Using Mapping Studies in Software Engineering

David Budgen[1], Mark Turner[2], Pearl Brereton[2], and Barbara Kitchenham[2]

[1] Department of Computer Science, Durham University
david.budgen@durham.ac.uk
[2] School of Computing & Maths, Keele University
{m.turner,o.p.brereton,b.a.kitchenham}@cs.keele.ac.uk

Abstract.
Background: A mapping study provides a systematic and objective procedure for identifying the nature and extent of the empirical study data that is available to answer a particular research question. Such studies can also form a useful preliminary step for PhD study.
Aim: We set out to assess how effective such studies have been when used for software engineering topics, and to identify the specific challenges that they present.
Method: We have conducted an informal review of a number of mapping studies in software engineering, describing their main characteristics and the forms of analysis employed.
Results: We examine the experiences and outcomes from six mapping studies, of which four are published. From these we note a recurring theme about the problems of classification and a preponderance of 'gaps' in the set of empirical studies.
Conclusions: We identify our challenges as improving classification guidelines, encouraging better reporting of primary studies, and argue for identifying some 'empirical grand challenges' for software engineering as a focus for the community[1].

1 Introduction

The evidence-based paradigm has had considerable success in terms of influencing research and practice in clinical medicine. In turn, the ideas and the practices involved (most notably, the *Systematic Literature Review*, or SLR), have been adopted and adapted by many other disciplines that depend upon empirical data for such purposes as building theories and obtaining an understanding of practice, such as education, psychology and many of the health-related and social sciences. The use of a Systematic Literature Review is particularly intended to provide an unbiased, objective and systematic approach to answering a research question by finding all of the relevant research outcomes from 'primary' empirical studies, and aggregating these—paying appropriate attention to quality—and is therefore regarded as forming a 'secondary' study.

Computing in general, and software engineering in particular, has a very weak track record of using empirical data to support the development of models and methods, as well as of adopting practices from other disciplines (Whitley 1997, Glass et al. 2002, Glass et al. 2004). However in 2004 it was suggested that the evidence-based paradigm might form a valuable addition to our empirical practices that would help address this problem (Kitchenham et al. 2004). This has been adopted with enthusiasm, particularly in Europe and Scandinavia: to the effect that, between that date and the end of 2007, we are aware of at least 20 published SLRs—albeit with the caveat that many were exploring research practices rather than those relevant to systems development (Kitchenham et al. 2007). In addition, drawing upon various sources of experience and expertise, the *Guidelines* for employing evidence-based practices in software engineering that were originally formulated in 2004, based upon the clinical guidelines that were all that were then available, have recently been extensively updated (Kitchenham & Charters 2007)[2].

A more 'open' form of SLR, intended to 'map out' the research that has been undertaken rather than to answer a detailed research question is usually termed a *Mapping Study* or a *Scoping Review* (Kitchenham & Charters 2007, Petticrew & Roberts 2006). Such a study is intended to identify 'gaps' in the set of primary studies, where new or better primary studies are required, as well as 'clusters' where there may be scope for more complete SLRs to be undertaken.

[1] We have the abstract in 'structured' form for this paper–for reasons that will be explained later.
[2] Available from www.ebse.org.uk.

Reviews, especially 'expert' reviews, are of course not unknown in computing, although they are not undertaken very frequently. Within the PPIG context, the study of metacognitive theories of visual programming by Blackwell (1996) could well be categorised as an excellent example of an expert review. The paper by Whitley (1997) falls into a similar category—however, although it is concerned with the methodological aspects of the primary studies it review, this paper provides no specific framework for the review process itself! So, a major distinguishing characteristic for a systematic review or mapping study is that the processes used for searching, and the criteria used for inclusion/exclusion are explicitly defined in the research protocol and reported as part of the outcomes.

We ourselves are involved in undertaking both SLRs and mapping studies in software engineering, and the purpose of this paper is to provide an informal summary of some experiences with the use of mapping studies, both our own and also those conducted by others. Some of these have been performed as part of a research project, while others have formed the basis of student projects—at both postgraduate and advanced undergraduate levels. Collectively they provide both a useful (if rather selective and informal) survey of how extensively some key software engineering topics have been subjected to empirical evaluation, as well as a source of experiences on how and when such studies might usefully be performed. For this paper, we have therefore set out to provide an informal *tertiary review* that itself reviews the use of this form of secondary study and identifies particular trends! Our underlying research question is to assess how extensively some of the major elements of software engineering practice are underpinned by empirical studies, with a secondary question about how far the use of mapping studies is suitable for student use? (We should note that we have not attempted to assess how extensively the empirical papers support or refute the use of particular practices—confining our analysis to the extent of the evidence available, rather than the content.)

In the rest of the paper we briefly summarise the form of a mapping study; review the main features of those known to us as having been undertaken on software engineering topics; identify the main challenges of performing such a review; and provide some recommendations that themselves may form the basis of further discussion.

2 The form of a mapping study

Petticrew and Roberts (2006) suggest that such a study "involves a search of the literature to determine what sorts of studies addressing the systematic review question have been carried out, where they are published, in what databases they have been indexed, what sorts of outcomes they have assessed, and in which populations".

The early stages of a mapping study are generally very similar to those of a systematic literature review, although the research question itself is likely to be much broader, in order to adequately address the wider scope of such a study. These three stages are:

1. identification of primary studies that may contain relevant research results (searching);
2. selecting the appropriate primary studies from these after further examination (inclusion/exclusion);
3. where appropriate, performing a quality assessment of the selected studies (bias/validity).

For a 'conventional' SLR, these stages would then be followed by data extraction and aggregation (analysis). However, for a mapping study this process is generally much broader, and both data extraction and analysis are largely concerned with classification of the available studies along the lines indicated in the quotation from Petticrew and Roberts.

One of the main reasons for undertaking the informal survey provided here is to examine how different mapping studies have performed this task of analysis. While the software engineering *Guidelines* (Kitchenham & Charters 2007) provide considerable guidance on performing data extraction and analysis for an SLR, they provide very little advice on how to undertake these tasks for a mapping study. Indeed, it may only be possible to determine the most appropriate forms of analysis once the form and extent of the available dataset is known and has been classified in some way. Our experiences suggest that deciding how to classify and categorise the studies found may well provide one of the major problems

for the inexperienced analyst—and given that mapping studies are (superficially at least) a potentially useful starting point for research students and even for some advanced undergraduate projects, it would seem to merit closer examination.

3 The examples of mapping studies

In this section we summarise our knowledge of existing mapping studies. As indicated above, some are already completed and the results have been published, while others are still in progress, possibly with some interim published findings. Table 1 provides a summary of the key features of the set of mapping studies that are discussed in more detail in the rest of this section. The number of primary studies is given in bold where the classification process has been subjected to some form of validation by either the use of two analysts or an analyst and checker, and other figures should therefore be treated as interim and therefore tentative. We should note here that we have not included the survey by Glass et al. (2002) since this was only a partial sample (they selected one paper in five) and was also concerned with identifying trends in the discipline rather than analysing specific topic areas.

Table 1. Summary of Mapping Studies in Software Engineering

Authors & Year Published	*Period Searched*	*Topic*	*No. of Primary Studies*	*Form of Analysis*	*Sources Searched*
Juristo (2006)	up to 2005	Unit Testing Techniques	**24**	Classify by technique for test--set generation and selection	IEEE ACM
Jørgensen & Shepperd (2007)	up to 4/2004	Software Development Effort & Cost Estimation	**54 (304)**	Classify by topic, estimation approach, research approach study context & dataset	76 journals
Søberg et al (2005)	1993–2002	Use of Experimental Studies	**103**	Classify by topic, form of study, participants etc.	9 journals 3 conferences
Bailey et al. (2007)	up to 2007	Object Oriented Design	138	Classify by forms of study, and forms of intervention	IEEE ACM Science Direct Web of Science Google Scholar
Cheng (n/a)	1995–2008	Software Design Patterns	185 (55 expts)	Classify against 8 assertions derived from patterns literature and by form of study	IEEE ACM Science Direct Web of Science CiteSeer Google Scholar
Pretorious (2008)	1994-2008	UML features	**33**	Classify by forms of study, aspects of the UML	IEEE ACM ScienceDirect Athens

3.1 Juristo et al.—Unit Testing Techniques

Juristo and co-authors have published two papers on this study: (Juristo et al. 2004, Juristo et al. 2006), but as the second study is (for our purposes) essentially an updated version of the first one, we will only discuss this version.

Neither paper gives much detail about such issues as searching, but an important limitation of this particular study is that it was confined to papers published in the IEEE and ACM electronic databases. (We are currently updating this study as part of an investigation into the stability of the first phases of an SLR, which will include also searching a wider set of databases.)

As with almost any mapping study, one of the problems is to find a consistent classification scheme for the topics of the papers. For this study, the authors chose to use the definitions provided in the

IEEE's *Software Engineering Body of Knowledge*—usually referred to as the *Swebok* (Abran et al. 2004). However, it is clear that Juristo *et al.* refined and extended the Swebok classification system to include more detailed categories for code coverage and test set classification. The study was confined to papers describing experiments, and the total number of studies included was 24. We should also note that this is not a straightforward example of a mapping study as the papers do also discuss the details of the particular testing experiments, rather than simply reporting numbers.

The form of analysis undertaken in this study was to tabulate the various studies using two major classification schemes:

- The first table looked at techniques for test-set generation and evaluation, grouping these into the types obtained from the Swebok (tester's intuition and experience; specification-based; code-based; fault-based; usage-based and fault seeding). This table had many gaps, so there were many techniques where there were no experiments for either test-set generation or test-set evaluation, and only a few where there were multiple studies addressing a particular technique.
- The second table looked at test-set selection techniques represented in the papers, categorised using the Swebok (filtering; prioritization and regression).

Since experimenters have their own goals and objectives, many papers were actually classified as coming under more than one heading in each of the tables. (This is a recurring theme in mapping studies of course.)

Overall, the study made only limited attempts to aggregate the outcomes, although the authors were able to make some observations that were based upon these.

3.2 Jørgensen & Shepperd—Software Cost Estimation

This study is reported as (Jørgensen & Shepperd 2007). It is a wider review than some, since it includes all papers relating to estimation of effort and cost (304), not just those that are empirical in nature. For our purposes, we have taken the latter as being those reported as being surveys (27), experiments (19) and case studies (8), giving the total of 54 that we cite in Table 1.

The study was very thorough, involving manual searches of over one hundred journals, and finding relevant papers in 76 of these. Ten journals accounted for two-thirds of the papers, with the remainder being very widespread—and as the authors observe, "reading only the 10 most relevant journals means that important research results may be missed". An electronic search via *Google Scholar* and *Inspec*, performed as a check, failed to find a large proportion of the relevant papers (30%), a problem that was worse for Google Scholar than for Inspec.

This study concentrated on journal papers, excluding those from conferences. The authors partly justified this by observing that many conference papers were later expanded into journal papers, which concurs well with the difference between the datasets used in the two Juristo studies. Like other reviews, they were concerned to report on the number of studies rather than of papers.

Papers were classified by one author, with the second performing a random sampling of 30 papers.

The papers were analysed against eight research questions—some were demographic, such as identifying the relevant research journals and the extent to which others looked at these, but the papers were also classified by research topic and research approach (as quoted above). The authors did not attempt to perform any form of quality estimation, nor did they report on the findings of the different papers.

3.3 Sjøberg et al.—Use of Experimental Studies in Software Engineering

This study, published as (Sjøberg et al. 2005), was methodological in nature and examined how controlled experiments were conducted and reported in software engineering. They used a set of major journals and conferences that publish papers on empirical studies, and searched these manually. The search period was set as a ten-year window (1993–2002) and the search involved examining the titles and abstracts for 5,453 articles, from which they selected 103 papers, reporting on a total of 113 experiments.

The analysis aspect most relevant to this paper involved classifying the papers by topic, which was undertaken using two different schemes. The first of these, as used in (Glass et al. 2002) is aimed at positioning software engineering research against the overall computing context. When assessed using this scheme, the two most dominant categories were *software life-cycle/engineering* (49%) and *methods/techniques* (32%). The authors ascribed this to the "relatively large numbers of experiments on inspection techniques and object-oriented design techniques". The second scheme used was the IEEE keyword taxonomy[3], and using this the two most dominant technical areas were *code inspections and walkthroughs* (35%) and *object-oriented design methods* (8%). Numbers for other areas were generally very small.

Other levels of analysis used included forms of participant (predominantly students); tasks performed (using categories of *plan*, *create*, *modify* and *analyse*); and the environment within which the study took place.

3.4 Bailey et al.—Object-Oriented Design

This study is being undertaken within our own research group, and the preliminary findings have been published as (Bailey, Budgen, Turner, Kitchenham, Brereton & Linkman 2007). The motivation was one of examining a major issue in software engineering (object-orientation) and identifying how extensively this had been studied empirically. In particular we wanted to know:

- What are the most investigated OO design topics and how these have changed over time?
- What are the most frequently applied research methods, and in what study context?

Given that the ideas of object-orientation, and much of the vocabulary, emanated from the 1960s and 1970s, the number of empirical studies could be considered as rather limited (138). Our analyses of these were based upon classification of the papers against a number of relatively informal measures, including:

form of intervention where these were identified (from observation) as being: OO versus non-OO; abstraction; design patterns; metrics and comprehension. Nearly half of the papers were on the topic of metrics.

experimental form with laboratory studies coming out as the dominant form (52) and relatively few studies that could reasonably be considered as 'field studies'.

We did some further classification within these (for example of the themes for the metrics papers) and are currently in the process of conducting a fuller analysis of the data-set. Essentially though, the forms of analysis used so far have been based on inspection.

3.5 Cheng—Software Design Patterns

This is part of an ongoing PhD study being undertaken at Durham University by Cheng Zhang, and as yet the results are unpublished. For this paper we therefore limit our description to a small number of issues that are particularly relevant to our theme.

The concept of software design patterns was popularised by the book by the 'gang of four' (Gamma et al. 1995) and has spawned continuing interest ever since. However, it can be argued that much of the effort has gone into creating and documenting patterns—and there is an open question as to how easily patterns can be found and used by others, especially inexperienced designers. Our study is therefore concerned with finding out how fully patterns have been studied, which aspects have been studied, through which empirical forms, and with what type of participants?

Overall, the study has identified 476 candidate papers of which 186 were classified as being empirical, with 55 of these describing experiments, after filtering for duplicates and applying the inclusion/exclusion criteria. The searching has been particularly thorough, with an electronic search being

[3] www.computer.org/mc/keywords/software.htm

followed up both by a manual search of a number of major journals and also by a process of 'snowballing' (following up the references of the papers found). An interesting aspect of the manual search was that it found rather more papers than we had expected, with these mainly being clustered in the mid-1990s when the concepts of design patterns were still becoming established, and hence when the vocabulary (needed for electronic searching) had not become fully established.

This study has presented an even greater problem of classification than the OO design study discussed above. The eventual solution to this has been to identify the major *assertions* that the patterns literature makes about patterns and their use. Analysis of the major texts and tutorial papers on software design patterns has produced a set of eight major assertions, listed in Table 2, that are commonly made by the patterns community, and we are using these to classify the primary studies (as well as classifying them by such aspects as experimental form etc.).

Table 2. Assertions about software design patterns

1.	"Design patterns make it easier to reuse successful designs."
2.	"Design patterns make it easier to reuse successful architectures."
3.	"Design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability."
4.	"Design patterns helps you identify less-obvious abstracts and the objects that can capture them."
5.	"Design patterns help you define interfaces by identifying their key elements and the kinds of data that get sent across an interface."
6.	"Patterns are a means of documentation."
7.	"Patterns support the construction of software with defined properties."
8.	"Patterns provide a common vocabulary and understanding for design principles."

3.6 Pretorius—models and forms used in the UML

This particular study has been conducted by an advanced undergraduate student at Durham University, Rialette Pretorius, as part of a specialist software engineering module (Pretorius & Budgen 2008). While the task of conducting an exhaustive electronic search and then filtering the results can easily become an overwhelming task for an undergraduate, experience so far indicates that providing a topic is chosen that is reasonably well defined (and known to have only been studied empirically to a limited extent), then this can provide an excellent research training experience for an able student. This study was particularly able to benefit from the student-oriented (and more concise) set of guidelines produced by Austen Rainer and Sarah Beecham (Rainer & Beecham 2008).

For this study, our aim was to find out how extensively the *Unified Modelling Language* or UML had been studied through empirical forms. While the UML has become a *de facto* standard for describing object-oriented designs, its origins do not have any theoretical or analytical underpinnings, stemming more from a series of compromises made when a number of object-oriented 'gurus' joined up to create a 'unified' model for software development. There are now many books on using the UML, there is support provided for it in many software development environments, and the Object Management Group (OMG) have now produced version 2.0 (with a total of 13 diagrammatic notations)[4]. However, like many software engineering forms and practices, neither the elements of the UML nor their use would appear to have been scrutinised very closely.

This study searched most of the major sources of empirical software engineering publications, and identified some 33 papers after the initial search results were filtered for relevance and duplications. These addressed a range of issues, with the largest number being papers that addressed comprehension (11). We might note that this figure is lower than the number of different notations used in the UML. Most of the papers were laboratory experiments (25).

So, for purposes of analysis, the initial classifications were confined to using the titles and abstracts alone, and papers were categorised by forms of publication, aspects of the UML studied and the forms

[4] The most recent specification of the UML, available from www.uml.org, runs to 738 pages.

of study. Further analysis of the dominant aspect (comprehension) identified a number of themes (notational variants, stereotyping for improved comprehension, graphical layout, comprehension prediction, reading approach and training), although most of these were only addressed in one study. The laboratory experiments were also analysed by topic, with comprehension again being the dominant topic (11). Only one paper addressed the question of adoption (by an organisation), this being one of only three papers to employ a survey.

The UML web site claims that a search on a site such as *Amazon* will yield over one hundred books about the UML. A ratio of 3:1 for books advocating the UML against the number of empirical studies analysing the claims about the UML should perhaps be a source of concern for the software engineering community!

4 Discussion

Our analysis is primarily confined to two issues: the extent and completeness of the set of empirical studies addressing a particular topic in software engineering; and the forms of analysis adopted by the authors. In addition, we also consider the practicality of using mapping studies for student projects at both undergraduate and postgraduate levels, and conclude by assessing the threats to validity implicit in our study.

4.1 The extent and completeness of empirical studies in Software Engineering

As can be seen from Table 1, the set of studies reviewed in this paper addressed a wide mix of topics, reviewed these over differing periods of time, and searched using different strategies and different combinations of sources. However, regardless of the approach taken, the scale of these studies remains relatively small when compared with the overall software engineering corpus, echoing the observations of Glass et al. (2002) concerning the dependency of software engineering research on advocacy and analysis in preference to empirical evidence. This is a problem also noted by Whitley (1997), who observed that "system-building techniques, such as object-oriented design, . . . are strongly advocated in the absence of evidence".

A common theme in the studies reviewed here is the difficulty of classifying papers, based solely upon their titles and abstracts, as well as the generally poor reporting qualities of the studies that were selected. There is also a strong reliance on laboratory experiments. In the same general vein, those studies that relied upon searching electronic databases also report problems with the reliability and coverage provided by these. (For a separate review of this theme, see (Bailey, Zhang, Budgen, Turner & Charters 2007).)

4.2 Forms of analysis used for Mapping Studies

While most of these studies have concentrated on classification, even where they appear to be similar (such as classifying by form of study), it is worth noting the lack of any common taxonomy describing empirical forms used in software engineering. However, most variation tends to occur when grouping papers by topic—perhaps because the original primary studies lack any sense of being positioned in some overall framework, and indeed, are generally reported from an isolated viewpoint rather than with the idea of contributing to a wider set of studies.

Analysis certainly provides a major challenge for surveys that are conducted by students. We discuss this further in the next sub-section.

4.3 Are mapping studies suitable for student use?

Most student projects in software engineering, whether conducted by undergraduates or postgraduates, involve an element of literature survey, and indeed, we have examples of both undergraduate and postgraduate studies in the set we have examined. Our experiences of encouraging students to adopt systematic practices when conducting a review (i.e. performing a mapping study) have been reasonably

encouraging, and can help change the planned direction for a research project. However, there are also some cautions that we need to recognise.

- A mapping study is very time-consuming. Even for a PhD topic, it generally extends well beyond the time normally spent on background reading. In compensation, a thorough mapping study can itself provide additional opportunity for publication. For undergraduates, the topic needs to be chosen with care, to ensure a manageable number of 'hits', and it may also be necessary to constrain searches to a specific set of electronic databases such as IEEE and ACM. There is also a need for guidelines that are tailored to their needs and timescale, such as (Rainer & Beecham 2008).
- Classification does seem to present a challenging task for students, especially where they are largely confined to working with the titles and abstracts for the papers they have found (which is certainly the case for undergraduates). The *Guidelines* advise that data extraction should preferably be performed using two or more independent analysts, or at least one analyst and a checker, with the latter possibly reviewing a random sample of papers—but this is not always practical for student projects (or at least, places a significant burden upon the supervisor). We might also note that even experienced researchers can differ substantially when classifying a paper (Jørgensen & Shepperd 2007).

Classification problems may arise for a variety of causes: students may not be fully familiar with the topic being studied (hence the need to perform a review); and they may also lack a solid understanding of the empirical terminology employed in the papers. Overall, since a mapping study essentially involves trying to find ways of classifying what is found against the framework created by the original research question, the essentially creative element this involves this probably constitutes a much more difficult task for the novice than for the experienced researcher.

4.4 Threats to validity

The main issue here is whether our approach adequately addresses the principal research question, which itself involves an assessment of how thoroughly software engineering ideas and practices are underpinned by empirical studies. There is obviously scope for *publication bias* to influence our conclusions, partly because we have not performed a systematic search for such surveys, and also because the set selected only covers a limited range of forms and topics.

Regarding the first of these, we should observe that we have already performed a more systematic search for secondary studies as part of our *tertiary* study on systematic reviews (Kitchenham et al. 2007) and that this would have been likely to have turned up other systematic mapping studies. In addition, we are active researchers in this area, one is an associate journal editor with responsibility for systematic reviews (BAK) and hence we would be reasonably likely to have identified further published studies. Overall therefore, we would argue that we are unlikely to have missed any significant published reviews appearing in the mainstream software engineering literature. The position regarding the unpublished studies is less convincing—we have essentially selected these on a convenience basis, supported by our expert judgement, and so it is quite possible that there are other studies that we are not aware of.

The set of topics addressed is inevitably selective, given so few mapping studies. However, here, the inclusion of the paper by Sjøberg et al. (2005) is significant, since it addresses techniques rather than a specific software engineering topic. From this, we can suggest that, code inspections apart, we are unlikely to have omitted a major software engineering topic that has a substantial corpus of empirical evaluation.

We should also note that, as researchers in this area of evidence-based software engineering, we may have been biased in our selection and in our analysis, although obviously, we have tried to avoid this.

5 Conclusions

Adoption of the evidence-based paradigm in software engineering has the potential to create a much sounder basis for standards, policies and practices. However, to do so, it will need a stronger underpinning of primary studies than currently appears to be available. Indeed, the set of mapping studies

discussed here, although small, raises a number of questions that the wider empirical community might well need to focus upon, and some of the more obvious ones are as follows.

- A recurring theme in this set of studies is that of how to classify and assess (for relevance) the primary studies that are found. This is a particular challenge for inexperienced researchers (students) as well as for more experienced researchers who may not be familiar with the detailed characteristics of a particular field. Indeed, even for a relatively topic-independent classification such as the form of research method used, it can often be hard to determine this from the title and abstract. So, while a mapping study can potentially provide an excellent starting point for a student project, it is only likely to do so where the question of classification is reasonably tractable. It may well be therefore that the most valuable step that we can make at this point is to provide fuller advice in documents such as the *Guidelines*, but if so, what should this be?
- The mapping studies reviewed tend to identify many 'gaps' in empirical knowledge, and relatively few 'clusters' where we have reasonably extensive knowledge, and where more systematic reviews are likely to be practical. So one question is whether we in the empirical software engineering community might do well to seek out our own 'grand challenges' and try to focus our collective effort on some key topics that will be of importance to practitioners and policy-makers.
- Another recurring theme, both in these studies and also in the SLRs already published, is the poor quality of reporting of empirical studies in software engineering, both in terms of the papers and of their abstracts. Some guidelines for this do exist (Kitchenham et al. 2002, Jedlitschka et al. 2008), but maybe we need stronger mechanisms to help encourage authors to comply with them—possibly a role that journal editors and others could perform? And, from our own experience, we would argue for greater use of structured abstracts—as used for this paper—to assist with data extraction (Kitchenham et al. 2008, Budgen et al. 2008).

Evidence-based software engineering is about identifying good practice with the aim of transferring it to practitioners and making it available to policy-makers. SLRs support EBSE by providing a methodologically rigourous means of aggregating evidence. Thus SLRs should be of value to any researchers, whether software engineers or cognitive scientists who want to provide evidence that the procedures and practices that they recommend are truly of value to practitioners. However, before we can undertake SLRs, we need to have a better understanding of what evidence is available within a domain—and mapping studies are particularly useful for this purpose. They are useful for identifying the areas where there is sufficient information for an SLR to be effective, as well as those areas where more primary studies are needed. They also have an important role to play in education. In the short term, we believe that, used appropriately, they can provide a valuable starting point for PhD students who need to organise and understand the existing research work in a specific domain. In the longer term, as more mapping studies are published, we believe that they will have the potential to change the way that we do our research—for example, Jørgensen and Shepperd's paper provides a major resource for anyone undertaking research into software cost estimation. We hope therefore that the next generation of researchers will be able to build on and extend existing mapping studies and SLRs, rather than starting from scratch, so making better use of scarce research effort.

Acknowledgements

This work was funded by grants from EPSRC (*Evidence-Based Software Engineering*, *Evidence-based Practices Informing Computing*. We would like to acknowledge all who have contributed to the various mapping studies cited here, and in particular: John Bailey, Cheng Zhang and Rialette Pretorius. We also thank the anonymous referees for their very helpful comments and suggestions.

References

Abran, A., Moore, J. W., Bourque, P. & Dupuis, R., eds (2004), *Guide to the Software Engineering Body of Knowledge*, IEEE Computer Society.

Bailey, J., Budgen, D., Turner, M., Kitchenham, B., Brereton, P. & Linkman, S. (2007), Evidence relating to Object-Oriented software design: A survey, *in* 'Proceedings of Empirical Software Engineering & Measurement, 2007', IEEE Computer Society Press, pp. 482–484.

Bailey, J., Zhang, C., Budgen, D., Turner, M. & Charters, S. (2007), Search Engine Overlaps: Do they agree or disagree?, *in* 'Proceedings of REBSE-2 Workshop, ICSE 2007', IEEE Computer Society Press.

Blackwell, A. F. (1996), Metacognitive Theories of Visual Programming: What do we think we are doing?, *in* 'IEEE Symposium on Visual Languages (VL'96)', IEEE Computer Society Press, pp. 240–246.

Budgen, D., Kitchenham, B. A., Charters, S., Turner, M., Brereton, P. & Linkman, S. (2008), 'Presenting software engineering results using structured abstracts: A randomised experiment', *Empirical Software Engineering* . Accepted for publication.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Glass, R., Ramesh, V. & Vessey, I. (2004), 'An Analysis of Research in Computing Disciplines', *Communications of the ACM* **47**, 89–94.

Glass, R., Vessey, I. & Ramesh, V. (2002), 'Research in software engineering: An analysis of the literature', *Information & Software Technology* **44**, 491–506.

Jedlitschka, A., Ciolkowski, M. & Pfahl, D. (2008), Reporting experiments in software engineering, *in* F. Shull, J. Singer & D. Sjøberg, eds, 'Guide to Advanced Empirical Software Engineering', Springer-Verlag, London, chapter 8.

Jørgensen, M. & Shepperd, M. (2007), 'A Systematic Review of Software Development Cost Estimation Studies', *IEEE Tran. on Software Engineering* **33**(1), 33–53.

Juristo, N., Moreno, A. & Vegas, S. (2004), 'Reviewing 25 years of testing technique experiments', *Empirical Software Engineering* **9**(1), 7–44.

Juristo, N., Moreno, A., Vegas, S. & Solari, M. (2006), 'In search of what we experimentally know about unit testing', *IEEE Software* **23**(6), 72–80.

Kitchenham, B., Brereton, P., Owen, S., Butcher, J. & Jefferies, C. (2008), 'Length and readability of structured software engineering abstracts', *IET Software* **2**, 37–45.

Kitchenham, B., Budgen, D., Brereton, P. & Turner, M. (2007), 2nd international workshop on realising evidence-based software engineering (REBSE-2): Overview and Introduction, *in* 'Proceedings of REBSE-2 Workshop, ICSE 2007', IEEE Computer Society Press, pp. 1–5.

Kitchenham, B. & Charters, S. (2007), Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Kitchenham, B., Dybå, T. & Jørgensen, M. (2004), Evidence-based software engineering, *in* 'Proceedings of ICSE 2004', IEEE Computer Society Press, pp. 273–281.

Kitchenham, B., Pfleeger, S. L., Pickard, L., Jones, P., Hoaglin, D., Emam, K. E. & J.Rosenberg (2002), 'Preliminary Guidelines for Empirical Research in Software Engineering', *IEEE Transactions on Software Engineering* **28**, 721–734.

Petticrew, M. & Roberts, H. (2006), *Systematic Reviews in the Social Sciences: A Practical Guide*, Blackwell Publishing.

Pretorius, R. & Budgen, D. (2008), A mapping study on empirical evidence related to the models and forms used in the UML. Accepted for publication as a short paper in ESEM 2008.

Rainer, A. & Beecham, S. (2008), Supplementary guidelines and assessment scheme for the use of evidence-based software engineering, Technical Report CS-TR-469, School of Computer Science, University of Hertfordshire.

Sjøberg, D., Hannay, J., Hansen, O., Kampenes, V., Karahasanovic, A., Liborg, N. & Rekdal, A. (2005), 'A survey of controlled experiments in software engineering', *IEEE Transactions on Software Engineering* **31**(9), 733–753.

Whitley, K. N. (1997), 'Visual Programming Languages and the Empirical Evidence For and Against', *Journal of Visual Languages and Computing* **8**, 109–142.