e-Informatica Software Engineering Journal, Volume 3, Issue 1, 2009

What support do systematic reviews provide for evidence-informed teaching about software engineering practice?

David Budgen*, Pearl Brereton**, Nikki Williams***, Sarah Drummond****

*Department of Computer Science, Durham University

**School of Computing & Maths, Keele University

***Centre for Electronic Warfare, Information & Cyber, Cranfield University

****Department of Computer Science, Durham University

david.budgen@durham.ac.uk, o.p.brereton@keele.ac.uk, nikki.williams@cranfield.ac.uk,

Abstract

Background: The adoption of the evidence-based research paradigm by software engineering researchers has created a growing knowledge base provided by the outcomes from systematic reviews.
Aim: We set out to identify and catalogue a sample of the knowledge provided by systematic reviews, to determine what support they can provide for an evidence-informed approach to teaching about software engineering practice.
Method: We undertook a tertiary study (a mapping study of systematic reviews) covering the period to the end of 2015. We identified and catalogued those reviews that had findings or made recommendations that were considered relevant to teaching about industry practice.
Results: We examined a sample of 276 systematic reviews, selecting 49 for which we could clearly identify practice-oriented findings and recommendations that were supported by the data analysis provided in the review. We have classified these against established software engineering education knowledge categories and discuss the extent and forms of knowledge provided for each category.
Conclusion: While systematic reviews can provide knowledge that can inform teaching about practice, relatively few systematic reviews present the outcomes in a form suitable for this purpose. Using a suitable format for presenting a summary of outcomes could improve this. Additionally, the increasing number of published systematic reviews suggests that there is a need for greater coordination regarding the cataloguing of their findings and recommendations.

1. Introduction

Over the half-century since software engineering became recognised as a distinct sub-discipline of computing [64], a degree of consensus has emerged about what it encompasses [14], as well as about the skills and knowledge that are needed by software engineers. For the latter, the ACM and IEEE produced a set of curriculum guidelines in 2004 aimed at consolidating ideas about what a software engineer should acquire from an undergraduate education, and this was updated in 2015 after wide consultation across academia and industry [5].

[0] The work reported in this paper was partly undertaken when Nikki Williams was employed by Keele University.

However, although there is fairly general agreement about *what* a software engineer should know, much less attention has been given to *how* that knowledge might be obtained. Indeed, much of our knowledge is still based upon 'expert opinion', and although this is largely derived from experience, it lacks rigour as the foundation for what aspires to be an engineering discipline [49]. And, even when more systematically-acquired evidence is available, this does not necessarily mean that it will be readily accepted or adopted by practitioners [25, 79].

This raises two related questions. The first is concerned with how rigorous knowledge about the effectiveness of software engineering procedures might be derived (that is, how can we identify what works or doesn't work, and under what conditions?). And then when we have such knowledge, how can it most usefully be used for educating students?

In many disciplines, the major source of such knowledge is practice-related research, which is usually derived from 'field studies' of the effects that arise from the use of some intervention. (For software engineering, the interventions might be the introduction of innovative technologies or processes, such as the use of agile practices.)

In software engineering research, there has been increasing use of empirical studies as a means of obtaining knowledge about software engineering practice. A comparison of the characteristics of papers submitted to, and accepted by, the ICSE conferences in 2002 and 2016 shows a significant increase in the reporting of empirical studies and the use of empirical models [87]. In particular, while no papers reporting empirical studies were accepted in 2002, this category made up 30% of the accepted papers in 2016.

Researchers have also adopted the *evidence-based* paradigm as a means of aggregating the knowledge available from a set of 'primary' studies that investigate a given topic, based upon the use of the *systematic review* as its main tool [51]. This in turn has helped to create a growing knowledge base of research findings about software engineering procedures that should potentially be able to inform teaching (and hence, implicitly, inform practitioners). In [51], the authors suggested that adopting evidence-based software engineering (EBSE) would potentially provide:

- "A common goal for individual researchers and research groups to ensure that their research is directed to the requirements of industry and other stakeholder groups."
- "A means by which industry practitioners can make rational decisions about technology adoption."

For the study reported here we consider teachers and students to be additional stakeholders. Teachers can be regarded as being direct beneficiaries, as such knowledge can lend appropriate authority to reinforce teaching about software engineering topics. We view students as being indirect stakeholders, largely benefiting through the material presented by their teachers, rather than through direct use of the findings from systematic reviews.

To set this paper into context, we explain here how it originated and how it relates to other analyses that we have published. As experienced teachers, we wondered whether knowledge derived from the use of EBSE might be used in support of our teaching about software development practices. We envisaged that this support would have a number of forms, but our main expectation was that they might provide some authoritative support for the use of particular practices, or at least, an indication of when these were likely to be effective (or otherwise). In addition we expected that we might obtain some examples from experience about how or when to adopt new technologies.

In order to identify the extent and forms of knowledge about practice that was available, we originally undertook a study of a sample of the systematic reviews that were available up to the middle of 2011, selected on the basis that their topics related to practice, with our findings being reported in [20]. Although that study identified a set of potentially useful systematic reviews, in trying to use these to inform our teaching, we realised that they rarely presented their findings in a readily-usable form. So, beginning in 2016, we undertook a further study (reported here) that extended the earlier one in two ways. Firstly, we included systematic reviews published to the end of 2015, so including more reviews that were undertaken when their form had become more established. Secondly, we have performed a more comprehensive process of selection and analysis, requiring that a review should not only cover a topic relevant to practice, but also provide topic-related findings that were supported by its analysis of the available data.

While conducting this review, the problems encountered in identifying both relevant information about the processes followed in the reviews, as well as about their findings, led us to use our material to analyse and report on the ways that systematic reviews in software engineering were being reported [18] before writing a summary of our findings (this paper). Our aim was to persuade authors and reviewers of the urgent need to improve the quality of published reviews.

A separate question that arose was how extensively practitioners formed the participants in the primary studies used in our set of 49 systematic reviews and to what extent these were conducted in an industry setting ('field studies')? Addressing this involved further additional data extraction, with the outcomes reported in [19]. We do discuss some of the findings from this analysis later, as they provide useful supplemental information about the context of the knowledge available from the systematic reviews.

We begin by examining the evidence-based paradigm and the way that this has been employed by other disciplines. We then examine how its use has been adopted in software engineering; identify the forms of knowledge systematic reviews can provide; describe the design and conduct of our own study; and report our findings. We also examine the ways that other disciplines have used such knowledge to inform their teaching, and what we might learn from their experiences.

2. The evidence-based paradigm and software engineering

Use of the evidence-based paradigm originated in what has become known as *Evidence-Based Medicine* (EBM), by which a medical practitioner can draw upon the findings and recommendations from systematic reviews to aid them in making decisions about how to treat individual patients. Some of its success, in terms of its widespread adoption, derives from the nature of the clinical studies used in the reviews. While these are human-centric, the participants are usually *recipients* of the treatment being studied, and so any variation in the outcomes is likely to occur mainly because of physiological differences among the participants. This, together with the extensive use of Randomised Controlled Trials (RCTs)—which are field experiments with rigorous controls, allows the findings from a set of primary studies to be synthesised using statistical meta-analysis [35]. Use of such forms of analysis makes it possible to assign a high level of confidence to the outcomes.

The evidence-based paradigm has also been successfully adapted to the needs of other 'social' disciplines (in which humans *interact* with each other), including management, education and psychology as well as to more general social and health-related fields [10, 73, 13]. For these, other forms of synthesis that may be more appropriate to particular forms and mixes of primary studies have been developed. An overview of the forms that are potentially useful for software engineering is provided in [23], and in addition, a form of synthesis that can aggregate qualitative and quantitative evidence has been proposed for use in software engineering research [61].

While the term evidence-based software engineering (EBSE) is often used in analogy with evidence-based medicine (EBM), this can lead to inflated expectations. Rather than RCTs, empirical research in software engineering employs a mix of primary study forms that is actually more typical of the social sciences. In addition, the 'treatment' used in software engineering studies usually involves participants in actively performing creative tasks related to software development, rather than being passive recipients. Since these tasks are likely to differ in detail between studies, this makes it more difficult to synthesise the data using forms such as statistical meta-analysis. (Comparison with a number of other disciplines using systematic reviews suggests that the discipline most similar to software engineering is that of Nursing and Midwifery [17], which helps to highlight the 'social' nature of software engineering, where humans both interact with each other, and also with (or via) technology.)

For many disciplines, systematic reviews, are apt to be commissioned by policy-makers and research agencies, and hence the topics studied are likely to be ones considered to be of strategic importance to that discipline and its practitioners. In addition, the task of searching for primary studies will often be performed by trained librarians [13]. In contrast, for systematic reviews on software engineering topics:

- coverage of key topics is uneven (see Appendix B) and the choice of topics appears to be almost entirely researcher-driven, with little to indicate that professional bodies, research agencies or industry have so far taken much interest in identifying suitable topics;
- the quality of reviews is apt to be uneven, particularly with regard to the rigour with which the primary studies are selected, categorised and synthesised [78].
- reporting is apt to be poorly structured and findings are not presented clearly [18];
- many studies use unnecessarily weak forms of synthesis [23];

Together, these influence the form and quality of the available knowledge.

3. The nature of Software Engineering Knowledge

In this section we consider what forms of knowledge useful for teaching about software engineering practice can be provided from systematic reviews.

3.1. The nature of the knowledge provided from systematic reviews

The knowledge provided from any systematic review can be expected to be organised around the research question that the review is seeking to answer, as well as whether this question is concerned with issues related to research or to practice. Three important aspects of this knowledge are: the form in which the findings are presented; the strength of evidence supporting these findings; and how useful they are.

In terms of their usefulness for teaching, in examining the reviews we selected, we have observed that systematic reviews commonly provide knowledge about practice in three different forms (and obviously, the findings of any review may consist of a mix of these).

The first way in which the presentation of the findings is structured is concerned with knowledge that has been derived from the *experiences* of others, in the form of lessons that have been derived about particular software engineering activities. The investigation of the effects of user participation in software development reported in [1] offers a good example of a topic where presenting qualitative knowledge about the experiences of others may well be the most useful form of knowledge to provide. Pedagogically, this can be viewed as providing broader knowledge about software engineering activities than can usually be provided in the classroom, or through practical exercises.

A rather different way of presenting knowledge that has been derived from *experience* is to provide a list of *factors* that should be considered when undertaking some task or adopting a technique. A good example is provided by [84], where the authors identify the factors that can make for the effective adoption of global software development practices. This type of knowledge can provide more directly useable guidance, possibly in the form of checklists, and hence can usefully be used to supplement classroom teaching about a given topic.

The third way to present knowledge is largely concerned with providing guidance about *choices* between different techniques. Such knowledge is more quantitative in its nature, and may well be involve *ranking* the different options in some way. A good example of using such a form is provided in the review by Dieste and Juristo [27] that assesses the effectiveness of different requirements elicitation techniques. From a pedagogical perspective, where the findings are organised in this form, they can be used to provide an authoritative basis for choosing to use particular practices.

The usefulness of any systematic review is also dependent upon the *provenance* for its findings—that is, how far we can be confident that the original primary studies are reliable and relevant. One reason for systematic reviewers to perform a quality check on the primary studies when performing a systematic review is to help make some assessment of their reliability, in order to inform the process of synthesis. If they conclude that a primary study was conducted well, this provides some reassurance that including its findings in a synthesis procedure will help with producing sound findings from that process. This in turn provides scope for assessing the *strength of evidence* supporting the findings from a review [28].

The issue of the *relevance* of the findings from the primary studies used in a review is more challenging. In [42] this is defined as the "potential impact the research has on both academia and industry", and the authors observe that the long maturation period for technology makes it "infeasible to use the actual uptake of research results by industry" as an evaluation tool. They propose an evaluation model for relevance that is based upon "potential for impact" and that uses four aspects: subjects; context; scale; and research method. Unfortunately, the reports of systematic reviews rarely provide much detail about these characteristics of the primary studies, and particularly about their context and scale, so we were not able to apply this model retrospectively in our analysis.

From the perspective of the teacher wishing to use the findings as supplemental material, the first aspect should require little more than explaining to students about the nature of a systematic review, in order for them to understand the nature of the evidence. An appreciation of the second (and to some extent the third) aspect may require a rather fuller explanation of the forms and limitations of empirical software engineering studies. However, since few software engineering systematic reviews provide any information about the strength of evidence, our own experiences suggest that this explanation need not be particularly detailed or extensive.

3.2. Categorising software engineering knowledge

Categorising and organising software engineering knowledge has been the goal of a number of initiatives. ACM and IEEE have jointly sponsored two that are relevant to this paper.

- The *Software Engineering Body of Knowledge* (SWEBoK) [14].
- The software engineering undergraduate curriculum guidelines (SE2014), and within this, the Software Engineering Education Knowledge (SEEK) categorisation of relevant knowledge [5].

The first of these is largely concerned with identifying the topics that collectively comprise the activities that make up software engineering practices, and where possible, identifying good sources of material related to these. So it can be considered to provide an expert interpretation of the nature of software engineering itself.

The second is concerned with identifying what an undergraduate studying software engineering should know, and hence the SEEK has been used in this paper to categorise the systematic reviews identified within the tertiary study. Even where a student is not studying software engineering as the major element of a degree, these are still topics that they need to be aware of, although perhaps in less detail than would be appropriate for a specialist course.

Table 1. Knowledge Areas used to categorise the SEEK

Knowledge Area	Key
Computing Essentials	CMP
Mathematical & Engineering Fundamentals	FND
Professional Practice	PRF
Software Modelling & Analysis	MAA
Requirements Analysis & Specification	REQ
Software Design	DES
Software Verification & Validation	VAV
Software Process	PRO
Software Quality	QUA
Security	SEC

As a framework, while the SEEK can appear to be organised around technology issues rather than 'social' issues, this impression is misleading. Table 1 lists the major *Knowledge Areas* (KAs) used to structure the 2014 version of the SEEK. Each Knowledge Area is organised as a set of *Knowledge Units* (KUs) and both in these and in the guidelines there is quite extensive emphasis upon the importance of more 'social' aspects of software engineering such as the human interactions that occur in agile development and groupwork. Also, as emphasised in the Curriculum Guidelines, the Knowledge Areas are not meant to be templates for modules.

4. Research Method

In order to answer the question posed in the title of this paper, we divided this into two separate, but linked, research questions, as follows.

RQ1: *Which systematic reviews published up to the end of 2015 produced findings that were relevant to teaching about practice in software engineering?*

RQ2: *What guidance did each systematic review provide that could help a student (or practitioner) to understand how to make an effective choice or use of a technology or practice?*

To answer RQ1, we conducted a systematic mapping study of published systematic reviews (a tertiary study). We then used the Knowledge Areas from the SEEK to categorise those that were selected as being relevant. To answer RQ2, we analysed the outcomes from each of the systematic reviews that we included, in order to identify relevant findings and explicit recommendations. As a point of clarification regarding RQ2, we did expect that for students, the process of understanding this guidance was something that would usually be mediated by a teacher. Indeed, for both students and practitioners, we expected that the findings of a review would mainly provide 'help' by identifying those circumstances where a technique or practice might be most effectively employed (or where it would be inappropriate to employ it).

In the rest of this section, we explain our choices for the procedures required to answer these two questions, and then the following section describes how these procedures were implemented.

4.1. Scope of the study

For a systematic review, the aim should be to find *all* of the primary studies that can provide findings relevant to the topic of the review, in order to avoid bias. Because a mapping study has the purpose of creating a 'map' of the knowledge available about a topic, rather than synthesising its inputs, it does not usually need to be quite as comprehensive. Our aim, as posed in the title and RQ1, is concerned with establishing whether teaching about practice *could* be supported by the findings of systematic reviews. We therefore considered that our question could be answered from a suitably large sample of systematic reviews.

We also restricted the scope of our study to those systematic reviews for which the findings were published in journals. The page constraints of conference proceedings often means that reports of systematic reviews have to omit important details. Additionally, while many systematic reviews are first reported in conference proceedings, it is quite common for a later and fuller version to also be published as a journal paper. Since we were concerned with finding those systematic reviews that were reported in sufficient detail to be of use in making decisions and choices, we felt that it was appropriate to constrain our study to reviews published in journals. It was also considered that this would make our final selection more readily accessible for teachers, students and practitioners.

For the period to the end of 2009, we selected the journal papers from three existing 'broad' tertiary studies to form our set of candidate systematic reviews [48, 52, 24]. These studies used a mix of manual and electronic searching to achieve a comprehensive degree of coverage for that period. As no equivalent sources were available for the period January 2010 to end 2015 and the number of published systematic reviews was rapidly increasing, we searched five major software engineering journals for those systematic reviews published in this later period. These were IEEE Transactions on Software Engineering, Empirical Software Engineering, Information & Software Technology, Journal of Systems & Software, and Software Practice & Experience.

Our choice of these journals was made on the basis that these were major publishers of systematic reviews addressing software engineering practices. One of the journals (*Information & Software Technology*) also had a special section for systematic reviews.

4.2. The inclusion/exclusion criteria

We required that any reviews included in our study should address a topic relevant to practice (rather than research) and that these topics should also be relevant to 'introductory'—as opposed to 'advanced postgraduate'—teaching. (Since our model for this was based on the SEEK, it could be considered to cover anything that would be material for an undergraduate degree programme.) In addition the review needed to provide some knowledge about practice that was explicitly supported by a synthesis of the findings of the primary studies. The resulting inclusion/exclusion criteria for the study are described in Table 2.

Table 2. The Inclusion and exclusion criteria adopted for this study

	Criteria
Inc-1	The paper is published in a journal and either included in the three broad tertiary studies or in one of the five journals (depending on publication date).
Inc-2	The topic of the paper is related to practice and is considered appropriate for use with introductory teaching of SE, as defined by the SEEK.
Inc-3	The paper contains findings and/or recommendations that are explicitly supported by the reviewers' analysis.
Excl-1	Systematic reviews addressing research trends.
Excl-2	Systematic reviews addressing issues related to research methodology.
Excl-3	Mapping studies with no synthesis of data.
Excl-4	Systematic reviews that address topics not considered to be relevant for introductory teaching of SE.

To be included, a systematic review needed to meet all of the inclusion criteria, while it could be excluded if it met any of the exclusion criteria. Using the SEEK gave us a reasonably clear measure of the set of topics that we considered appropriate for answering our research question. In particular, even where a review might meet all of the inclusion criteria, if we considered its topic as inappropriate for an introductory course, it could still be excluded (Excl-4). Typically such reviews were on relatively advanced topics that combined different aspects of software engineering, such as "security in process-aware information systems" [56].

4.3. Searching for systematic reviews

Our decisions about scope, as described above, meant that the searching process was relatively straightforward. The set of 120 papers from the three broad tertiary studies were listed in the reports, and for the journals we employed a manual search of index sections. We complemented the manual search by using an electronic search to check for any systematic reviews that might have been missed (not all systematic reviews have titles that explicitly identify them as being reviews).

4.4. Quality Assessment

Quality assessment of systematic reviews is commonly performed by using the DARE criteria (Database of Attributes of Reviews)[1] that were originally devised for use in clinical medicine. In its current form, the DARE assessment is based upon the following five questions.

1. Are the review's inclusion and exclusion criteria described and appropriate?
2. Is the literature search likely to have covered all relevant studies?
3. Did the reviewers assess the quality/validity of the included studies?
4. Were basic data/studies adequately described?
5. Were the included studies synthesised?

For this study we adopted the use of DARE as a means of providing an assessment of how thoroughly each systematic review had been performed, and hence some indication of how reliable the findings from it might be. (This was also employed in the three broad tertiary studies.) In doing so we also adopted the widely-used convention of scoring each question using a three-point scale: yes (1); partly (0.5); no (0), with the maximum score then being 5.0. Table 3 explains how the scoring was interpreted for the DARE criteria in the case of this study.

For each of the DARE questions, a score of 'no' was awarded where there was an absence of information (apart from 'search coverage' where we had defined a lower bound). Likewise, a 'yes' indicated that the description or related operations for that criterion exceeded some threshold. A score of 'partly' indicated that, while something was provided, it might only be for some of the primary studies (say), or that it was provided in some aggregate form. Hence a rating of 'partly' could be interpreted as 'present but incomplete'.

We should also note that DARE is only concerned with the systematic review *process* and whether these activities have been performed, rather than how well they have been done. However, until we have better reporting of systematic reviews performed in software engineering, it does not seem practical to employ some of the other forms of assessment discussed in [28] and [53].

4.5. Data extraction

Our inclusion criteria, as summarised in Table 2, required that we should be able to identify *findings* and *recommendations* for any systematic review that was to be included. This stemmed

[1] http://www.crd.york.ac.uk/CRDWeb/AboutPage.asp.

Table 3. Interpretation of the DARE Criteria used for the tertiary study

Criterion	**Score**	**Interpretation**
Inclusion & exclusion	yes	The criteria used are explicitly defined in the paper.
	partly	The inclusion/exclusion criteria are implicit.
	no	The criteria are not defined and cannot be readily inferred.
Search coverage	yes	The authors have searched four or more digital libraries and included additional search strategies OR identified and referenced all journals addressing the topic of interest.
	partly	Searched three or four digital libraries with no extra search strategies OR searched a defined but restricted set of journals and conference proceedings.
	no	Searched up to two digital libraries or an extremely restricted set of journals.
Assessment of quality	yes	The authors have explicitly defined quality criteria and extracted them from each primary study.
	partly	The research question involved quality issues that are addressed by the study.
	no	No explicit quality assessment of individual papers has been attempted.
Study description	yes	Detailed information is presented about each study.
	partly	Only summary information is presented about the studies.
	no	Details of the studies are not provided.
Synthesis of studies	yes	The authors have performed a meta-analysis or used another form of synthesis for all the data of the study.
	partly	Synthesis has been performed for some of the data from some of the primary studies.
	no	No explicit synthesis has been performed (as in a mapping study).

from a concern that, to be of use, a study had to present results that end-users could readily employ. For the purpose of data extraction, we used the following descriptions.

- A *finding* provides knowledge about the topic that an end-user might find useful in order to gain knowledge about the topic [2]. However it is not of such a nature, or accompanied by such a degree of confidence, as to be able to act as the source of explicit advice about good or undesirable practice related to that topic.
- A *recommendation* provides an operationalisation of a finding that provides deeper understanding and that can be taken into consideration when making decisions about practice. So if possible, a recommendation should be accompanied by some measure of its *strength*, derived from the evidence available from the systematic review.

Because the presence of these could only be determined with certainty at the stage of data extraction, we accepted that some decisions about exclusion would occur during data extraction. The data extracted from each study is itemised in Table 4.

[2] In [18] we used the term ‘conclusion’ rather than ‘finding’. Upon reflection, we felt that this could be ambiguous in this context, and so have adopted the use of ‘finding’ in this paper.

Table 4. Core data extraction from systematic reviews

Item	Description
1.	Bibliographic information (title, authors, publication details).
2.	Our scores for the DARE criteria. (As interpreted in Table 3).
3.	Data about any quality assessment performed in the systematic review for the primary studies, including details about any checklist used for this.
4.	Details of how the quality scores from Item 3 were actually used in the systematic review.
5.	The size and nature of the *body of evidence* used in the review (numbers and types of primary study).
6.	The *context* relating to the body of evidence: details of participant types, period covered by the searching, search engines used, details of any manual searches, use of snowballing, number of studies retained at each stage of inclusion/exclusion.
7.	Any *findings* that are reported, or that could be derived from the later sections of the paper.
8.	Any *recommendations* reported or that could be derived.

5. Conduct of the Study

The study was conducted according to the plan, and this section provides some details about the procedures followed as well as the outcomes.

Because there was some overlap between the set of systematic reviews selected for our earlier study [20], for brevity when comparing the two studies, we refer to that study as EPTS1 (Education and Practice Tertiary Study 1), and refer to this study as EPTS2.

5.1. Study Identification

As the set of papers found by the three broad tertiary studies was already determined, searching was only necessary for the papers from the five journals published in the period 2010-2015. The manual search process was conducted by one of the authors (DB) and involved reading through the contents pages of the five journals examining titles of papers, and where necessary, also inspecting the abstracts.

To complement the manual search, an electronic search was also performed by an independent researcher. This was undertaken in two stages. In the first of these, covering the period 2010-2014, the *Scopus* digital library was used to perform a forward citation analysis of six papers that discussed the principles of EBSE and systematic reviews (listed in Table 5). This was performed in April 2016. The papers identified as being systematic reviews or mapping studies and published in the five journals were compared with the papers that had been found by manual search. However, this identified a large number of false positives, and for papers published in 2015, this problem became much greater. So for the second stage (period) for 2015, Scopus was searched using the terms: TITLE-ABS-KEY ("systematic literature review" OR "systematic review" OR "systematic mapping study" OR "mapping study") AND DOCTYPE (ar OR re) AND PUBYEAR = 2015 AND (LIMIT_TO (SUBJAREA, "COMP")). The results from this were sub-setted to select studies from each of the five journals and the papers that were identified as being mapping studies and systematic reviews were compared with the papers found by the manual search. This second search took place in May 2016.

The manual search identified 140 papers and the electronic search added a further 16, giving a total of 156 systematic reviews from searching the journals. All studies were allocated an index

Table 5. Papers used for forward citation analysis in Scopus

Title	**Reference**
Evidence-Based Software Engineering	[51]
Evidence-Based Software Engineering for Practitioners	[29]
Procedures for Undertaking Systematic Reviews	[47]
Guidelines for Performing Systematic Literature Reviews in Software Engineering	[50]
Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain	[15]
Systematic Review in Software Engineering	[12]

number, those from the broad tertiary studies being numbered #1–120, and those from the journals #121–276.

Our sources are described in Table 6. For ease of reference, we have labelled these as *Source-set1* and *Source-set2*. We have also indicated the number of papers obtained from each of these sources.

Table 6. Details of the sources used

Period	**Sources**	Count
2004-2009 (Source-set1)	TS1: Tertiary Study 1 [48]	20
	TS2: Tertiary Study 2 [52]	33
	TS3: Tertiary Study 3 [24]	67
		120
2010-2015 (Source-set2)	IEEE Transactions on S/W Eng.	13
	Empirical Software Engineering	10
	Information & Software Technology	97
	Journal of Systems & Software	31
	Software Practice & Experience	5
		156

5.2. The Inclusion-Exclusion process

The process of inclusion/exclusion was performed in two stages. This was because the relevance of the topic could be fairly easily determined from the title and abstract, whereas determining the availability of appropriate findings and recommendations (Inc-3) did require that the complete paper had to be read.

In the first stage the two criteria used were whether or not a study was a systematic review, published in a journal, that addressed a potentially relevant topic (Inc-1 & Inc-2). The studies that had earlier been included in EPTS1, published in the period up to mid-2011 and described in [20], had already been identified as meeting the second criterion, and so the only action required was to remove those published in conferences. Hence a full selection process was only performed for the studies with index values #146-276, which were those published from mid-2011 onwards and hence had not been used in EPTS1. This was performed by all four authors, working in different pairings that were allocated on a random basis. The only exceptions were the papers for which two of us (DB and PB) were authors, which were assessed by the other two reviewers. If the reviewers were unable to agree on exclusion of a paper, it was retained for the second stage. Using the Fleiss' Kappa [31] to assess the level of rater agreement for this first stage, as we were using multiple raters, produced a score of 0.490, which indicates *moderate* agreement, falling into the band of values usually considered as being acceptable ("fair to good") [8].

The second stage was combined with the process of data extraction, which was based upon the data extraction model described in Table 4. This was applied to all of the reviews identified from the first stage, and all data extractions were performed by two members of the team, working independently, who then resolved any differences to produce an agreed dataset for a review.

Studies were only retained at this stage if we could identify clear findings and/or recommendations that could be linked to the data extracted as part of the systematic review (criterion Inc-3). Some papers originally included in EPTS1 were excluded in this stage, on the basis that they had inadequate findings or recommendations. Figure 1 provides an overview of the overall process and resulting numbers. We have referred to the reviews selected from Source-set1 as *Dataset1*, and those from Source-set2 as *Dataset2*.

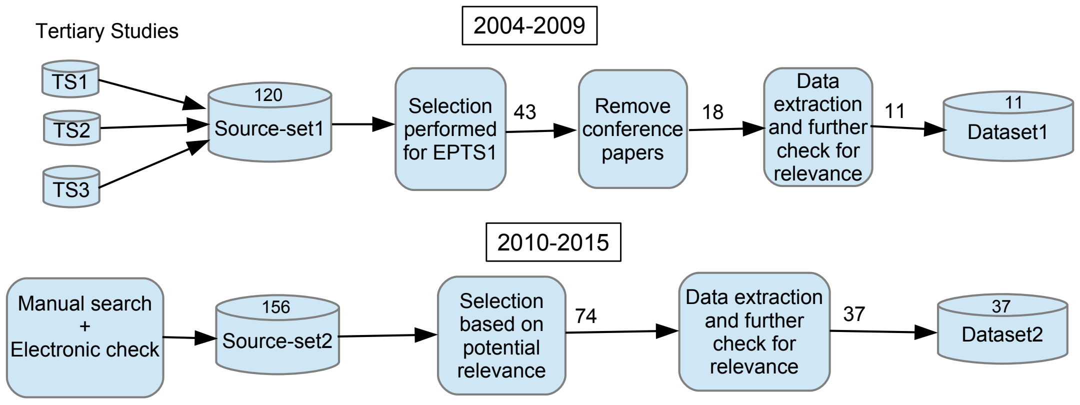

Tertiary Studies
2004-2009
TS1
TS2
TS3
120
Source-set1
Selection performed for EPTS1
43
Remove conference papers
18
Data extraction and further check for relevance
11
11
Dataset1
2010-2015
Manual search + Electronic check
156
Source-set2
Selection based on potential relevance
74
Data extraction and further check for relevance
37
37
Dataset2

Figure 1. The overall selection process (TS1–TS3 are the three tertiary studies)

Because identification of the findings and recommendations from the systematic reviews was often a complex process (elements of these were apt to be spread around the final sections of a paper), we performed a further check upon the reliability of our interpretation. For each systematic review we tried to contact the designated corresponding author by e-mail and asked them to comment on our interpretation of the outcomes.. Where this was no longer a valid address, we then tried contacting any of the authors for whom we could find a suitable e-mail address. In only two cases were we unable to trace any of the authors. We received 27 responses, all of which were generally in agreement with our interpretation, with 16 of them suggesting changes of wording, with all of these being minor.

The final set used in this study (EPTS2) contained 49 reports of systematic reviews, listed in Appendix A. Because the data for one systematic review was used for two analyses (#54 and #118), both of which met the inclusion criteria, there were actually 48 sets of primary studies.

5.3. Quality Assessment

An assessment against the DARE criteria, using the interpretations provided in Table 3 was performed as part of the process of data extraction, and using the same randomly-allocated pairings of reviewers.

5.4. Categorisation against the SEEK

To categorise the systematic reviews against the SEEK two of the reviewers (DB and PB) performed another analysis of the reviews after all of the data extractions had been completed. Again, our argument for doing this as a separate analysis, as against performing it as part of data extraction, was largely a matter of ensuring greater consistency of interpretation. It was considered that this would be more easily achieved if the whole set of studies was categorised in a single process.

For each study we determined both the most appropriate *Knowledge Area* (summarised in Table 1) and also what we considered to be a suitable assignment to the more detailed *Knowledge Unit* within this. While for some studies the most appropriate KA and KU values were relatively obvious, many did require quite extensive discussion to determine an appropriate allocation as inevitably, the topic of a systematic review and its findings may well span more than one KA or KU. Indeed, the nature of the findings may be more important than the topic of the review in terms of determining how it should most appropriately be categorised.

5.5. Further Data Extraction

As noted in Section 1, we have performed further analyses of the 49 systematic reviews. These are reported in [18] and [19] and involved some additional data extractions. These were performed by two of us (DB and PB) and are summarised in Table 7. Some of this supplemental information is included in Appendix B. We should note that for both of these, the process of study selection was as reported here. So, while they investigated further questions about reporting and provenance, their analyses were limited to providing answers related to studies about software engineering practice.

Table 7. Additional data extraction from systematic reviews

Item	Description
9.	Whether and how any quality scores derived for the primary studies were used (if at all).
10.	The form(s) of synthesis used in the study, and whether these classifications were made by the authors of the systematic review or by us. Categories used were: meta-analysis, narrative synthesis, meta-ethnography, grounded theory, cross-case analysis, thematic analysis, vote counting, and 'other'. The definitions of these were taken from [23].
11.	The forms of primary studies used, where the primary studies were performed; who conducted these, and who formed the participants (students or industry practitioners) or what sources of data were used (industry or artificial).

6. The findings—What knowledge is available?

To present the outcomes from the process described in the previous sections, we begin by providing an overview of all of the studies. We also look at some of the supplementary information about these, with particularly regard to such aspects as *provenance*. We then look at the studies in more detail, and in particular, present the *findings* and *recommendations* that were extracted for each one. These are grouped under the different SEEK headings, enabling us to also comment on the extent of the available knowledge for each heading.

6.1. Summary of the Systematic Reviews

As the total number of studies is quite large, we have presented the summary of the findings for the two datasets separately in Tables 8 and 9. This is largely a convenience for presentation, although it also helps distinguish the reviews that were undertaken when the practices for systematic reviews in software engineering were less well established. Both are described using the same format. Each entry is described in terms of its index number (#1–#276) as used in this study, the period covered by the search in the systematic review, its topic, and reference. The tables also provide some of the key information about each review: the SEEK Knowledge Area (KA) it has been assigned to (the keys we use were provided in Table 1); the DARE score we derived for the study; the number of primary studies that we could identify as being either explicitly or implicitly conducted in an industry setting or making use of industry participants; and the total number of primary studies.

Table 8. Details of the systematic reviews included in this study: Dataset1 (2004-2009)

No.	Period covered	Topic & Citation	SEEK KA	DARE	Primary Studies	
					ind.	total
52	unclear	Motivations for adopting CMM-based SPI [85]	FND	2.5	49	49
54	1980-6/2006	Motivation in software engineering [11]	PRF	5.0		79
118	(as 54)	Models of motivation [81]	PRF	(as 54)		(79)
15	1992-2002	Capture-recapture in s/w inspections [72]	VAV	1.5	1	25
66	1996-2007	Search-based non-functional testing [7]	VAV	4.5	17	35
82	1969-2006	Regression test selection techniques [30]	VAV	4.5	4	36
8	to 2006	Estimation of s/w development work effort [45]	PRO	1.0	14	16
22	unclear	Assessment of development cost uncertainty [44]	PRO	2.5		40
39	1994-2005	Benefits of software reuse [62]	PRO	3.5	11	11
50	1996-3/2006	SPI in small & medium s/w enterprises [74]	PRO	4.0	45	45
84	to 2007	Effectiveness of pair programming [38]	PRO	4.0	5	19
102	1995-2005	Managing risks in distributed s/w projects [70]	PRO	2.5	72	72

The issue of the provenance of the primary studies is discussed in more detail in [19], where we have categorised the context for the primary studies used in each systematic review as far as we were able, based upon the available information. Two key points from this are worth repeating here. The first is that for those systematic reviews where we could not determine whether some of the primary studies were explicitly or implicitly conducted in an industrial setting, it is highly probable that many of these were actually industry-related, but the lack of detail meant that we simply could not tell. The second point is that what was considered to be an acceptable primary study in terms of the inclusion/exclusion criteria used in the review, and the way that these were interpreted, did vary quite considerably. Some reviews included a number of non-empirical reports among the primary studies, as well as papers that were classified as 'opinion', 'experience' and even 'theory'. So while other characteristics such as DARE scores might usefully be compared across a set of systematic reviews, it is definitely not appropriate to make comparisons between the numbers for each type of primary study as reported by different systematic reviews.

The answer to RQ1 (which systematic reviews produced findings relevant to teaching about practice?) is provided by the entries in Tables 8 and 9. Overall, as indicated, we were able to identify 49 systematic reviews (from 276) that contained findings considered to be of use in teaching about software engineering. In the tables, the systematic reviews have been grouped under the SEEK Knowledge Areas, which also highlights the uneven distribution of reviews across the KAs. Table 10 gives the counts of the reviews categorised under each KA. The large proportion categorised as PRO arises in part because much of what we do in software engineering involves processes. Many of the systematic reviews can be described as investigating 'best practice', where this may relate to testing, design etc., and these ended up being categorised as PRO wherever we concluded that the emphasis was more upon practice rather than the technology involved.

Table 9. Details of the systematic reviews included in this study: Dataset2 (2010-2015)

No.	Period covered	Topic & Citation	SEEK KA	DARE	Primary Studies	
					ind.	total
135	1980-2008	Antecedents to personnel's intention to leave [32]	PRF	3.0	72	72
246	2003-4/13	Newcomers on OSS projects [86]	PRF	3.5	20	20
167	2006-2011	Evaluating cloud services [57]	VAV	4.0	82	82
197	to 10/2011	Software fault prediction metrics [76]	VAV	4.5	81	106
205	2000-2011	Test-Driven Development [63]	VAV	4.5	22	41
252	2002-2013	Metrics in Agile/Lean development. [54]	VAV	3.5	30	30
124	1970-2007	Characterising s/w architecture changes [91]	DES	3.5		130
130	1997-2008	Aspect-oriented programming [3]	DES	4.5	6	22
154	1995-2009	Software design patterns [93]	DES	2.0	11	18
123	unclear	Domain analysis tools [58]	MAA	3.5	7	19
126	1989-2006	Does the TAM predict actual use? [89]	MAA	5.0		79
146	2000-2010	Dependency analysis solutions [6]	MAA	2.5	38	65
155	2000-2010	Fault prediction performance [37]	MAA	4.5	35	36
134	to 3/2005	Elicitation techniques [27]	REQ	5.0	7	32
161	1993-2011	Stakeholders for requirements elicitation [68]	REQ	4.5	42	42
259	1992-2/14	Use case specifications research [88]	REQ	4.0	27	119
219	to 2012	OO measures and quality [43]	QUA	4.5	33	99
121	2000-2007	Global software engineering [84]	PRO	3.0	37	56
138	to 2009	Measuring & predicting software productivity [71]	PRO	4.5	25	38
150	to 6/2010	Agile product line engineering [26]	PRO	3.5	14	39
157	to 2/2011	Test-Driven Development [77]	PRO	4.0	10	37
160	to 4/2009	Reconciling software development methods [59]	PRO	2.5	42	42
174	unclear	Industrial use of software process simulation [4]	PRO	3.5	87	87
175	to mid-2008	Selecting outsourcing vendors [46]	PRO	3.5	77	77
193	to 7/2010	Social software for global software dev. [33]	PRO	4.0	61	84
215	to 12/2013	Software development in start-ups [69]	PRO	4.5	30	43
217	1997-2011	Influence of user participation [1]	PRO	3.5	82	82
222	1990-2012	The Kanban approach [2]	PRO	4.0	37	37
228	1997-1/08	Software process assessment [92]	PRO	2.5	22	22
236	2001-2013	Global team dispersion [66]	PRO	4.5	40	43
239	to 2011	Using CMMI with Agile [82]	PRO	4.5	59	60
241	1980-2012	User-involvement and success [9]	PRO	4.5	87	87
244	1990-2012	Development effort estimation [41]	PRO	5.0	61	61
249	2002-10/12	User-centred agile development [16]	PRO	4.5	26	83
260	to 5/15	Use of SE practices in science [39]	PRO	2.5		43
268	1996-2/08	Product derivation support [75]	PRO	2.0		118
276	1996-10/13	Adopting SPL [90]	PRO	3.0	31	31

The answer to RQ2 (what guidance did each systematic review provide?) is contained in the fuller descriptions of the findings and recommendations, together with their context, provided in Appendix B. As discussed earlier, we observed that systematic reviews provide guidance in a number of forms, largely depending upon the research question being addressed by the review. In Table 11 we identify those reviews providing each of the three types of guidance (experience, lists of factors, and comparisons). Inevitably, the findings of reviews do not always fall exactly into one of these categories, and so we have only included the 35 systematic reviews where we collectively felt that the findings mainly fitted one category. What Table 11 does show though is that few reviews provided comparative findings. VAV was the only KA for which there was more than one systematic review (3 from 5) providing comparative findings, largely because these were comparing testing practices

Table 10. Counts of reviews for each Knowledge Area

KA	**Topic**	**Dataset 1** (up to 2009)	**Dataset2** (2010-2015)	**Total**
FND	Fundamentals	1	-	1
PRF	Professional Practice	2	2	4
VAV	Verification & Validation	3	4	7
DES	Design	-	3	3
MAA	Modelling & Analysis	-	4	4
REQ	Requirements Analysis/Specification	-	3	3
QUA	Software Quality	-	1	1
PRO	Software Process	6	20	26
Totals		12	37	49

that produced deterministic outcomes concerned with whether or not a test was successful. (Many aspects of software engineering, such as analysis and design, address 'ill-structured' problems [83], and so rarely provide true-false results when comparisons are being made.)

Table 11. Forms of guidance provided by reviews

Type	**Systematic Reviews**							
	FND	PRF	VAV	DES	MAA	REQ	QUA	PRO
Experience	–	–	#167, #252	#124, #154	#123, #146	#259	–	#39, #50, #174, #193, #222, #239, #260, #102
Lists of Factors	#52	#54, #118, #135	–	–	#155	–	–	#160, #276, #84, #215, #236, #241, #121, #244
Comparisons	–	–	#15, #66, #197	#130	–	#134	#219	#8

6.2. The Findings & Recommendations for each Review

In this subsection we present the material that helps answer RQ2 (guidance "that could help a student (or practitioner) to understand how to make an effective choice or use of a technology or practice"). For each review, we excluded any findings that were related to research issues or future developments. (Almost every systematic review identifies a need for more and better primary studies.) Where possible, we have taken the wording for the findings and recommendations directly from the systematic reviews. One consequence of this is that the findings from different systematic reviews are apt to be formulated at different levels of granularity. However, given the heterogeneity of the reviews, we considered that it was impractical to present the findings in a uniform matter.

In reporting the findings, we have also provided information about their *provenance*, wherever this was available. This information is provided to aid the reader to make some assessment of the confidence that they might choose to place in the findings. However, as noted earlier, the variation in different reviews between the way that primary study types were interpreted, as well as in the inclusion/exclusion criteria used, means that this information should only be treated as indicative.

The details for each systematic review are provided in Appendix B. For each review we provide the following information (where available)

1. The main SEEK knowledge area and knowledge unit identified as appropriate.

2. The title of the systematic review.
3. Citation details.
4. The DARE score, reported on a scale of 0–5.
5. Any information available that might provide an assessment of the *strength of evidence* for the findings. Where possible, we report this for each finding.
6. The number of *primary studies*. Where possible, we included the following additional information:
   – The count of primary studies that we could *explicitly* identify as being conducted in an industry setting.
   – The count of primary studies that were *implicitly* conducted in an industry setting, based upon comments in the text.
   – The count of primary studies conducted in an 'academic' setting (such as experiments that used student participants).
7. The form(s) of *synthesis* used in the study, noting that some did use more than one form to answer different research questions. (We did not attempt to classify the forms of synthesis used in the earlier studies (*Dataset1*).)
8. The *findings* from the study.
9. The *recommendations* from the study.
10. Information about any response from the authors to our request for them to check the accuracy of our extraction of the findings and recommendations.

We have grouped the reviews according to their assignment to SEEK Knowledge Areas. For each review, we suggest the most relevant Knowledge Area and Knowledge Unit, accepting that many reviews do not fit neatly into the SEEK model. We have also noted where there are Knowledge Units (other than those dealing with issues such as 'concepts') for which there are no systematic reviews, in order to help illustrate the overall degree of coverage.

6.2.1. Findings—Fundamentals (FND)

Perhaps not surprisingly, there is only one systematic review categorised under this heading. The key details for this are provided in Table 13. The reason for including this review under FND was that we felt it best fitted the Knowledge Unit *engineering economics for software*. (This was the only heading for this KA that did not address 'foundations'.)

In this review, the conclusions about the reasons for adopting SPI (Software Process Improvement) largely reinforce the claims made in the literature.

6.2.2. Findings—Professional Practice (PRF)

We classified four systematic reviews under this heading, described in Tables 14, 15, 16 and 17. Two of these (#54 and #118) used the same dataset, but performed quite different analyses of the material. We were also unable to determine a specific Knowledge Unit for those two analyses, due to the wide span of issues that they address. There were no systematic reviews directly addressing the KUs *communications skills* or *professionalism*, although some other systematic reviews did indirectly address issues related to team and group communication.

The first two reviews (which share a dataset) address issues around what motivates software engineers and provide details of factors considered relevant. Study #135 is also related to staff (de)motivation, providing a set of related recommendations. The remaining study addresses the role that group dynamics plays when participating in open source development.

6.2.3. Findings—Software Verification and Validation (VAV)

There were seven reviews included under this heading. These are summarised in Tables 18–24. These reviews provide a set of findings that span three of the four Knowledge Units making up the VAV Knowledge Area. We have no reviews for one KU, *problem analysis and reporting.*

These reviews span a range of issues. Most are concerned with techniques for selecting or evaluating tests (such as those used for regression testing) and provide rankings of different approaches that are likely to be directly applicable to practice.

6.2.4. Findings—Software Design (DES)

Software engineering can be considered as very much a 'design' discipline, with 'design thinking' permeating many activities, including of course, software design. However, the creative element involved in designing also means that this Knowledge Area forms a significant challenge for empirical studies. There are only three systematic reviews in this group, summarised in Tables 25–27, although they do address three separate Knowledge Units. KUs with no contributions are *design concepts*, *human-computer interaction design* and *design evaluation.*

None of the reviews offer very strong conclusions, and in the only one that offered more specific guidance about design choices (#130), it was noted that these were based upon a low strength of evidence.

6.2.5. Findings—Modelling & Analysis (MAA)

The four systematic reviews under this heading are all classified as belonging to the same Knowledge Unit (*types of models*). Given that the other two KUs address *foundations* and *fundamentals*, this is perhaps not surprising. Tables 28–31 provide a summary of these reviews.

The reviews span a range of issues including model reliability (#126) and observations about fault prediction (#155). Collectively they do provide helpful guidance about some specific models that are used by software engineers.

6.2.6. Findings—Requirements Analysis & Specification (REQ)

The three systematic reviews addressing requirements, described in Tables 32–34, cover two of the four Knowledge Units making up the REQ Knowledge Area. In particular, we have no reviews that address the KU *requirements validation.*

All the reviews provide useful insight into the approaches used in requirements engineering. Review #134 particularly provides a useful rating of different elicitation techniques, and all three offer useful insight.

6.2.7. Findings—Software Quality (QUA)

There is only one systematic review categorised under this heading. The key details for this are provided in Table 35. This was categorised against the KU *product assurance*, and there were no reviews covering the KU *process assurance.* The review does offer useful insight into the relative merits of a range of object-oriented measures.

6.2.8. Findings—Software Process (PRO)

By far the largest set of reviews fall into this Knowledge Area (which does form something of a 'catch-all'). We have grouped these by Knowledge Unit (KU), although we should note that only three of the five knowledge units were covered. There were no reviews classified as *configuration management* (PRO.cm) or *evolution processes and activities* (PRO.evo). Tables 36 onward provide the details for this set of reviews.

With so many reviews being classified as belonging to this KA, it is difficult to provide a concise and general summary of what is useful for practice and teaching. Many of the reviews classified against *project planning and tracking* offer quite specific and detailed advice that is highly relevant to both teaching and practice. In contrast, for *process concepts* many of the findings tend to be more in the nature of observations, with *process implementation* coming somewhere between these. Overall though, this set of reviews do provide a fertile source of experience for others to draw upon.

7. Discussion

We first consider what the outcomes from our study tell us about the knowledge available from this set of systematic reviews, and what the limitations on this knowledge are. We go on to consider how this knowledge might be used to inform teaching (and hence in the longer term, practice) by looking at how such knowledge is used in other disciplines, and hence what lessons might be learned. We then consider the threats to validity for this study, since we need to determine how trustworthy our findings are, both in terms of the selection of the systematic reviews, and also the findings and recommendations that we extracted from them. Finally, we consider how such knowledge can be gathered more effectively and completely in the future, and in particular, how it might be possible to avoid having to do this retrospectively (and laboriously), as in this tertiary study.

7.1. How good is the knowledge available from systematic reviews?

In answering RQ1, we can identify 49 (out of 276) systematic reviews that provide knowledge about software engineering practice and hence might be used to support teaching about software engineering. The systematic reviews that we identified also span a range of topics when matched against the SEEK, although they are not evenly distributed between the Knowledge Areas. The extent, quality, and form of the knowledge is also unevenly distributed, with some reviews providing findings that provide quite useful information about practice, while others are rather less specific.

In addition, few reviews provided any indication of the *strength of evidence* available to support their findings from the primary studies. Examination of the 49 reviews shows that only two of them (#130 and #239) made use of the GRADE approach to assess the strength of evidence for their findings [36], as recommended in [28]. A few of the others (#008, #022, #039, #197, #215) did also make assessments through unspecified means. Where provided, such assessments tend to indicate a strength of evidence for recommendations as being 'low' (#130 and #239) or 'modest' (#008). However, as noted in the revised guidelines on conducting systematic reviews in software engineering [53], empirical software engineers "must often make do with much weaker forms of study" (than those working in other disciplines).

It is also worth noting that some of the more qualitative reviews, such as those identifying "factors relevant to the adoption of X", are unsuited to the use of an approach such as GRADE. A number of these did provide tables that listed and enumerated the primary studies that identified a particular factor as being significant, with examples of this occurring in #54, #161 and #205.

To address this question, we have identified the set of systematic reviews which we consider offer both useful and usable guidance about practice. To select these, each of the authors was asked to rate each review, using the information presented in Appendix B, and assigning one of the following values to it.

'y' if the review was one that *could* be readily used as an example when teaching;
'p' for reviews that *might* be used;
'x' if the review should *not* be used as an example.

In performing the rating, each author was asked to consider the following three factors.

1. The *usefulness* of the review: such that its outcomes relate to a reasonably 'mainstream' topic that might be included in an introductory course on software engineering.
2. The *usability* of the review's findings, whereby these can provide some element of guidance about what a software engineer might be advised to do in practice.
3. The *quality* of the review, largely based upon the DARE score. It was suggested that a score of ≥ 3.5 would be acceptable, while also bearing in mind that earlier reviews often had less conventional reporting structures.

Each review was considered on its own merit, and there was no constraint upon how many reviews could be given a particular rating. (The number of 'y' ratings employed ranged between 10 and 17.)

Since our teaching experiences stemmed from teaching different courses and we had different interests within software engineering, we did not expect to obtain close agreement from this process. So a 'score' for each review was computed by assigning each 'y' to a value of 1.0, a 'p' as 0.5, and an 'x' as 0.0, and then summing the four values.

Table 12 shows the index values for the top-scoring reviews that emerged from this process. We have also indicated the type of knowledge provided by these reviews, where a single value was available, and provided a summary of their findings together with a reference to the Table in the appendix where further details can be found. While not too much weight should be placed upon this relatively informal exercise, it is interesting to note the predominance of reviews categorised as VAV and PRO, as well as of reviews with more 'structured' findings in the form of lists or comparisons. It does also indicate that, while all 49 reviews were considered relevant enough to be included in the tertiary study, few of them achieved this quite basic quality threshold for the three criteria, with 'good' studies available for only a few Knowledge Areas.

So, to answer the question posed in the heading for the sub-section (and RQ2), we can conclude that while suitable evidence-based material is becoming available for use by teachers, only a rather disappointingly small proportion of systematic reviews appear to have findings that can readily be used.

However, there is one quite important caveat that should be mentioned here. In the above exercise we only considered *direct* use of this material in teaching on introductory courses, enhancing what is covered in the textbooks. There are however other ways of using this knowledge, such as in course design (for example, using the findings on the unsuitability of design patterns for use by novices to determine how this topic would be covered in a course). There is also scope to use the findings differently on more advanced courses, including postgraduate ones, or with individual student projects. All of the 49 reviews are viewed as having findings that are *potentially* useful, but these may need to be used in different ways. We address this issue further in the next sub-section.

Table 12. Findings considered most useful and usable

Score	Review	KA.KU	Table	Knowledge	Summary
4.0	#134	REQ.er	A.32	comparison	Examines knowledge elicitation techniques. Unstructured interviews generally perform as well as or better than other forms such as introspective techniques when considering effectiveness, efficiency and completeness.
	#236	PRO.imp	A.52	list of factors	Assesses impact of global dispersion for development process and product quality. Lists key effects and recommends issues to consider for such projects.
3.5	#197	VAV.fnd	A.22	comparison	Assesses different metrics for their usefulness in fault prediction. Recommendations relate to project characteristics.
	#84	PRO.imp	A.46	list of factors	Provides guidance on when to employ pair programming.
	#241	PRO.imp	A.54	list of factors	Analyses how far system success is related to user involvement in development and the forms that this takes.
3.0	#82	VAV.tst	A.20		Assesses the effectiveness of test selection techniques for regression testing.
	#205	VAV.fnd	A.23		Examines studies of test-driven development (TDD).
	#39	PRO.con	A.36	experience	Identifies the benefits of software reuse based upon its use in industrial studies.
	#217	PRO.imp	A.50		Looks at the consequences of user participation & involvement (UPI) in terms of project success in industry projects.

7.2. Using the findings to support teaching and practice

Having identified a set of systematic reviews that contain knowledge that is useful for teaching and practice, this raises the question of how to *use* this material? To help answer this we looked at how other disciplines make use of such material.

Studies in education and healthcare have investigated how students and practitioners understand and engage with the findings from empirical research. This is relevant for software engineering, since using the material from this study would require familiarity with evaluation practices and empirical studies.

In education, a rapid evidence review investigated what is known about effective approaches to school and teacher engagement with evidence [65]. The report points out that *knowledge mobilisation*, the process of making research findings more accessible and usable requires a supportive infrastructure, including collaborations between researchers and teaching professionals, intermediaries to translate evidence into tools and professional bodies that provide leadership on the use of evidence in education. Also, the review suggests that evidence needs to be contextualised and presented in clear and structured summaries of effective approaches. This point is also made by Goldacre[3], who emphasises the need for better support for the dissemination of research findings,

[3] https://www.gov.uk/government/news/building-evidence-into-education

as well as by others, in relation to evidence based healthcare, where structured abstracts and plain language summaries are advocated [40, 80].

As well as learning the skills necessary to acquire, appraise and apply evidence, students and trainees can also benefit from acquiring an awareness of ways to use this knowledge to bring about change at the organisational level [60]. A discussion of this is beyond the scope of this paper, however, desirable skills might include being able to identify where changes to guidelines or to established practice are needed and where change would be worthwhile.

The importance of leadership in enhancing engagement with, and use of, research findings is also a key message from a recent study on evidence-informed teaching practice, published by the UK's Department of Education [22].

Viewed overall though, there seems to be little guidance available on how to provide advice for teachers about using empirical material such as the knowledge-set from these systematic reviews to support the way that software engineering is taught. Clearly, as such knowledge accumulates, this will present an increasingly important pedagogical research question to be pursued.

7.3. Limitations of this study

We can identify a number of limitations upon the outcomes from our tertiary study that stem from the way that we performed the various elements of the study. We discuss these here, together with any factors that may help to alleviate their effects.

1. One limitation is the way that we selected the secondary studies (*Dataset1* + *Dataset2*). Since we were performing a mapping study, we did not attempt to find all of the systematic reviews that were published during the period covered by our tertiary study, and confined ourselves to those reviews identified in the three broad tertiary studies and then the five software engineering journals, while explicitly excluding any studies published as conference papers. We did however conduct a broad electronic search as a check that we were not missing any significant source of systematic reviews, and we should observe that eight of the 11 reviews included in *Dataset1* were published in the five journals that we used in the later part of the search.
   Since we were investigating the use of systematic reviews in teaching, there was the possibility that relevant reviews could be found in educational journals related to software topics. A check of the papers published in ACM Transactions on Computer Education (TOCE) and IEEE Transactions on Education (ToE) for the period 2004-2018 inclusive, identified only five systematic reviews. All of these were addressing pedagogical knowledge rather than 'topic' knowledge and we could not identify any papers related to the use of evidence-based material in teaching.
2. In our original research protocol we selected a cut-off date for inclusion as the end of 2015. Because the processes of inclusion/exclusion and data extraction were complicated by the heterogenous nature of the selected set of systematic reviews, and as changes in circumstances also meant that two members of the team would not be available for this task, we felt that we could not ensure that any extension would be consistent with the original study, particularly regarding the interpretation for Inc-2 and Inc-3. As explained in §1 we also performed and published two other analyses on this dataset, further delaying the production of this paper.
   There is therefore the possibility that in the time following our cut-off date and submitting this paper, there may have been some changes in the way that systematic reviews have been reported, and obviously, new topics will have been covered. Informally, based upon our experiences over this period reviewing systematic reviews as well as performing some informal monitoring of journal contents, we have not observed any developments that would have significantly affected our findings. It is also possible that the balance of systematic reviews across the SEEK KAs might have changed. However, topics such as design and requirements elicitation still continue

to present some real challenges to conducting rigorous primary studies [34], limiting the scope to perform systematic reviews for those KAs.

3. When calculating the DARE score for a review, our definition in Table 3 does not address the question of whether or not the search conducted by the reviewers was *adequate* for the purpose of the systematic review. While it would be desirable to make such an assessment, we did not feel our knowledge about the research areas related to the review topics would allow us to do this in a consistent manner.
4. The selection process that we used to identify relevant systematic reviews did require an element of human interpretation, including for inclusion criterion Inc-2 (relevance to introductory teaching). We drew upon our experience of teaching software engineering to determine whether a review addressed a suitable topic, as well as using randomly allocated pairs of team members for all aspects of this part of the process, and discussing our decisions.
5. Further interpretation was required for the purpose of identifying the findings and recommendations embodied in a review (criterion Inc-3). The quality of reporting did not always assist with this [18], so as a check, we did seek to consult the original authors wherever possible. We received responses from approximately half of these, with no-one suggesting other than minor rewording or clarification, which suggests that we managed to perform this task fairly well.
6. Our supplementary data extractions were performed by two of the authors, partly to ensure consistency of interpretation. For our interpretation of the *synthesis* methods adopted in the 49 systematic reviews, we were able to check a proportion of our decisions against a baseline study [23].
7. For categorisation of the studies against the SEEK we again used two members of the team, to provide consistency in our allocations. Since many systematic reviews span different knowledge areas, this is very much an issue of interpretation, and we would certainly advise anyone seeking knowledge about a topic to check that whether it appears as an element in other studies (particularly those categorised as PRO).
8. Our informal assessment of how ‘useful’ reviews were (summarised in Table 12) used a simple ranking procedure as described in Section 7.1. However, our individual assessments, as indicated by the different numbers of ‘y’ rankings used by each assessor, were inevitably influenced both by our own teaching experience (as we note) and also possibly by our familiarity with the topics of specific reviews.
9. We were unable to obtain assessments of the strength of evidence for the findings from most of the reviews. Where an assessment was available, the findings were generally rated as being based on low or moderate strength of evidence.

Hence there may be some variation in consistency between different elements of our overall dataset, particularly as regards the confidence that we can place in the findings from each systematic review.

7.4. The way ahead?

Conducting a tertiary study of this form requires quite extensive interpretation of the reported findings from a heterogeneous set of systematic reviews. So an obvious question is whether this knowledge, assuming it is considered to be useful to the community, can be extracted from the reports of systematic reviews by other (and better) means in the future. In particular it would be better if this avoided the need to perform studies such as this one that involve retrospective analysis, both because the distance from the original study means that much of the knowledge about how it was done may not be available, and also because the original systematic reviewers can be expected to possess greater expertise about the topic of a review, as well as being better able to assess the *quality* of the evidence [34].

A relatively simple and efficacious mechanism for enabling this does exist, and is already used in other disciplines [55]. In healthcare research where the needs of policy-making may go alongside those of practice, this consists of requiring that a systematic review and its findings are reported as a set of documents with different lengths and levels of abstraction, in order to meet different needs. The Canadian Health Services Research Foundation describe this as a *1-3-25* format, consisting of: a one-page summary of 'take-home' messages; a three-page executive summary; and a more detailed report. Like any such mechanism this is not infallible of course, and as Oliver and Dickson comment: "some teams were better than others at producing a policy-friendly report" [67].

Adapting this model to the needs of software engineering appears to be quite feasible. At its most simple, it would consist of requiring, as a condition of publication, that authors also provide a one-page summary of their findings, worded in a form that made them readily accessible to practitioners and students, and including an estimate of the strength of the supporting evidence. Appendix A provides two examples of a one-page summary to illustrate this concept. The first is a summary of this tertiary study, while the second is a summary of a systematic review from our set of 49, for which one of us was an author (#154). When used for healthcare reviews, the single page often consists of a brief summary of the purpose of a study followed by a set of bullets that summarise the key findings, and we have largely adopted this model. However, for teaching purposes this may need to be supplemented by a more effective visual structure such as the one proposed for *evidence briefings* [21], and our choice of layout has also been influenced by that model.

There are obviously a number of practical issues to address in creating such a mechanism (including obtaining the cooperation of journal editors). It would require reporting guidelines for authors (we already have a set of these in [18]); a means of checking that the summary was appropriate; and (preferably) some central means of indexing the summaries. But in exchange, adopting such a system has the potential to make it likely that future reviews had findings that were translated for practice by the people most familiar with the material. Prospective reviewers would also be able to check more easily if there was an existing systematic review addressing their planned topic.

8. Reflections and Conclusions

Our tertiary mapping study identified 49 systematic reviews, published in the period 2004-2015, that contained findings and recommendations considered to be useful for teaching about software engineering (RQ1). Within these, we were able to identify a smaller number that did provide guidance and information that could be used to help make "effective choices" (RQ2).

However, it is evident that useful findings are available from only a small proportion of the published systematic reviews that we surveyed. There may be many reasons why this is so: one of which may simply be that in software engineering the role of the systematic review has so far been mainly to be used as a tool to aid research and to provide a useful training exercise and preparation for PhD students.

This underlying emphasis upon research may also explain many of the quality issues that have been identified regarding the conduct and reporting of systematic reviews in software engineering. Some may well arise because there is therefore no requirement to report to an external sponsor, others because the reviews are sometimes conducted by relatively inexperienced researchers. In contrast, other disciplines tend to use information specialists to undertake much of the work involved in searching and selecting material [13].

Following on from these conclusions, empirical researchers and others might wish to consider how researchers can better provide information about the outcomes from systematic reviews, so that this is of greater use to others. From this study, and from the other analyses we have performed upon our data, we can suggest three mechanisms that could contribute towards achieving this aim.

1. Providing better reporting of the *conduct* of a systematic review. In our analysis of reporting quality [18] we identify 12 lessons about reporting, and suggest a checklist that should be used by reviewers (and authors). Better reporting can help to establish the *provenance* for the findings from a review, and so help justify its publication.
2. Facilitating better reporting of the *findings* from a study. In part this overlaps with item 1 above, in that the reporting of a review should make its findings clear. This was only the case for fewer than one in five of the 276 systematic reviews that we examined, and even for the 49 included in the final set, we often found it difficult to extract the findings and recommendations, as these were sometimes spread over different sections of a paper. In addition, as discussed in the previous section, making the provision of a summary of findings a pre-requisite for publication will also help to make the findings more widely and readily available to others. This is clearly a concern that is also shared by other disciplines, hence the emphasis upon such mechanisms as the 1-3-25 model.
3. Creating the means to provide effective *curacy* of the knowledge about and from systematic reviews, particularly as the number of these increases. As a newcomer to the use of systematic reviews, software engineering has so far not embraced the idea of creating anything equivalent to the Cochrane and Campbell collaborations that oversee and facilitate the conduct of systematic reviews in clinical medicine and social science respectively. These bodies play a number of roles, including providing public information about relevant findings from systematic reviews.

We see the first two mechanisms as needing to be adopted in collaboration with those journals that publish systematic reviews. And all three may need the involvement of the professional bodies. What is clear from our findings though, is that without such interventions, systematic reviews in software engineering will very likely remain a tool used mainly for academic research rather than, as in other disciplines, forming a valuable (and valued) source of knowledge for software developers, teachers, and researchers.

Acknowledgements

Our thanks to Professor Barbara Kitchenham for advice and observations, as well as for conducting independent electronic searching for publications that we might have missed. We would like to thank the authors of the systematic reviews we studied, especially those who were good enough to check the accuracy of our extracted conclusions and to pass comment on these. We are also grateful to Professor Steve Higgins for informing us about current ideas about evidence-based teaching in education. We also thank the anonymous reviewers for their comments and added insight.

References

[1] U. Abelein and B. Paech. Understanding the influence of user participation and involvement on system success — a systematic mapping study. *Empirical Software Engineering*, 20:28–81, 2015.

[2] O. Al-Baik and J. Miller. The Kanban approach between agility and leanness: a systematic review. *Empirical Software Engineering*, 20:1861–1897, 2015.

[3] M. S. Ali, M. A. Babar, L. Chen, and K.-J. Stol. A systematic review of comparative evidence of aspect-oriented programming. *Information and Software Technology*, 52(9):871 – 887, 2010.

[4] N. B. Ali, K. Peterson, and C. Wohlin. A systematic literature review on the industrial use of software process simulation. *Journal of Systems & Software*, 97:65–85, 2014.

[5] M. Ardis, D. Budgen, G. W. Hislop, J. Offutt, M. Sebern, and W. Visser. SE2014: Curriculum Guidelines

for undergraduate degree programs in software engineering. *IEEE Computer*, pages 106–109, November 2015.

[6] T. B. C. Arias, P. van der Spek, and P. Avgeriou. A practice-driven systematic review of dependency analysis solutions. *Empirical Software Engineering*, 16:544–586, 2011.

[7] W. Azfal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51:957–976, 2009.

[8] M. Banerjee, M. Capozzoli, L. McSweeney, and D. Sinha. Beyond kappa: A review of interrater agreement measures. *Canadian Journal of Statistics*, 27(1):3–23, 1999.

[9] M. Bano and D. Zowghi. A systematic review on the relationship between user involvement and system success. *Information & Software Technology*, 58(148-169), 2015.

[10] E. Barends and D. M. Rousseau. *Evidence-Based Management: How to use evidence to make better organizational decisions.* Kogan Page, 2018.

[11] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp. Motivation in software engineering: A systematic literature review. *Information and Software Technology*, 50(9–10):860 – 878, 2008.

[12] J. Biolchini, P. Mian, A. Natali, and G. Travassos. Systematic review in software engineering. Technical Report ES679/05, COPPE/UFRJ, 2005.

[13] A. Booth, D. Papaioannou, and A. Sutton. *Systematic Approaches to a Successful Literature Review.* Sage Publications Ltd, 2012.

[14] P. Bourque and R. E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0.* IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014.

[15] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571 – 583, 2007.

[16] M. Brhel, H. Meth, A. Maedche, and K. Werder. Exploring principles of user-centered agile software development: A literature review. *Information & Software Technology*, 61:163–181, 2015.

[17] D. Budgen, J. Bailey, M. Turner, B. Kitchenham, P. Brereton, and S. Charters. Cross-domain investigation of empirical practices. *IET Software*, 3(5):410–421, 2009. EASE special section.

[18] D. Budgen, P. Brereton, S. Drummond, and N. Williams. Reporting systematic reviews: Some lessons from a tertiary study. *Information and Software Technology*, 95:62 – 74, 2018.

[19] D. Budgen, P. Brereton, N. Williams, and S. Drummond. The contribution that empirical studies performed in industry make to the findings of systematic reviews: A tertiary study. *Information and Software Technology*, 94:234 – 244, 2018.

[20] D. Budgen, S. Drummond, P. Brereton, and N. Holland. What scope is there for adopting evidence-informed teaching in software engineering? In *Proceedings of 34th International Conference on Software Engineering (ICSE 2012)*, pages 1205–1214. IEEE Computer Society Press, 2012.

[21] B. Cartaxo, G. Pinto, E. Vieira, and S. Soares. Evidence Briefings: Towards a medium to transfer knowledge from systematic reviews to practitioners. In *Proceedings of 2016 Conference on Empirical Software Engineering and Measurement (ESEM 2016)*, pages 1–10, 2016.

[22] M. Coldwell, T. Greany, S. Higgins, C. Brown, B. Maxwell, B. Stiell, L. Stoll, B. Willis, and H. Burns. Evidence-informed teaching: an evaluation of progress in England. Technical report, Department for Education, 2017.

[23] D. S. Cruzes and T. Dybå. Research synthesis in software engineering: A tertiary study. *Information and Software Technology*, 53(5):440 – 455, 2011.

[24] F. Q. da Silva, A. L. Santos, S. Soares, A. C. C. França, C. V. Monteiro, and F. F. Maciel. Six years of systematic literature reviews in software engineering: An updated tertiary study. *Information and Software Technology*, 53(9):899–913, 2011.

[25] P. Devanbu, T. Zimmermann, and C. Bird. Belief & evidence in empirical software engineering. In *Proceedings 38th IEEE International Conference on Software Engineering (ICSE 2016)*, pages 108–119. ACM Press, 2016.

[26] J. Díaz, J. Pérez, P. P. Alarcón, and J. Garbajosa. Agile product line engineering–a systematic literature review. *Software — Practice and Experience*, 41:921–941, 2011.

[27] O. Dieste and N. Juristo. Systematic review and aggregation of empirical studies on elicitation techniques. *IEEE Transactions on Software Engineering*, 37(2):283–304, 2011.

[28] T. Dybå and T. Dingsøyr. Strength of evidence in systematic reviews in software engineering. In

Proceedings of International Symposium on Empirical Software Engineering and Metrics (ESEM), pages 178–187, 2008.

[29] T. Dybå, B. Kitchenham, and M. Jörgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, 2005.

[30] E. Engström, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52:14–30, 2010.

[31] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76:378–382, 1971.

[32] A. H. Ghapanchi and A. Aurum. Antecedents to IT personnel's intentions to leave: A systematic literature review. *Journal of Systems & Software*, 84:238–249, 2011.

[33] R. Giuffrida and Y. Dittrich. Empirical studies on the use of social software in global software development–A systematic mapping study. *Information & Software Technology*, 55:1143–1164, 2013.

[34] C. L. Goues, C. Jaspan, I. Ozkaya, M. Shaw, and K. T. Stolee. Bridging the Gap: From research to practical advice. *IEEE Software*, 35(5):50–57, 2018.

[35] J. Gurevitch, J. Koricheva, S. Nakagawa, and G. Stewart. Meta-analysis and the science of research synthesis. *Nature*, 555:175–182, 2018.

[36] G. H. Guyatt, A. D. Oxman, G. E. Vist, R. Kunz, Y. Falck-Ytter, P. Alonso-Coello, and H. J. Schünemann. Grade: an emerging consensus on rating quality of evidence and strength of recommendations. *British Medical Journal*, 336:924–926, 2008.

[37] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[38] J. Hannay, T. Dybå, E. Arisholm, and D. Sjøberg. The effectiveness of pair programming. a meta analysis. *Information & Software Technology*, 51(7):1110–1122, 2009.

[39] D. Heaton and J. C. Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information & Software Technology*, 67:207–219, 2015.

[40] S. Hopewell, A. Aisinga, and M. Clarke. Better reporting of randomized trials in biomedical journal and conference abstracts. *Journal of Information Science*, 34(2):162–173, 2008.

[41] A. Idri, F. A. Amazal, and A. Abran. Analogy-based software development effort estimation: A systematic mapping and review. *Information & Software Technology*, 58:206–230, 2015.

[42] M. Ivarsson and T. Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16:365–395, 2011.

[43] R. Jabangwe, J. Borstler, D. Smite, and C. Wohlin. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering*, 20:640–693, 2015.

[44] M. Jørgensen. Evidence-based guidelines for assessment of software development cost uncertainty. *IEEE Transactions on Software Engineering*, 31(11):942–954, 2005.

[45] M. Jørgensen. Forecasting of software development work effort: Evidence on expert judgement and formal models. *Int. Journal of Forecasting*, 23(3):449–462, 2007.

[46] S. U. Khan, M. Niazi, and R. Ahmad. Barriers in the selection of offshore software development oursourcing vendors: An exploratory study using a systematic literature review. *Information & Software Technology*, 53:693–706, 2011.

[47] B. Kitchenham. Procedures for undertaking systematic reviews. Technical report, Joint Technical Report Keele and Durham Universities, 2004.

[48] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering — a systematic literature review. *Information & Software Technology*, 51(1):7–15, 2009.

[49] B. Kitchenham, D. Budgen, P. Brereton, M. Turner, S. Charters, and S. Linkman. Large-Scale Software Engineering Questions–Expert Opinion or Empirical Evidence? *IET Software*, 1(5):161–171, 2007.

[50] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and Durham University Joint Report, 2007.

[51] B. Kitchenham, T. Dybå, and M. Jørgensen. Evidence-based software engineering. In *Proceedings of ICSE 2004*, pages 273–281. IEEE Computer Society Press, 2004.

[52] B. Kitchenham, R. Pretorius, D. Budgen, P. Brereton, M. Turner, M. Niazi, and S. Linkman. Systematic

literature reviews in software engineering — a tertiary study. *Information & Software Technology*, 52:792–805, 2010.

[53] B. A. Kitchenham, D. Budgen, and P. Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. Innovations in Software Engineering and Software Development. CRC Press, 2015.

[54] E. Kupiainen, M. V. Mäntylä, and J. Itkonen. Using metrics in agile and lean software development – a systematic literature review of industrial studies. *Information & Software Technology*, 62:143–163, 2015.

[55] J. Lavis, G. Permanand, A. Oxman, S. Lewin, and A. Fredheim. SUPPORT tools for evidence-informed health policy-making (stp) 13: Preparing and using policy briefs to support evidence-informed policy-making. *Health Research Policy and Systems*, 7:S13, 2009.

[56] M. Leitner and S. Rinderle-Ma. A systematic review on security in process-aware information systems. *Information & Software Technology*, 56(3):273–293, 2014.

[57] Z. Li, H. Zhang, L. O'Brien, R. Cai, and S. Flint. On evaluating commercial cloud services: A systematic review. *Journal of Systems & Software*, 86:2371–2393, 2013.

[58] L. B. Lisboa, V. C. Garcia, D. Lucrédio, E. S. de Almeida, S. R. de Lemos Meira, and R. P. de Mattos Fortes. A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1 – 13, 2010.

[59] A. M. Magdaleno, C. M. L. Werner, and R. M. de Araujo. Reconciling software development models: a quasi-systematic review. *Journal of Systems & Software*, 85:351–369, 2012.

[60] S. Malick, K. Das, and K. S. Khan. Tips for teaching evidence-based medicine in a clinical setting: lessons from adult learning theory. *Journal of the Royal Society of Medicine*, 101(11):536–543, 2008.

[61] S. Martinez-Fernandez, P. S. M. dos Santos, G. P. Ayala, X. Franch, and G. H. Travassos. Aggregating empirical evidence about the benefits and drawbacks of software reference architectures. In *Proceedings 2015 Conference on Empirical Software Engineering & Measurement*, pages 154–163, 2015.

[62] P. Mohagheghi and R. Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12:471–516, 2007.

[63] H. Munir, M. Moayyed, and K. Peterson. Considering rigor and relevance when evaluating test driven development: A systematic review. *Information & Software Technology*, 56:375–394, 2014.

[64] P. Naur and B. Randell, editors. *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*. NATO, 1968.

[65] J. Nelson and C. O'Beirne. Using evidence in the classroom: What works and why? Technical report, National Foundation for Educational Research (NFER), 2014.

[66] A. Nguyen-Duc, D. S. Cruzes, and R. Conradi. The impact of global dispersion on coordination, team performance and software quality – a systematic literature review. *Information & Software Technology*, 57:277–294, 2015.

[67] S. Oliver and K. Dickson. Policy-relevant systematic reviews to strengthen health systems: models and mechanisms to support their production. *Evidence & Policy*, 12(2):235–259, 2016.

[68] C. Pacheco and I. Garcia. A systematic literature review of stakeholder identification methods in requirements elicitation. *Journal of Systems & Software*, 85:2171–2181, 2012.

[69] N. Paternoster, C. Giardino, M. Unterkalmsteiner, and T. Gorschek. Software development in startup companies: A systematic mapping study. *Information & Software Technology*, 56:1200–1218, 2014.

[70] J. S. Persson, L. Mathiassen, J. Boeg, T. S. Madsen, and F. Steinson. Managing risks in distributed software projects: An integrative framework. *IEEE Transactions on Engineering Management*, 56(3):508–532, 2009.

[71] K. Peterson. Measuring and predicting software productivity: A systematic map and review. *Information & Software Technology*, 53:317–343, 2011.

[72] H. Petersson, T. Thelin, P. Runeson, and C. Wohlin. Capture-recapture in software inspections after 10 years research—theory, evaluation and application. *Journal of Systems and Software*, 72:249–264, 2004.

[73] M. Petticrew and H. Roberts. *Systematic Reviews in the Social Sciences A Practical Guide*. Blackwell Publishing, 2006.

[74] F. J. Pino, F. Garcia, and M. Piattini. Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Journal*, 16:237–261, 2008.

[75] R. Rabiser, P. Grunbacher, and D. Dhungana. Requirements for product derivation support: Re-

sults from a systematic literature review and an expert survey. *Information & Software Technology*, 52:324–346, 2010.

[76] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič. Software fault prediction metrics: A systematic literature review. *Information & Software Technology*, 55:1397–1418, 2013.

[77] Y. Rafique and V. Misic. The effects of test-driven development on external quality and productivity: A meta-analysis. *IEEE Transactions on Software Engineering*, 39(6), 2013.

[78] T. V. Ribeiro, J. Massollar, and G. H. Travassos. Challenges and pitfalls on surveying evidence in the software engineering technical literature: an exploratory study with novices. *Empirical Software Engineering*, 23:1594–1663, 2018.

[79] E. M. Rogers. *Diffusion of Innovations*. Free Press, New York, 5 edition, 2003.

[80] S. E. Rosenbaum, C. Glenton, and A. D. Oxman. Summary-of-findings tables in Cochrane reviews improved understanding and rapid retrieval of key information. *Journal of Clinical Epidemiology*, 63:620–626, 2010.

[81] H. Sharp, N. Baddoo, S. Beecham, T. Hall, and H. Robinson. Models of motivation in software engineering. *Information and Software Technology*, 51:219–233, 2009.

[82] F. S. Silva, F. S. F. Soares, A. L. Peres, I. M. de Azevedo, A. P. L. F. Vasconcelos, F. K. Kamei, and S. R. de Lemos Meira. Using CMMI together with agile software development: A systematic review. *Information & Software Technology*, 58(20-43), 2015.

[83] H. A. Simon. The structure of ill-structured problems. *Artificial Intelligence*, 4:181–201, 1973.

[84] D. Smite, C. Wohlin, T. Gorschek, and R. Feldt. Empirical evidence in global software engineering: a systematic review. *Empirical Software Engineering*, 15:91–118, 2010.

[85] M. Staples and M. Niazi. Systematic review of organizational motivations for adopting CMM-based SPI. *Information and Software Technology*, 50:605–620, 2008.

[86] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information & Software Technology*, 59(67-85), 2015.

[87] C. Theisen, M. Dunaiski, L. Williams, and W. Visser. Software engineering research at the international conference on software engineering in 2016. *ACM Software Engineering Notes*, 42(4):1–10, 2017.

[88] S. Tiwari and A. Gupta. A systematic literature review of use case specifications research. *Information & Software Technology*, 67:128–158, 2015.

[89] M. Turner, B. Kitchenham, P. Brereton, S. Charters, and D. Budgen. Does the technology acceptance model predict actual use? A systematic literature review. *Information and Software Technology*, 52(5):463 – 479, 2010.

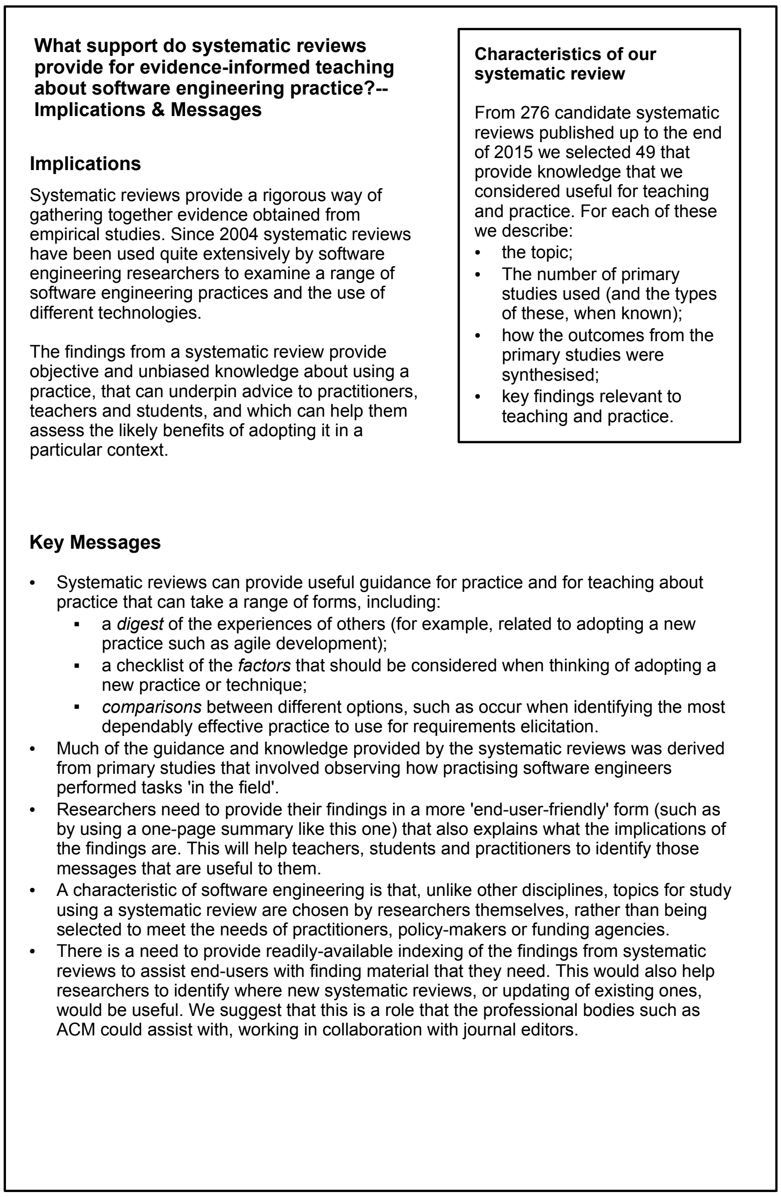
[90] E. Tüzün, B. Tekinerdogan, M. E. Kalender, and S. Bilgen. Empirical evaluation of a decision support model for adopting software product line engineering. *Information & Software Technology*, 60:77–101, 2015.

[91] B. J. Williams and J. C. Carver. Characterizing software architecture changes: A systematic review. *Information & Software Technology*, 52(1):31–51, 2010.

[92] M. Zarour, A. Abran, J.-M. Desharnais, and A. Alarifi. An investigation into the best practices for the successful design and implementation of lightweight software process assessment methods: A systematic literature review. *Journal of Systems & Software*, 101:180–192, 2015.

[93] C. Zhang and D. Budgen. What do we know about the effectiveness of software design patterns? *IEEE Transactions on Software Engineering*, 38(5):1213–1231, 2012.

A. Examples of a one-page summary



What support do systematic reviews provide for evidence-informed teaching about software engineering practice?-- Implications & Messages

Implications

Systematic reviews provide a rigorous way of gathering together evidence obtained from empirical studies. Since 2004 systematic reviews have been used quite extensively by software engineering researchers to examine a range of software engineering practices and the use of different technologies.

The findings from a systematic review provide objective and unbiased knowledge about using a practice, that can underpin advice to practitioners, teachers and students, and which can help them assess the likely benefits of adopting it in a particular context.

Characteristics of our systematic review

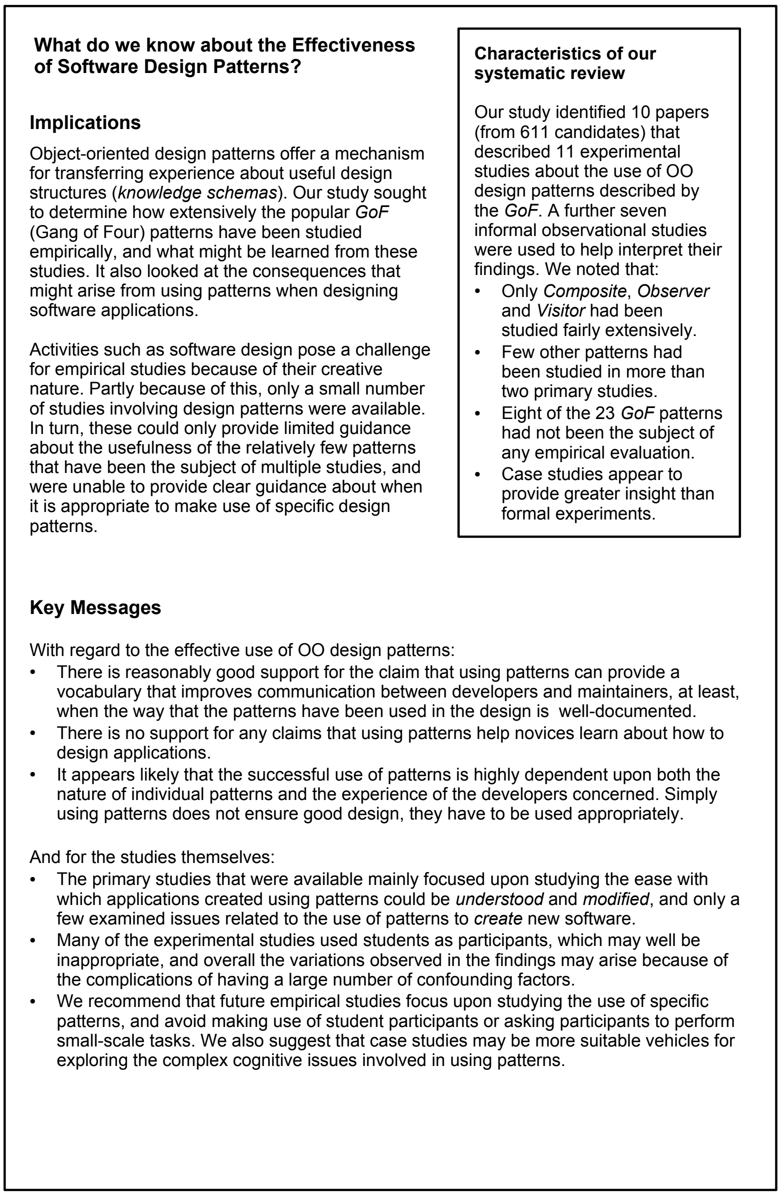
From 276 candidate systematic reviews published up to the end of 2015 we selected 49 that provide knowledge that we considered useful for teaching and practice. For each of these we describe:

- the topic;
- The number of primary studies used (and the types of these, when known);
- how the outcomes from the primary studies were synthesised;
- key findings relevant to teaching and practice.

Key Messages

- Systematic reviews can provide useful guidance for practice and for teaching about practice that can take a range of forms, including:
  - a *digest* of the experiences of others (for example, related to adopting a new practice such as agile development);
  - a checklist of the *factors* that should be considered when thinking of adopting a new practice or technique;
  - *comparisons* between different options, such as occur when identifying the most dependably effective practice to use for requirements elicitation.
- Much of the guidance and knowledge provided by the systematic reviews was derived from primary studies that involved observing how practising software engineers performed tasks 'in the field'.
- Researchers need to provide their findings in a more 'end-user-friendly' form (such as by using a one-page summary like this one) that also explains what the implications of the findings are. This will help teachers, students and practitioners to identify those messages that are useful to them.
- A characteristic of software engineering is that, unlike other disciplines, topics for study using a systematic review are chosen by researchers themselves, rather than being selected to meet the needs of practitioners, policy-makers or funding agencies.
- There is a need to provide readily-available indexing of the findings from systematic reviews to assist end-users with finding material that they need. This would also help researchers to identify where new systematic reviews, or updating of existing ones, would be useful. We suggest that this is a role that the professional bodies such as ACM could assist with, working in collaboration with journal editors.

Figure 2. Example 1, summarising this tertiary study



What do we know about the Effectiveness of Software Design Patterns?

Implications

Object-oriented design patterns offer a mechanism for transferring experience about useful design structures (*knowledge schemas*). Our study sought to determine how extensively the popular *GoF* (Gang of Four) patterns have been studied empirically, and what might be learned from these studies. It also looked at the consequences that might arise from using patterns when designing software applications.

Activities such as software design pose a challenge for empirical studies because of their creative nature. Partly because of this, only a small number of studies involving design patterns were available. In turn, these could only provide limited guidance about the usefulness of the relatively few patterns that have been the subject of multiple studies, and were unable to provide clear guidance about when it is appropriate to make use of specific design patterns.

Characteristics of our systematic review

Our study identified 10 papers (from 611 candidates) that described 11 experimental studies about the use of OO design patterns described by the *GoF*. A further seven informal observational studies were used to help interpret their findings. We noted that:

- Only *Composite*, *Observer* and *Visitor* had been studied fairly extensively.
- Few other patterns had been studied in more than two primary studies.
- Eight of the 23 *GoF* patterns had not been the subject of any empirical evaluation.
- Case studies appear to provide greater insight than formal experiments.

Key Messages

With regard to the effective use of OO design patterns:

- There is reasonably good support for the claim that using patterns can provide a vocabulary that improves communication between developers and maintainers, at least, when the way that the patterns have been used in the design is well-documented.
- There is no support for any claims that using patterns help novices learn about how to design applications.
- It appears likely that the successful use of patterns is highly dependent upon both the nature of individual patterns and the experience of the developers concerned. Simply using patterns does not ensure good design, they have to be used appropriately.

And for the studies themselves:

- The primary studies that were available mainly focused upon studying the ease with which applications created using patterns could be *understood* and *modified*, and only a few examined issues related to the use of patterns to *create* new software.
- Many of the experimental studies used students as participants, which may well be inappropriate, and overall the variations observed in the findings may arise because of the complications of having a large number of confounding factors.
- We recommend that future empirical studies focus upon studying the use of specific patterns, and avoid making use of student participants or asking participants to perform small-scale tasks. We also suggest that case studies may be more suitable vehicles for exploring the complex cognitive issues involved in using patterns.

Figure 3. Example 2, summarising paper #154

Table 13. Details for Review categorised as FND: #52

Characteristic	**Values**
1. Knowledge Unit	FND.ec (Engineering Economics for software)
2. Title	Systematic Review of Organizational Motivations for Adopting CMM-based SPI
3. Citation	[85]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made. (Gives counts of studies that identify different reasons.)
6. #Primary Studies	49 (all explicit industry)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. Organisations adopted CMM based SPI mainly to improve product quality and project performance but also to improve process management. 2. Satisfying customers was not a common reason for adopting CMM-based SPI. 3. The two most common process related reasons for adopting SPI were to make processes more visible and measurable.
9. Recommendations	None.
10. Author Response	The authors observed that meeting 'customer demands' in the form of contractual requirement was a fairly major reason for adoption, rather than 'customer satisfaction'. They also observed that providing *assurance* for customers through high ratings was a legitimate reason for recommending the adoption of CMM(I).

B. The findings and recommendations from the reviews

Table 14. Details for Reviews categorised as PRF: #54

Characteristic	**Values**
1. Knowledge Unit	(no specific KU)
2. Title	Motivation in Software Engineering: A systematic literature review
3. Citation	[11]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made. (Lists studies identifying specific motivators.)
6. #Primary Studies	79 (not described)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. The most frequently cited motivators relate to the 'need to identify with the task' (clear goals, personal interest, understanding the purpose of a task, how it fits with the whole, job satisfaction, and working on an identifiable piece of quality work). Having a clear career path and a variety of tasks is also found motivating. 2. Learning, exploring new techniques and problem solving appear to be motivating aspects of SE. 3. Indicators of demotivation were mainly turnover and absenteeism. 4. Key de-motivators are poor working conditions and lack of resources
9. Recommendations	None.
10. Author Response	None.

Table 15. Details for Reviews categorised as PRF: #118

Characteristic	**Values**
1. Knowledge Unit	(no specific KU)
2. Title	Models of motivation in software engineering
3. Citation	[81]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made. (See note for #54.)
6. #Primary Studies	Same as #54 above.
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. A list of 21 motivators is provided in the paper. 2. A new model of motivation in SE (the MOCC model) is presented using the results from the review reported in detail in paper #54)
9. Recommendations	None.
10. Author Response	None.

Table 16. Details for Reviews categorised as PRF: #135

Characteristic	**Values**
1. Knowledge Unit	PRF.psy (Group dynamics and psychology)
2. Title	Antecedents to IT personnel's intentions to leave: A systematic literature review
3. Citation	[32]
4. DARE Score	3.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	72 (all implicitly using industry participants)
7. Synthesis used	Thematic Analysis
8. Findings	"Publications reviewed suggest that male IT workers are more likely to leave an organisation than their female counterparts. Younger employees also appear more inclined to leave (mainly due to lower job satisfaction) compared to their older counterparts). Importantly, higher educated IT professionals are more likely to leave a company because of low job satisfaction. Additionally, married IT practitioners as well as those with a lower organisational tenure have a lower tendency to leave an organisation. IT managers can use these insights to assist with their recruitment decisions and employee retention initiatives."
9. Recommendations	1. To overcome role ambiguity and role conflict, managers should: a. communicate clearly and provide clear and precise information about what they expect from their IT professionals. b. make sure that their personnel have the required training and knowledge to carry out their jobs well. c. allow their IT professionals to know the intent of and reasons for doing a specific task. d. better design and define tasks so that the start and end of each task is clear. e. clearly define the sequence in which sub-tasks are carried out. 6. determine task priorities associated with the job. 2. To overcome perceived workload demands managers should maintain an awareness of the workloads of their high valued IT professionals. Direct face-to-face communications has been reported as the most effective means of overcoming this problem. 3. IT managers should be conscious of the benefits of enhanced employee autonomy because lack of autonomy can lead to turnover decision through work exhaustion. Managers should provide IT professionals with enough autonomy and flexibility to reduce exhaustion they might feel because of the structure of their work and should design IT roles that offer enough freedom for IT professional to be innovative and pursue their own thoughts and ideas. *(plus five other recommendations, omitted for reasons of space)*
10. Author Response	None.

Table 17. Details for Reviews categorised as PRF: #246

Characteristic	**Values**
1. Knowledge Unit	PRF.psy (Group dynamics and psychology)
2. Title	A systematic literature review on the barriers faced by newcomers to open source software projects
3. Citation	[86]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made. (Lists studies identifying specific barriers.)
6. #Primary Studies	20 (implicitly drawn from industry)
7. Synthesis used	Grounded Theory
8. Findings	1. For projects, improvement in community receptivity and more appropriate collaborative environments for OSS development can result in better support for newcomers. 2. “Keeping the code simple and the documentation organized and up-to-date could potentially increase the odds of receiving contributions from newcomers.”
9. Recommendations	“..newcomers that wish to contribute must have a blend of domain knowledge, technical skills, and social interaction, which can increase the odds of a successful joining. The interactions are driven by artifacts that reflect the technical and domain expertise. It is the result of these interactions that will allow both newcomers and developers to perceive the level and possibly lack of background that hinders effective contributions to the project.”
10. Author Response	None.

Table 18. Details for Reviews categorised as VAV: #15

Characteristic	**Values**
1. Knowledge Unit	VAV.rev (Reviews and static analysis)
2. Title	Capture-recapture in software inspections after 10 years research—theory, evaluation and application
3. Citation	[72]
4. DARE Score	1.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	25 (1 explicitly from industry, the others not identified)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. Most estimators underestimate. 2. Mh-JK (Jackknife) is the best estimator for software inspections. 3. Mh-JK is appropriate to use for 4 reviewers and more. 4. DPM is the best curve fitting method. 5. Capture-recapture estimators can be used together with perspective-based reasoning (PBR).
9. Recommendations	None.
10. Author Response	None.

Table 19. Details for Reviews categorised as VAV: #66

Characteristic	**Values**
1. Knowledge Unit	VAV.tst (Testing)
2. Title	A systematic review of search-based testing for non-functional system properties
3. Citation	[7]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	35 (17 explicitly from industry; 18 academic)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. For performance, genetic algorithms (GAs) consistently outperform random and statistical testing in a wide variety of situations, producing comparatively longer execution times faster and also finding new bounds on best case execution times. 2. GAs were also able to perform better than human testers, and where this failed to occur, it could be attributed to the complexity of the test objects inhibiting evolutionary testability.
9. Recommendations	None.
10. Author Response	None.

Table 20. Details for Reviews categorised as VAV: #82

Characteristic	**Values**
1. Knowledge Unit	VAV.tst (Testing)
2. Title	A systematic review on regression test selection techniques
3. Citation	[30]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	36 (4 explicitly from industry; 32 not stated)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. The minimization technique is the most efficient in reducing time and/or number of test cases to run. However this is an unsafe technique and all but one of six studies report significant losses in fault detection. 2. DejaVu (Rothermel and Harrold) is the most efficient safe technique for reducing test cases. However, analysis time for this is shown to be too long (exceeding the time for rerunning all test cases) in early experiments, although in later ones (using different subject programs) it is shown to be good. 3. Regression test selection techniques have to be tailored to specific situations e.g. initially based on the classification of techniques.
9. Recommendations	None.
10. Author Response	None.

Table 21. Details for Reviews categorised as VAV: #167

Characteristic	**Values**
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	On evaluating commercial Cloud services: A systematic review
3. Citation	[57]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	82 (all implicitly from industry)
7. Synthesis used	Could not be identified
8. Findings	1. Existing evaluations have used a large number of metrics to measure performance as well as cost. 2. There is still a lack of metrics for evaluating Cloud elasticity. 3. There are still no metrics that can be used to assess security.
9. Recommendations	None.
10. Author Response	The authors suggest a further finding is: 4. Various traditional benchmarks have been employed to evaluate performance of Cloud services.

Table 22. Details for Reviews categorised as VAV: #197

Characteristic	**Values**
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	Software fault prediction metrics: A systematic literature review
3. Citation	[76]
4. DARE Score	4.5
5. Strength of Evidence	An informal assessment was provided.
6. #Primary Studies	106 (81 explicitly from industry; 25 academic studies)
7. Synthesis used	Vote Counting
8. Findings	1. Cyclomatic complexity was fairly effective in large and OO environments. Although not effective in all categories, the overall effectiveness was estimated as moderate. 2. Halstead's metrics were ineffective when compared with other metrics and were estimated as inappropriate for software fault prediction. 3. The most frequently used and most successful among the OO metrics were the CK metrics. From these, COB, WMC and RFC were effective across all groups. LCOM is not not very successful at finding faults, DIT and NOC were reported as untrustworthy. 4. OO and process metrics are more successful at fault prediction than traditional size and complexity metrics. 5. Source code metrics do not perform well in finding post-release faults. 6. Process metrics were found to be successful at finding post-release faults. 7. Size measures like LOC metrics are simple and easy to extract; but as with complexity metrics, have only limited predictive capabilities. They are partly successful at ranking the most fault prone modules, not the most reliable or successful metrics.
9. Recommendations	1. Industry practitioners looking for effective and reliable process metrics should consider code churn, the number of changes, the age of a module and the change set size metrics. 2. Not all OO metrics are good predictors of faults. NOC and DIT are unreliable and should not be used in fault prediction models. 3. Industry practitioners looking for effective and reliable process metrics (in large post-release systems) could also try static code metrics (e.g. CBO, RFC & WMC) but should keep in mind that they have some limitations in highly iterative and agile development environments.
10. Author Response	None.

Table 23. Details for Reviews categorised as VAV: #205

Characteristic	**Values**
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	Considering rigor and relevance when evaluating test driven development: A systematic review
3. Citation	[63]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	41 (22 explicitly from industry; 19 academic studies)
7. Synthesis used	Vote Counting
8. Findings	1. Studies with high rigour and relevance indicate that practitioners wanting to adopt TDD will improve their code quality and at the same time maintain or reduce their productivity levels. 2. Studies with high rigour and relevance indicate that practitioners wanting to adopt TDD will reduce complexity and at the same time maintain or reduce their productivity levels. 3. Studies with high relevance and low rigour suggest that there is a potential to increase external quality, but at the expense of development time and productivity.
9. Recommendations	None.
10. Author Response	None.

Table 24. Details for Reviews categorised as VAV: #252

Characteristic	**Values**
1. Knowledge Unit	VAV.fnd (VAV terminology and foundations)
2. Title	Using metrics in Agile and Lean Software Development—A systematic literature review of industrial studies
3. Citation	[54]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	30 (all explicitly from industry)
7. Synthesis used	Thematic Analysis
8. Findings	1. The targets of measurement are the product and the process, but not the people. 2. Documentation is not measured, instead the focus is on the actual product and features. 3. The use of metrics can motivate people and change the way that people behave in terms of which issues they pay attention to. 4. Industrial agile teams use situative metrics based on need. 5. Defect counts and customer satisfaction are two of the four high influence metrics (after velocity and effort estimate), although not directly recommended by Lean or Agile methods. 6. Areas where metrics are used are sprint and project planning, sprint and project progress tracking, understanding and improving quality, fixing software process problems and motivating people - so not dissimilar to use in plan driven.
9. Recommendations	None.
10. Author Response	None.

Table 25. Details for Review categorised as DES: #124

Characteristic	**Values**
1. Knowledge Unit	DES.ar (Architectural design)
2. Title	Characterising software architecture changes: A systematic review
3. Citation	[91]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	130 (not stated)
7. Synthesis used	Thematic Analysis
8. Findings	The Software Architecture Change Characterization Scheme (SACCS) was developed as a result of the review and can (could?) be used to assist developers and maintainers in assessing the potential impact of a proposed change and deciding whether it is feasible to implement the change. Where the change is crucial the scheme will (could?) help generate consensus on how to approach change implementation and provide an indication of the difficulty.
9. Recommendations	None.
10. Author Response	The authors reviewed our analysis and agreed with it.

Table 26. Details for Review categorised as DES: #130

Characteristic	**Values**
1. Knowledge Unit	DES.str (Design strategies)
2. Title	A systematic review of comparative evidence of aspect-oriented programming
3. Citation	[3]
4. DARE Score	4.5
5. Strength of Evidence	The GRADE system was used. Overall the current strength of evidence about benefits and limitations of AOP approaches compared to non-AOP approaches is *low*.
6. #Primary Studies	22 (6 implicitly industry studies, 16 academic studies)
7. Synthesis used	Narrative synthesis + Vote counting
8. Findings	1. Overall AOP provides improvement over non-AOP based solutions. 2. AOP has a positive effect on performance (within contexts similar to those used in the evaluations). 3. In larger systems where concern scattering and tangling is expected to be widespread, introducing aspects is likely to significantly reduce the number of lines of code. 4. AOP has a positive effect on modularity (but context of use should be carefully assessed). 5. AOP has the potential to develop evolvable and maintainable software.
9. Recommendations	None.
10. Author Response	None.

Table 27. Details for Review categorised as DES: #154

Characteristic	**Values**
1. Knowledge Unit	DES.dd (Detailed design)
2. Title	What do we know about the Effectiveness of Software Design Patterns?
3. Citation	[93]
4. DARE Score	2.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	18 (11 industrial and 7 academic studies)
7. Synthesis used	Narrative Synthesis
8. Findings	Patterns do not appear to help novices learn about design.
9. Recommendations	None.
10. Author Response	The authors reviewed our analysis and agreed with it.

Table 28. Details for Review categorised as MAA: #123

Characteristic	**Values**
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	A Systematic review of domain analysis tools
3. Citation	[58]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	19 (7 are implicitly industry studies, 12 are academic studies)
7. Synthesis used	Could not be determined.
8. Findings	1. No tools support all functionalities of a specific process. 2. The majority of the analysed tools have similar functionalities. 3. The majority of tools are still being developed and used in an academic environment. 4. The documentation function is being explored more in the more recent tools investigated. 5. The domain analysis process without tool support can lead to an unsuccessful result but the use of any tool will not necessarily lead to an effective result.
9. Recommendations	None.
10. Author Response	The authors observe that a finding could be to categorize functionalities as essential, important and low.

Table 29. Details for Review categorised as MAA: #126

Characteristic	**Values**
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	Does the technology acceptance model predict actual use?
3. Citation	[89]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	79 (type not reported)
7. Synthesis used	Vote counting
8. Findings	Perceived usefulness (PU) and perceived ease of use (PEU) are less likely than behavioural intention (BI) to be correlated with actual use.
9. Recommendations	None.
10. Author Response	None.

Table 30. Details for Review categorised as MAA: #146

Characteristic	**Values**
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	A practice-driven systematic review of dependency analysis solutions
3. Citation	[6]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	65 (38 explicit industry, 27 not specified)
7. Synthesis used	Narrative Synthesis
8. Findings	1. Source-code based solutions identify dependencies through code constructs such as function calls and shared variables. Approaches that use this concrete evidence have a high degree of accuracy when it comes to the dependencies they identify, which makes them very reliable and very attractive for practitioners as the resulting information is very tangible. However, they are less suited to analyzing runtime system behaviour. 2.Solutions using diagrammatic and semi-formal descriptions are more appealing for practitioners following architecture-driven approaches. Practitioners find these solutions useful to describe dependency information at an architecture level. However, for an efficient application of these solutions, it is necessary to keep up-to-date and synchronize the system requirements, design, and implementation. 3. Solutions using run-time and configuration information are applicable in practice due to two main characteristics. First, these solutions are non-intrusive with respect to the development activities. Often, in a research setting, the overhead and maintenance cost of an infrastructure to collect data for dependency analysis is overlooked, whereas practitioners are more concerned about the cost and overhead of maintaining a reliable and up-to-date instrumentation of their system. This is even more important, in heterogeneous situations where multi-vendor components are used and instrumentation cannot be inserted into the system because of security, licensing, lack of knowledge, or other technical constraints. Second, although these solutions are limited by their coverage and links to the system source code, practitioners consider these solutions valid approximations, especially for problem-driven approaches.
9. Recommendations	None.
10. Author Response	None.

Table 31. Details for Review categorised as MAA: #155

Characteristic	**Values**
1. Knowledge Unit	MAA.tm (Types of models)
2. Title	A Systematic Literature Review on Fault Prediction Performance in Software Engineering
3. Citation	[37]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	36 (35 explicit industry, 1 academic study)
7. Synthesis used	Thematic Analysis
8. Findings	1. Models that work well tend to be built in a context where the systems are large. 2. In terms of context, there is no evidence to suggest that the maturity of systems or language used is related to predictive performance 3. It may be more difficult to build reliable prediction models for some application domains (e.g. embedded systems). 4. The independent variables used by predictive models that work well seem to be sets of metrics. 5. Models that use KLOC perform no worse than where only single sets of other static code metrics are used. 6. The spam filtering technique based on source code performs relatively well. 7. Models that perform well tend to use simple, easy to use modeling techniques such as Nave Bayes or Logical Regression. More complex modeling techniques such as SVM tend to be used by models which perform relatively less well. 8. Successful models tend to be trained on large datasets which have a relatively high proportion of faulty units. 9. Successful models tend to use a large range of metrics on which feature selection was implied. 10. For successful models, default parameters for the modelling technique were adjusted to ensure the technique would perform effectively.
9. Recommendations	None.
10. Author Response	Agreed with our extracted findings.

Table 32. Details for Reviews categorised as REQ: #134

Characteristic	**Values**
1. Knowledge Unit	REQ.er (Eliciting requirements)
2. Title	Systematic Review and Aggregation of Empirical Studies on Elicitation Techniques
3. Citation	[27]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	32 (7 explicit industry; 7 academic; 18 unclear)
7. Synthesis used	Vote counting
8. Findings	1. Unstructured interviews (although it is reasonable to assume that the same applies to structured interviews) are equally as or more effective than introspective technique (such as protocol analysis) and sorting techniques. 2. Unstructured interviews (although it is reasonable to assume that the same applies to structured interviews) output more complete information than introspective technique (such as protocol analysis), sorting techniques and Laddering. 3. Unstructured interviews (although it is reasonable to assume that the same applies to structured interviews) are less efficient than sorting techniques and Laddering but as efficient as introspective techniques (such as protocol analysis). 4. The introspective techniques (such as protocol analysis) are the worst of all the tested techniques in all the dimensions (effectiveness, efficiency, completeness) and are outperformed by unstructured interviews (although it is reasonable to assume that the same applies to structured interviews), and sorting techniques and Laddering. 5. Laddering is preferable to sorting techniques (as well as introspection techniques).
9. Recommendations	None
10. Author Response	None.

Table 33. Details for Reviews categorised as REQ: #161

Characteristic	**Values**
1. Knowledge Unit	REQ.er (Eliciting requirements)
2. Title	A systematic literature review of stakeholder identification methods in requirements elicitation
3. Citation	[68]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Lists papers addressing specific practices and issues.)
6. #Primary Studies	42 (all implicitly industry)
7. Synthesis used	Thematic Analysis
8 Findings	None.
9. Recommendations	1. Assign appropriate roles to stakeholders through analysis of skills, behaviors in group dynamics and personality tests. 2. Establish constructive interaction between all stakeholders and between stakeholders and the system. 3. Classify requirements elicited from stakeholders according to an evaluation of their priorities in the project
10. Author Response	The authors agreed with our interpretation.

Table 34. Details for Reviews categorised as REQ: #259

Characteristic	**Values**
1. Knowledge Unit	REQ.rsd (Requirements specification and documentation)
2. Title	A systematic literature review of use case specifications research
3. Citation	[88]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made. (Lists papers addressing specific issues.)
6. #Primary Studies	119 (27 explicit industry; 11 academic; 81 unclear)
7. Synthesis used	Could not be identified.
8. Findings	1. Use case specifications were typically employed in two perspectives: documenting the functional requirements (typically using informal tabular or paragraph style formats); and for generating the lower-level software artifacts by using greater formalism to support a model-transformation process. 2. Use cases have evolved from paragraph format textual descriptions to a more formal keyword-oriented format for facilitating automated information retrieval. 3. Use cases have been applied and used in almost all the software development life cycle activities. However, knowledge about their applicability in planning and estimation and maintenance phases is limited, due to the limited number of published studies.
9. Recommendations	None.
10. Author Response	The authors suggested some rewording of the third conclusion (incorporated).

Table 35. Details for Review categorised as QUA: #219

Characteristic	**Values**
1. Knowledge Unit	QUA.pda (Product assurance)
2. Title	Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review
3. Citation	[43]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	99 (33 are implicitly from industry; 5 academic; and 61 could not be classified)
7. Synthesis used	Vote counting
8. Findings	1. Measures for complexity, cohesion, coupling and size show better consistency in their relationship with reliability and maintainability attributes across the primary studies than inheritance. 2. Measures that quantify inheritance properties show poor links to reliability and maintainability.
9. Recommendations	None.
10. Author Response	The authors observed that for the second conclusion, the poor showing of the inheritance measures might have stemmed from confounding factors in the primary studies.

Table 36. Details for Review categorised as PRO: #39

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Quality, productivity and economic benefits of software reuse: a review of industrial studies
3. Citation	[62]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	11 (all explicit industry)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	1. Defect, error or fault density is significantly reduced. 2. Rework effort is significantly reduced. 3. Apparent productivity improves significantly.
9. Recommendations	None.
10. Author Response	None.

Table 37. Details for Review categorised as PRO: #50

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Software process improvement in small and medium software enterprises: a systematic review
3. Citation	[74]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	45 (all explicit industry)
7. Synthesis used	(Not assessed for Dataset1)
8. Findings	It is difficult to successfully apply formal SPI programmes which use models such as CMM to SMEs.
9. Recommendations	None.
10. Author Response	None.

Table 38. Details for Review categorised as PRO: #138

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Measuring and predicting software productivity: a systematic map and review
3. Citation	[71]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	38 (25 explicit industry, 13 academic)
7. Synthesis used	Narrative Synthesis
8. Findings	1. The variety of model forms means that strong recommendations cannot be provided. However, the studies did not come to conclusions that contradicted each other. 2. Simulation overall provided promising results. 3. Time-series analysis/statistical process control also shows good results in identifying sharp shifts in process performance as well as shifts due to changes in the process. 4. To be able to give a recommendation on the predictive accuracy of regression for software productivity, the model should be built on a sub-set of data points and then used to predict the remaining data points. Thereafter the difference between prediction and actual values should be observed and measured.
9. Recommendations	1. When using univariate models it is important to be aware of high variances and difficulties when comparing productivities. Hence it is important to carefully document the context to be able to compare between products. Comparison should not be on productivity value alone and it is recommended that a scatter diagram be produced based on inputs and outputs to assure comparability of projects with respect to size. 2. When comparing projects it should be made clear what output and input consists of, for example, which lines are included in LOC measures. 3. When possible, use multivariate analysis when data is available, as throughout the software process many outputs are produced. Otherwise, productivity is biased towards one measure (eg LOC). 4. Managers need to be aware of validity threats present in the measures when conducting a comparison. Data should be interpreted with care and awareness of possible bias and noise in the data arising from measurement error. 5. No generic prediction model can be recommended as studies do not clearly agree on what are the predictors for software productivity. In fact, the predictors might differ between contexts. Hence companies need to identify and test predictors relevant to their context.
10. Author Response	The authors identify the following additional finding. 5. Data envelopment analysis is promising as it supports multivariate productivity measures, and allows identification of reference projects to which inefficient projects should be compared. This helps with identifying projects from which one can learn, and that are similar, so that evidence may be transferable.

Table 39. Details for Review categorised as PRO: #157

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis
3. Citation	[77]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	37 (10 explicit industry; 23 academic; 4 unclear)
7. Synthesis used	Meta-analysis
8. Findings	Use of TDD can result in a small improvement in quality (implicit).
9. Recommendations	None.
10. Author Response	None.

Table 40. Details for Review categorised as PRO: #160

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Reconciling software development models: A quasi-systematic review
3. Citation	[59]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	42 (all implicit industry studies)
7. Synthesis used	Thematic analysis
8. Findings	1. Three different levels of reconciliation are found: organisational, group and process. 2. Main opportunities for reconciliation are derived from collaboration and code availability. 3. There is a diversity of challenges—most salient is overcoming barriers to culture change.
9. Recommendations	None.
10. Author Response	Our findings were confirmed by the authors.

Table 41. Details for Review categorised as PRO: #174

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	A systematic literature review on the industrial use of software process simulation
3. Citation	[4]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	87 (all implicit industry studies)
7. Synthesis used	Could not be determined
8. Findings	No evidence of widespread adoption and impact of SPSM research on industry.
9. Recommendations	When using software process simulation models for scientific purposes, need to be sure that the appropriate steps with respect to model validity checking have been conducted, and do not rely upon a single simulation run.
10. Author Response	We have used a slight rewording of the recommendation suggested by the authors.

Table 42. Details for Review categorised as PRO: #228

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	An investigation into the best practices for the successful design and implementation of lightweight software process assessment methods: A systematic literature review
3. Citation	[92]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made. (Lists papers identifying successful practices.)
6. #Primary Studies	22 (all explicit industry studies)
7. Synthesis used	Thematic analysis
8. Findings	A set of 38 best practices has been collected and classified into five main areas: method, supportive tool, procedure, documentation and user best practices.
9. Recommendations	The paper has identified a set of best practices to support and inform designers and assessors for software process assessment.
10. Author Response	We have used some rewording suggested by the authors.

Table 43. Details for Review categorised as PRO: #249

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Exploring principles of user-centred agile software development: A literature review
3. Citation	[16]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Provides counts of papers identifying issues related to principles.)
6. #Primary Studies	83 (26 implicit industry studies, 57 that could not be classified)
7. Synthesis used	Content analysis
8. Findings	None.
9. Recommendations	1. User-centered agile software development should be based on separated product discovery and product creation phases. 2. In user-centered agile approaches, design and development should proceed in parallel interwoven tracks. 3. In user-centered agile approaches, tangible and up-to-date artifacts should be used to document and communicate product and design concepts, and should be accessible to all involved stakeholders.
10. Author Response	None.

Table 44. Details for Review categorised as PRO: #268

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Requirements for product derivation support: Results from a systematic literature review and an expert survey
3. Citation	[75]
4. DARE Score	2.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	118 (unclassified as no details provided)
7. Synthesis used	Narrative synthesis
8. Findings	Systematic Review followed by expert survey identified the following six requirements for product derivation support: 1. automated and interactive variability resolution 2. adaptability and extensibility 3. application requirements management support 4. flexible and user-specific visualisations of variability 5. end-user guidance 6. project management support
9. Recommendations	None.
10. Author Response	The authors agreed with our extracted findings.

Table 45. Details for Review categorised as PRO: #276

Characteristic	**Values**
1. Knowledge Unit	PRO.con (Process Concepts)
2. Title	Empirical evaluation of a decision support model for adopting software product line engineering
3. Citation	[90]
4. DARE Score	3.0
5. Strength of Evidence	No assessment was made. (Lists papers identifying relevant factors.)
6. #Primary Studies	31 (all implicit industry)
7. Synthesis used	Thematic Analysis + Vote Counting
8. Findings	1. The study identifies 25 factors that should be considered when investigating adoption of SPLE (e.g. business motivation, market potential, software architecture competence). In all, 39 questions that might be asked and 312 rules that could be applied are developed. Rules include recommendations & strategies (but only one example provided)
9. Recommendations	None.
10. Author Response	The authors agreed with our extracted findings.

Table 46. Details for Review categorised as PRO: #84

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	The effectiveness of pair programming: A meta-analysis
3. Citation	[38]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	19 (5 explicit industry and 14 academic studies)
7. Synthesis used	Meta-analysis
8. Findings	None.
9. Recommendations	If you do not know the seniority or skill levels of your programmers, but do have a feel for task complexity, then employ PP either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important.
10. Author Response	The authors agreed with our extracted recommendations.

Table 47. Details for Review categorised as PRO: #150

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Agile product line engineering—a systematic literature review
3. Citation	[26]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	39 (14 explicit industry and 25 not specified)
7. Synthesis used	Narrative synthesis
8. Findings	1. If software product line (SPL) developers do not have enough knowledge to completely perform the domain engineering (DE), agile software development (ASD) may facilitate the elicitation of further knowledge. 2. Trade-offs between SPLE and ASD provide the opportunity to apply the agile product line engineering (APLE) approach to a wider variety of projects than those served by only applying ASD or SPL methods. 3. When anticipated changes cannot be predicted and the product life cycle is not known, it would be advantageous to use an incremental approach such as APLE. 4. Agile processes may facilitate fast feedback cycles between requirements engineering (RE), development and field trial in innovative business.
9. Recommendations	None.
10. Author Response	The authors agreed with our extracted findings and observed that Table VII does implicitly provide some recommendations for practice.

Table 48. Details for Review categorised as PRO: #193

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Empirical studies on the use of social software in global software development—A systematic mapping study
3. Citation	[33]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	84 (61 explicit industry and 23 academic studies)
7. Synthesis used	Narrative synthesis
8. Findings	1. Social Networking sites help identify experts and provide awareness of people's expertise. 2. It is necessary to develop structures, rules, good practices and agreements for using SoSo in a work context and on a project basis.
9. Recommendations	None.
10. Author Response	None.

Table 49. Details for Review categorised as PRO: #215

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Software development in startup companies: A systematic mapping study
3. Citation	[69]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Lists papers identifying factors.)
6. #Primary Studies	43 (30 implicit industry and 13 that could not be classified)
7. Synthesis used	Thematic analysis
8. Findings	1. Light-weight methodologies to obtain flexibility in choosing tailored practices, and reactiveness to change the product according to business strategies is a useful process management practice in startups. 2. Fast releases to build a prototype in an evolutionary fashion and quickly learn from the users' feedback to address the uncertainty of the market is a useful process management practice in startups. 3. The use of well-known frameworks able to provide fast changeability of the product in its refactoring activities is a useful design and architectural practice in startups. 4. The use of existing components, leveraging third party code reinforcing ability to scale the product is a useful design and architectural practice in startups. 5. The use of ongoing customer acceptance with the use of focus groups of early adopters, which aims to determine the fitness of the product for the market is a useful quality assurance practice in startups. (plus six further conclusions)
9. Recommendations	None.
10. Author Response	The authors agreed with our interpretation.

Table 50. Details for Review categorised as PRO: #217

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Understanding the Influence of User Participation and Involvement on System Success—A Systematic Mapping Study
3. Citation	[1]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	82 (all implicit industry)
7. Synthesis used	Meta-analysis
8. Findings	1. “Given the vast amount of positive correlations, we can conclude that, even though the results are not completely consistent, the amount of studies with positive correlations of the various aspects of UPI on system success provides evidence of a robust and transferable effect.” 2. Most studies with negative correlations from aspects of UPI on system success were published more than 10 years ago. 3. UPI has a positive effect on user satisfaction and system use.
9. Recommendations	None.
10. Author Response	The authors suggested some revisions which were partly adopted.

Table 51. Details for Review categorised as PRO: #222

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	The Kanban approach, between agility and leanness: a systematic review
3. Citation	[2]
4. DARE Score	4.0
5. Strength of Evidence	No assessment was made. (Lists papers identifying relevant benefits.)
6. #Primary Studies	37 (all implicit industry)
7. Synthesis used	Case survey
8. Findings	1. There is a lack of of details and guidelines on how the Kanban approach can be used by IT organisations. 2. The Kanban board is an efficient visualisation tool.
9. Recommendations	Discuss Kanban elements together, based on the five pillars of the lean approach, to minimize the risk of evolving contradictory elements and to facilitate establishing guidelines and instructions on how to set up the Kanban approach, to give practitioners an overall framework that increases the likelihood of successfully implementing the Kanban approach in IT organisations.
10. Author Response	None.

Table 52. Details for Review categorised as PRO: #236

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	The impact of global dispersion on coordination, team performance and software quality—A systematic literature review
3. Citation	[66]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Provides counts of studies addressing factors.)
6. #Primary Studies	43 (40 explicit industry and 3 academic)
7. Synthesis used	Thematic analysis + Vote counting
8. Findings	1. The impact of each dispersion dimension on project outcomes is mediated by a different set of coordination issues in GSD. 2. A distributed task takes a longer time to communicate and resolve than a co-located task does in GSD. 3. Temporal dispersion has a positive impact on objective team performance while it has a negative impact on perceived team performance in GSD. 4. Geographical dispersion has a negative impact on software quality, at both file and project level in GSD. 5. Temporal dispersion has a negative impact on software quality, at both file and project level in GSD.
9. Recommendations	1. Managers should be aware of the influence of dispersion dimension at different organisational levels. At the individual level, lack of face-to-face interaction and working in different time zones affects directly and negatively a developers work. At the team and project level, the negative influences of these dispersion dimensions might be underestimated in considering different goals and priorities. This issue must be taken into account when aligning the objective of individuals and teams with organizational goals. 2. Decisions on which coordination mechanisms to use should depend on the current dispersion context setting, the current team coordination technology and practices, and prioritized type of interdependencies. Our summary shows that communication and shared artifacts should be used together as needed and a defined process should be adopted at the team and organizational levels.
10. Author Response	The authors agreed with our extracted findings.

Table 53. Details for Review categorised as PRO: #239

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Using CMMI together with agile software development: A systematic review
3. Citation	[82]
4. DARE Score	4.5
5. Strength of Evidence	Use of GRADE. Strength of evidence considered to be *low* for all findings.
6. #Primary Studies	60 (59 explicit industry and 1 academic)
7. Synthesis used	Thematic analysis
8. Findings	1 Agile methodologies have been used by companies to enhance their efforts to reach levels 2 and 3 of CMMI, with reports of applying agile practices to achieve level 5. 2. Agile methodologies alone are not sufficient to achieve the level required, it being necessary to resort to additional practices. 3. Organisations should seek to ensure that how CMMI and agile can be combined is understood and undertaken by those involved.
9. Recommendations	None.
10. Author Response	None.

Table 54. Details for Review categorised as PRO: #241

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	A systematic review on the relationship between user involvement and system success
3. Citation	[9]
4. DARE Score	4.5
5. Strength of Evidence	No assessment was made. (Lists papers identifying specific benefits.)
6. #Primary Studies	87 (all implicit industry)
7. Synthesis used	Thematic analysis + Vote counting
8. Findings	1. Identification of the right type of users who will be involved, and who will participate, are important factors according to the literature, but the review did not find enough empirical evidence about this to confirm it. 2. The perspective of user involvement is one of the most important factors. Analysis identified five major perspectives for user involvement: psychological, managerial, methodological, political, and cultural. 3. User involvement takes different forms for development of different types of system. 4. User satisfaction leads to system success (the top cited factor). 5. To achieve benefits in methodological and psychological perspectives, user involvement in the requirements phases seems to be most effective. 6. To achieve benefits for political and cultural perspectives, users need to be involved in the design and implementation phases.
9. Recommendations	None.
10. Author Response	The authors provided some comments which we have used to modify the findings.

Table 55. Details for Review categorised as PRO: #260

Characteristic	**Values**
1. Knowledge Unit	PRO.imp (Process Implementation)
2. Title	Claims about the use of software engineering practices in science: A systematic literature review
3. Citation	[39]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made. (Lists papers addressing specific issues.)
6. #Primary Studies	43 (all academic)
7. Synthesis used	Thematic analysis
8. Findings	1. Scientific software developers benefit from using a wide range of testing practices from software engineering. 2. Open-source is especially useful to scientific software developers. 3. Documentation is a necessary enabler of software quality. 4. Version control software is necessary for research groups with more than one developer. (Note: These were the conclusions with strongest supporting evidence.)
9. Recommendations	None.
10. Author Response	The authors provided some comments which we have used to modify the findings.

Table 56. Details for Review categorised as PRO: #8

Characteristic	**Values**
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Forecasting of software development work effort: Evidence on expert judgement and formal models
3. Citation	[45]
4. DARE Score	1.0
5. Strength of Evidence	Informally estimated as 'modest'.
6. #Primary Studies	16 (14 explicit industry and 2 academic studies)
7. Synthesis used	(Synthesis was not assessed for Dataset1)
8. Findings	1. The review does not support the view that we should replace expert judgement with models. 2. The review does not support the view that software estimation models are useless.. 3. Models failed to systematically perform better than the experts when estimating. 4. Two conditions for producing more accurate expert judgement-based effort seem to be that the models are not calibrated to the organization using them, and that the experts possess important contextual information not included in the formal models and apply it efficiently. 5. The use of models, either alone or in combination with expert judgement, may be particularly useful when i) there are situational biases that are believed to lead to a strong bias towards overoptimism; ii) the amount of contextual information possessed by experts is low; and iii) the models are calibrated to the organization using them.
9. Recommendations	It is best to use a combination of models and experts when estimating the level of effort required to complete software development tasks.
10. Author Response	None.

Table 57. Details for Review categorised as PRO: #22

Characteristic	**Values**
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty
3. Citation	[44]
4. DARE Score	2.5
5. Strength of Evidence	An assessment was made for each guideline, identifying supporting papers.
6. #Primary Studies	40 (none could be classified)
7. Synthesis used	(Synthesis was not assessed for Dataset1)
8. Findings	None.
9. Recommendations	1. Do not rely solely on unaided, Intuition-based processes. (*Strong* evidence.) 2. Do not replace expert judgement with formal models. (*Medium* evidence.) 3. Apply structured and explicit judgement-based processes. (*Strong* evidence. 4. Apply strategies based on an outside view of the project. (*Medium* evidence.) 5. Use motivational mechanisms with care and only if it is likely that more effort leads to improved assessments. (*Medium* evidence.) 6. Frame the assessment problem to fit the structure of the uncertainty relevant information and the assessment process. (*Medium* evidence.)
10. Author Response	None.

Table 58. Details for Review categorised as PRO: #102

Characteristic	**Values**
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Managing risks in distributed software projects: An integrative framework
3. Citation	[70]
4. DARE Score	2.5
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	72 (implicit industry studies)
7. Synthesis used	(Synthesis was not assessed for Dataset1)
8. Findings	1. Built framework demonstrating complex nature of risks in GDSP and offers concepts and heuristics that practitioners can use to assess and control the risks they face in specific projects. Can be used by project managers. 2. Provides a useful vocabulary.
9. Recommendations	1. Revisit risk management regularly during project lifetime. 2. Practitioners are advised to go through the steps of risk assessment, risk control and risk management planning.
10. Author Response	None.

Table 59. Details for Review categorised as PRO: #121

Characteristic	**Values**
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Empirical evidence in global software engineering: A systematic review
3. Citation	[84]
4. DARE Score	3.0
5. Strength of Evidence	No assessment was made.
6. #Primary Studies	56 (37 explicit industry studies, 16 academic, 3 not stated)
7. Synthesis used	Narrative synthesis
8. Findings	1. Trust, cohesiveness and effective teamwork can be achieved through F2F meetings, temporal colocation and exchange visits - but entail extra costs. 2. Greater awareness and process transparency can be achieved through the use of a centralised repository and common configuration management tool support - but requires overcoming heterogeneity. 3. Trust and cohesiveness can be improved through effective and frequent synchronous communications—but entail extra costs. 4. Effective communications can be achieved if infrastructure is reliable and communications media are rich. 5. Effective teamwork can be achieved through synchronous interaction—but requires temporal proximity . 6. Effective teamwork can be achieved through task distribution based on architectural decoupling and low dependencies across remote locations—but requires full transition of parts of the work. 7. Early feedback and capability evaluation can be achieved through the use of incremental short-cycle development—but requires frequent and transparent communications. There is still no recipe for successful and efficient performance in globally distributed software engineering.
9. Recommendations	None.
10. Author Response	The authors observe that since key practices that help minimise risk require additional investments, global collaboration might not be suitable for companies that enter global projects to reduce costs.

Table 60. Details for Review categorised as PRO: #175

Characteristic	**Values**
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review
3. Citation	[46]
4. DARE Score	3.5
5. Strength of Evidence	No assessment was made. (Provides counts of papers identifying specific barriers.)
6. #Primary Studies	77 (All explicit industry studies)
7. Synthesis used	Thematic analysis
8. Findings	1. Barriers vary with organisation size. These are summarised in Table 7. The one common barrier is ‘language and cultural barriers’. 2. Viewed over two decades, different barriers have ‘risen’ and ‘fallen’ in importance.
9. Recommendations	1. Outsourcing vendors should focus on the identified barriers in order to have a positive impact on outsourcing clients and to win outsourcing contracts: language and cultural barriers. 2. Vendors should focus on the barriers identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: country instability. 3. Vendors should focus on the barrier identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: lack of project management. 4. Vendors should focus on the barriers identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts: lack of protection for IPR. 5. Vendors should focus on the barriers identified in order to have a positive impact on outsourcing clients and to win outsourcing contracts:lack of technical capability.
10. Author Response	The authors agreed with our extracted data.

Table 61. Details for Review categorised as PRO: #244

Characteristic	**Values**
1. Knowledge Unit	PRO.pp (Project Planning and Tracking)
2. Title	Analogy-based software development effort estimation: A systematic mapping and review
3. Citation	[41]
4. DARE Score	5.0
5. Strength of Evidence	No assessment was made. (Lists studies identifying specific factors.)
6. #Primary Studies	61 (all were explicitly industrial)
7. Synthesis used	Narrative synthesis
8. Findings	1. ASEE methods tend to yield acceptable estimates. 2. ASEE methods outperform regression based methods. 3. ASEE methods outperform ANN based methods. 4. ASEE methods outperform DT based methods. 5. One ASEE technique alone may not be the best estimation method in all contexts. However, in any context, an appropriate effort estimation model can be built by combining an ASEE technique with other techniques to overcome the weaknesses." 6. The results suggest overall that the estimation accuracy of ASEE methods is improved when used in combination with other techniques, especially FL and GA. As has been found, SM improves the accuracy of ASEE techniques much less than the other techniques. This suggests that using ML rather than non ML techniques in combination with analogy would be preferable, in particular, fuzzy logic, genetic algorithms, the model tree, and the collaborative filtering. The limited number of studies on ASEE methods combined with these techniques may account for these inconclusive results. 7. Taking into consideration the number of evaluations and based on the median of the MMRE, MT is the technique that improves the accuracy of ASEE methods the most (59.42% improvement), followed by CF combined with RSA (51.85%) and LSR (41.03%). Based on the median of the MdMRE, MT has the greatest impact (67.75%), followed by FL combined with GRA (40.80%) and GA (37.93%). Based on the arithmetic median of Pred(25), ASEE techniques are improved the most by MT (129.01% improvement), followed by CF (108.33%) and GA (100.00%). 8. Results suggest overall that all the techniques listed in Section 3.5 improve the estimation accuracy of ASEE methods, especially GA and FL, which are supported by 4 studies each. There is much less improvement in the accuracy of ASEE techniques when combined with SM.
9. Recommendations	None.
10. Author Response	The authors agreed with our extracted findings.