that would otherwise be unavailable. (Ardis et al., 2014)