Chapter 14

Evidence to Practice: Knowledge Translation and Diffusion

The preceding chapters making up Part I of this book have addressed the various issues concerned with *adapting* the practices of the evidence-based paradigm to the needs of software engineering. In particular, they have described the role of a systematic review in amassing and synthesising evidence related to software engineering topics. So in this, the final chapter of Part I, we consider what should happen *after* the systematic review, and in particular, how the outcomes from a review (the *data*) can be interpreted to create *knowledge* that can then be used to guide practice, to help set standards, and to assist policy-making.

In other disciplines that make use of systematic reviews, this process of interpretation for practical use is often termed *Knowledge Translation* (KT), although as we will see, there are questions about how appropriate the "translation" metaphor is. Clearly, the way that KT is performed should itself be as systematic and repeatable as possible, and it should also reflect the needs and mores of practitioners, as well as of the different forms of organisational context within which they work.

In the interpretation of evidence-based practices for software engineering provided in Section 2.3, Step 4 was described as:

> *Integrate the critical appraisal with software engineering expertise and stakeholders' values.*

This essentially describes the role of KT, and while this does occur (we will examine some examples later in the chapter), the processes used tend to be rather *ad hoc* and to lack adequate documentation.

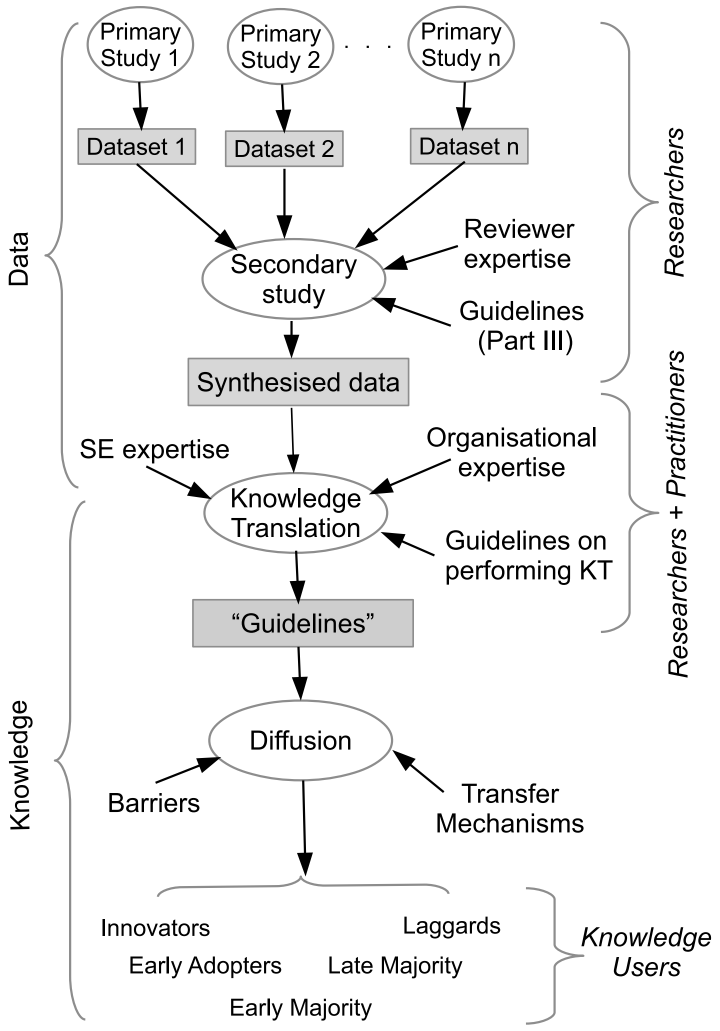

FIGURE 14.1: The pathway from data to knowledge.

Knowledge translation in itself is of course only part of the overall process of encouraging practitioners and others to make use of the evidence from a systematic review. Knowledge needs to be disseminated to be useful, and the processes through which new forms of knowledge become accepted and adopted by the relevant parts of society has been studied for many years, with the classic work on this being the book *Diffusion of Innovations* by Rogers (2003). The model shown in Figure 14.1 illustrates the sequence of quite complex processes involved in turning data into something that forms part of the professional's "knowledge toolkit".

Both KT and diffusion are large and complex topics, and we can only address them fairly briefly here. So, in this chapter we examine how KT is organised in other disciplines; discuss how it might be placed on a more systematic basis in software engineering; and review a number of examples of where KT has been performed to provide guidelines about software engineering practices. We also provide a short discussion of the nature of knowledge diffusion, and how this may occur for software engineering. Finally, we review the software engineering knowledge that has emerged from the first ten years of performing systematic reviews in software engineering and consider how this might help to inform and underpin better quality teaching, practice and research—which after all is the purpose of EBSE.

14.1 What is knowledge translation?

While there is quite an extensive literature exploring the concept of KT, we should observe that it is not the only term used to describe "post-systematic-review" activities. Other terms that are in use include "Knowledge To Action" (KTA) and "Knowledge Exchange" (KE), and such words as 'uptake' and 'transfer' are also used in this context. Part of the reason for the frequent use of "Knowledge Translation" appears to stem from it being a term used in the mandate of the Canadian Institutes of Health Research (Straus, Tetroe & Graham 2009), and because Canadian researchers have authored many papers on this topic.

A useful definition of KT is that produced by the *World Health Organisation* (WHO) in 2005, as:

> "the synthesis, exchange and application of knowledge by relevant stakeholders to accelerate the benefits of global and local innovation in strengthening health systems and advancing people's health" (WHO 2005)

(We might note that this actually refers to the 'exchange' of knowledge.) Other definitions are to be found, with a common thread being the emphasis on putting the knowledge into use.

The use of guidelines for performing KT so as to produce recommendations for practice has been investigated extensively for both clinical medicine and education. Rather confusingly, in the literature, both the procedures for producing recommendations and the recommendations themselves are apt to be referred to as *guidelines*. So in this chapter we will use *KT recommendations* for the guidance provided to the eventual users about how to interpret the outcomes from an individual systematic review. Similarly, wherever we refer to *KT guidelines*, these will refer to the set of *activities* used for deriving recommendations for practice or policy (KT recommendations), along with any guidance that might be provided about how to describe them.

An overview of how KT-related guidelines are used in clinical medicine is provided by the EU recommendations for drawing up KT guidelines on best medical practices (Mierzewski 2001). This also reviews the KT guidelines programmes used in different EU member states.

Within the UK, the National Institute for Clinical Excellence (NICE) has produced its own KT guidelines (NICE 2009), and these also provide a useful source of descriptions of translation models. International efforts towards evaluating KT guidelines produced by different organisations have included the AGREE II programme for *appraisal* of KT guidelines (Burgers, Grol, Klazinga, Mäkelä & Zaat 2003, AGREE 2009) and the assessment of these processes for the World Health Organisation (WHO) described in (Schünemann, Fretheim & Oxman 2006). Together these provide systematic approaches, for both producing KT recommendations, and also for evaluation of the procedures involved.

As noted earlier, the literature on this topic is extensive, which emphasises that, for healthcare, the process of translation is complicated by many factors. For example, Zwarenstein & Reeves (2006) observe that KT is often directed at producing KT recommendations for a single professional group, whereas the treatment of patients is likely to involve inter-professional collaboration. Similarly, Kothari & Armstrong (2011) observe that for KT research in more general health care, "developing processes to assist community-based organizations to adapt research findings to local circumstances may be the most helpful way to advance decision-making in this area".

We should also note that the appropriateness of this terminology has been challenged. Greenhalgh & Wieringa (2013) argue that the 'translation' metaphor is an unhelpful one and that its use "constrains thinking". Essentially, they argue that this term implicitly creates a model in which the only form of useful knowledge stems from "objective, impersonal research findings". In examining equivalent metaphors from other disciplines they emphasise the need to also involve such factors as "tacit knowledge of the wider clinical and social" situation when using such knowledge. In particular, they propose that a wider set of metaphors should be used, including ones such as "knowledge intermediation".

So, what their work highlights is that there are dangers implicit in simply adopting the 'translation' metaphor with its implication of researchers "handing down" scientifically distilled guidance. More realistically, the process of developing guidelines for use should be something that is shared between researchers and other stakeholders (which reiterates the earlier point about the use of 'exchange' in the definition from the WHO). And, taken together, what all of these studies also indicate is that systematising KT is a specific research activity in its own right, and that the process of KT involves much more than simply supplying the outcomes from systematic reviews to professionals.

14.2 Knowledge translation in the context of software engineering

In this section we examine how the activities of KT could be interpreted for software engineering. As a starting point, we have adapted the description of KT provided by the WHO, quoted in the previous section, as well as the variation used in Davis, Evans, Jadad, Perrier, Rath, Ryan, Sibbald, Straus, Rappolt, Wowk & Zwarenstein (2003), in order to define a process of KT for software engineering as being:

> The exchange, synthesis and ethically sound application of knowledge—within a complex system of interactions between researchers and users—to accelerate the capture of the benefits of research to help create better quality software and to improve software development processes.

The three key elements involved in achieving this are: the *outcomes* of a systematic review; the set of *interpretations* of what these outcomes mean in particular application contexts; and the forms appropriate for *exchanging* these interpretations with the intended audience.

In health care, a widely-cited paper by Graham, Logan, Harrison, Straus, Tetroe, Caswell & Robinson (2006) refers to this as "knowledge to action" (a term which we noted earlier). In this, the authors suggest that the process of KT (or KTA) can be described using a model of two nested and interlocked cycles that are respectively related to knowledge creation and knowledge application. Figure 14.2 shows how this concept can be interpreted for software engineering (Budgen, Kitchenham & Brereton 2013). The inner *knowledge cycle* is concerned with "knowledge creation" (which in the case of software engineering will be based upon primary studies and systematic reviews). The outer *action cycle* "represents the activities that may be needed for knowledge application", including the creation and evaluation of both KT guidelines and KT recommendations produced by using these. This element of the KT process is highly-interactive and holistic (Davis et al. 2003), being driven by the needs of the given topic, and hence evaluation will need to be a key element in maintaining consistency of practice.

Because of this, the positioning of the elements in the outer cycle should not be regarded as forming a sequence in the same way as occurs for the elements of the inner cycle. Rather, the outer cycle describes a set of activities that may well be interleaved and iterative.

Indeed, the value of Figure 14.2 lies less in its structure than in its identification of the various factors involved in performing KT. It also highlights the point that a transition to an evidence-informed approach to software engineering can only be achieved through a partnership of researchers, practitioners, and policy-makers.

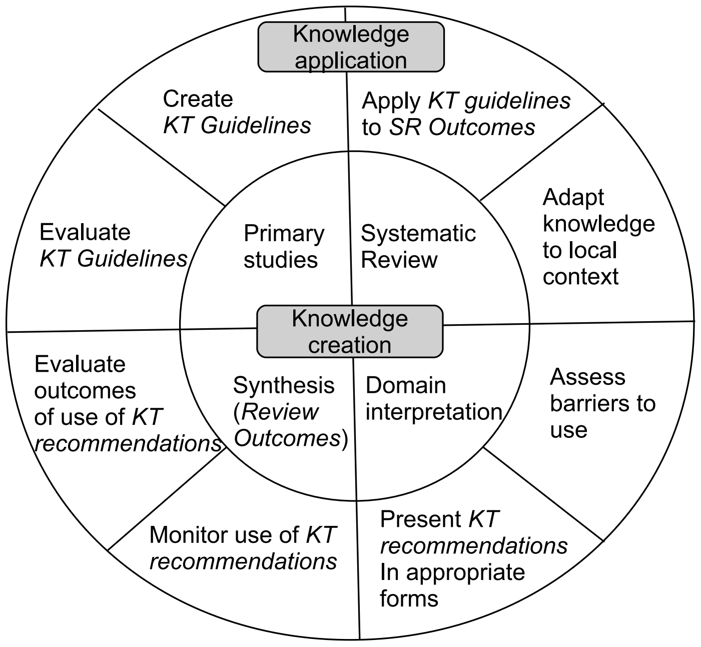
Knowledge application
Create KT Guidelines
Apply KT guidelines to SR Outcomes
Adapt knowledge to local context
Evaluate KT Guidelines
Primary studies
Systematic Review
Knowledge creation
Evaluate outcomes of use of KT recommendations
Synthesis (Review Outcomes)
Domain interpretation
Assess barriers to use
Monitor use of KT recommendations
Present KT recommendations In appropriate forms

FIGURE 14.2: A knowledge translation model for SE. Reproduced with permission.

To conclude this discussion of KT, in the book by Khan, Kunz, Kleijnen & Antes (2011), the authors make some useful observations about the form that recommendations should have that are as relevant for software engineering as they are for clinical medicine. In particular, they highlight the following key points.

- Recommendations should convey a clear message and should be as simple as possible to follow in practice.
- What possible users "really want to know about recommendations is how credible (trustworthy) they are", noting too that 'credibility of a recommendation depends only in part on the strength of evidence collated from the review'.

In particular, they suggest classifying any recommendations as being either strong or weak. They define a *strong recommendation* as effectively forming a directive to adopt a practice or treatment, whereas a *weak recommendation* indicates that a decision about its adoption is something that needs to depend upon due consideration of other relevant factors.

Another approach to categorising the quality of evidence and strength of recommendations from a systematic review, again developed in a medical context, is the GRADE system (Grades of Recommendation, Assessment, Development and Evaluation), described in (GRADE Working Group 2004). A discussion and example of using GRADE in a software engineering context can be found in the paper by Dybå & Dingsøyr (2008*b*). To employ GRADE they used four factors: study design; study quality; consistency ("similarity of estimates of effect across studies"), and directness ("the extent to which the people, interventions and outcome measures are similar to those of interest").

The GRADE scheme specifies four levels for strength of evidence, and hence of recommendations. In Table 14.1 we provide the GRADE descriptions for each level, and suggest how these might be interpreted in a software engineering context. As expressed, this allows for the possibility that further studies may increase the classification assigned to any recommendations.

Evidence-based studies in software engineering are still far from reaching the level of maturity where strong recommendations can be generated with any confidence. Indeed, the nature of the effects that occur for the creative processes of software engineering is likely to make the generation of strong recommendations a relatively rare occurrence, indeed, it is worth noting that Dybå & Dingsøyr assessed the strength of evidence from their earlier study on agile methods as 'very low'.

TABLE 14.1: Strength of Evidence in the GRADE System.

Level	GRADE definition	SE Interpretation
High	Further research is very unlikely to change our confidence in the estimation of effect.	Supported by significant results from more than one good quality systematic review as well as by experiences from systematically conducted field studies.
Moderate	Further research is likely to have an important impact on our confidence in the estimate of effect and may change the estimate.	Supported by moderately significant results from at least one good quality systematic review, and by observational studies.
Low	Further research is likely to have an important impact on our confidence in the estimate of effect and is likely to change the estimate.	Supported by moderately significant results from at least one good quality systematic review.
Very Low	Any estimate of effect is very uncertain.	Only supported by results from one systematic review.

So, in the next section we examine some examples of where systematic reviews have led to the creation of some form of recommendations about how the outcomes might be used, and how the authors have qualified these.

14.3 Examples of knowledge translation in software engineering

The catalogue of 143 published systematic reviews described in (Budgen, Drummond, Brereton & Holland 2012) identified 43 reviews that contained material that could be used to inform teaching and practice. However, only three of these actually provided any form of *recommendations* about how the outcomes from the review could be interpreted in terms of practice. So, in this section we briefly describe each of the studies, the recommendations that they made, and how these were derived (where known).

14.3.1 Assessing software cost uncertainty

Cost and effort estimation have been studied quite extensively, which is perhaps not surprising given that they can have a major impact upon project success (and company profit). The study by Jørgensen (2005) particularly looked at uncertainty in forecasts, and some of the likely causes for this.

This particular study drew upon primary studies that came from both the software engineering domain and also a range of other domains that employ forecasting procedures. The paper presents a set of recommendations (termed 'guidelines' in the paper) derived from the analysis. For each recommendation, the paper identifies both the primary studies that provide supporting evidence and also identifies where this is discussed. The author also provides a rating using the same terms that we use above, together with explanations of what these mean in terms of the evidence provided.

Rather than present the whole set of seven recommendations here, we simply give two examples from them.

Recommendation 1: "Do not rely solely on unaided, intuition-based processes." This is rated as being *strong*, based upon the number of studies favouring it.

Recommendation 2: "Do not replace expert judgement with formal models." This is rated as *medium.*

Indeed, although the author describes the rating process as informal and subjective, it is actually quite systematic and provides an excellent model for use by other authors.

It is also interesting to note that the conclusions from a more recent summary of work in this area suggest that most of these recommendations are probably still valid (Jørgensen 2014*b*).

14.3.2 Effectiveness of pair programming

The meta-analysis on the effectiveness of pair programming described by Hannay et al. (2009) found wide variation in the form and organisation of the primary studies included, limiting the confidence with which any recommendations could be produced. There are also many other factors that influence whether or not such a technique might be considered effective, particularly the expertise of the programmers and the complexity of the task involved.

However, they did suggest that two recommendations were appropriate when pair programming was being used by "professional software developers", and in a context where "you do not know the seniority or skill levels of your programmers, but do have a feeling for task complexity". In this context they suggested that it was appropriate to employ pair programming for either of the following situations:

- When task complexity is low and time is of the essence
- When task complexity is high and correctness is important

No specific process for deriving these was described, although the analysis of the data implicitly supported them. In terms of the classifications suggested in Table 14.1 these should probably be considered as recommendations with *low* strength. However, in the future, they could be regraded as *moderate* if supporting outcomes from observational studies become available, given that this was a 'good' systematic review.

14.3.3 Requirements elicitation techniques

Our third example is a paper by Dieste & Juristo (2011) that examines elicitation techniques that are often used for determining system requirements. They present five recommendations (again, termed 'guidelines' in the paper), and for each one they identify the aggregated (synthesised) evidence that supports or refutes the recommendation. The authors have not attempted to assess the strength of these recommendations.

Again, we present two of these without attempting to include all of the supporting detail. We have also slightly reworded them, mainly to fit the role of a recommendation.

Recommendation 1: "The use of unstructured interviews is equally as, or more *effective* than, using introspective techniques (such as protocol analysis) and sorting techniques." The authors observe that it is reasonable to assume that this recommendation also applies to structured interviews.

Recommendation 3: "The use of unstructured interviews is less *efficient* than using sorting techniques and Laddering, but is as efficient as introspective techniques such as protocol analysis." Again, the authors observe that this should also apply to structured interviews.

One of the benefits (and complexities) of this paper was that it looked at studies that made comparisons between the different techniques, allowing the reviewers to provide an element of ranking in their recommendations.

14.3.4 Presenting recommendations

Looking at these three examples, we can see some common threads among them.

- The authors are experts at performing systematic reviews and have extensive expertise related to the topic of the review, assisting them with interpreting the outcomes of the review.
- Their reviews found quite substantial numbers of primary studies, so that the authors have been able to identify areas where these reinforced each other (or vice versa).
- They present supporting evidence for their recommendations, in two cases, directly listing the studies that agree/disagree with the recommendation. They also provide a discussion that explains how the primary studies support a recommendation (or otherwise).

All three examples also follow the advice of Khan et al. (2011) to keep the recommendations simple and to provide some indication of how trustworthy they are (in these cases, by discussing the underpinning evidence).

So, where a review team has the technical expertise to do so, there is scope to provide at least a basic element of knowledge translation of the review outcomes, in the form of recommendations. We provide a summary of key points for doing so in the box below.

Guidelines for Producing Recommendations

- Only do so if you have appropriate technical expertise.
- Only do so if your systematic review is a 'strong' one.
- Keep any recommendations simple and easy to follow.
- Identify the studies that support/refute each recommendation.
- Provide a separate derivation, related to the studies.
- Provide an indication of how strong a recommendation is (and what you mean by this).
- Identify the audience for the recommendation, and, where appropriate, whether the evidence comes from using practitioners or students as participants in the primary studies.

14.4 Diffusion of software engineering knowledge

Neither innovativeness nor quality will necessarily ensure that new devices or processes will be successful in being accepted by the communities most likely to benefit from them. So, even if we can produce strong recommendations that address topics of major importance to the software engineering community, their adoption still requires the community to be persuaded of their merits.

This situation is by no means unique to software engineering, and the terminology that is used at the base of Figure 14.1 is drawn from the ideas of *diffusion research* as set out by Rogers (2003). This is based upon the premise that the process of acceptance of innovative ideas and technologies tends to follow broadly similar processes, regardless of discipline. The topic overall is a large one and we can only touch lightly upon it here, where our main concern is to encourage *awareness*. The world does not automatically beat a path to the doorway of the person with a better mousetrap, agricultural practice or software development process. To gain recognition, that person needs to ensure that knowledge about their innovation gets to, and is accepted by, the people who will influence others.

The 'classical' diffusion model produced by Rogers recognises five major 'adopter categories' who are involved in the process of moving an innovation into the mainstream. Briefly, these are as follows.

- The *innovators* are people who like to try new ideas and are willing to take a high degree of risk in doing so. Communication between them is a strong element in sharing of new ideas, but it is likely that this will be across organisations rather than within them.
- The *early adopters* are opinion formers who have influence within organisations, and so they are the people whose opinion is sought by others who are considering change.
- The *early majority* are those who are more cautious than the preceding two groups, but still tend to be ahead of the average. They take longer to decide about changing their processes than the early adopters, and tend to follow rather than lead.
- The *late majority* are even more cautious, and only join in when they can see that their peers are taking up a change, and that they might even be disadvantaged by not doing so.
- Finally, the *laggards* are apt to be suspicious of change and may have only limited resources, which may also encourage caution about change.

So generally, the key to successful adoption lies in achieving buy-in from the people who fall into the first two categories.

An authoritative study in the field of health care by Greenhalgh, Robert, MacFarlane, Bate & Kyriakidou (2004), based upon a large-scale systematic review, suggests a wider and rather more proactive view of the way that knowledge can be transferred. In particular the authors suggest that it is useful to distinguish between the following three mechanisms.

1. *diffusion* where knowledge and awareness spread passively through a community, largely through natural means
2. *dissemination* through the active communication of ideas to a target audience
3. *implementation* through the use of communication strategies that are targeted at overcoming barriers, using administrative and educational techniques to make the transfer more effective

For recommendations produced from systematic reviews in software engineering, all three mechanisms can potentially play a useful role.

Pfleeger (1999) suggests that different adopter categories are motivated by distinct transfer mechanisms. She introduces the idea of the *gatekeeper* whose role is to "identify promising technologies for a particular organisation". Another part of their role is to assess the evidence presented for a new or changed technology. We should also note that identifying appropriate vehicles for communicating with the different gatekeeper roles is important, These vehicles might be social networks (particularly for innovators), trusted media sources (for early adopters) and some form of 'packaging' (which might be the incorporation of recommendations into standards) for the early and late majorities.

An associated issue here is that of *risk*. For the innovators and early adopters, taking up a new or changed technology may involve a higher level of risk. So any presentation of new knowledge has to help them make an assessment of how significant this risk might be in their particular context (and in exchange, what benefits they might derive).

While we cannot really delve deeper into these issues here, they are important ones for systematic reviewers to note. Publication of the outcomes of a review in respected refereed journals is only the first stage of getting knowledge out to users. In particular, the knowledge embodied in any recommendations may need to be spread through such means as social media, professional journals, and incorporation into standards. Another important vehicle is the use of educational channels (relating to 'implementation'), and we address this in a little more detail in the next, and final, section of this chapter.

14.5 Systematic reviews for software engineering education

One way in which software engineering does differ significantly from other disciplines that use systematic reviews is in the way that the topics of these are decided. For disciplines such as education, social science, and to some degree, healthcare, systematic reviews are often commissioned by policy-makers, who may well work within government agencies. To our best knowledge, there are so far no instances of systematic reviews in software engineering being commissioned by either government agencies or industry—this may be partly because secondary studies are still immature for software engineering, but also because IT-related decisions are rarely evidence-informed at any level. So, the available secondary studies do tend to have been motivated more by the interest of particular researchers than through any efforts to inform decision-making.

So, in presenting this 'roadmap' to available systematic reviews, the reader should remember that the distribution of topics is driven by 'bottom-up' interest from researchers, rather than 'top-down' needs of policy-makers. This may change in the future but for the present it is the situation that exists for software engineering.

And of course, even before we have finished compiling such a roadmap, it is inevitably out of date as new reviews become available.

14.5.1 Selecting the studies

The roadmap provided here is based upon a *tertiary study* that we performed in 2011, and published as (Budgen et al. 2012). The research question posed for this was:

> *What is available to enable evidence-informed teaching for software engineering?*

Although this was posed as an educational question, we did seek to identify any systematic reviews that could provide knowledge, advice or guidance relevant to either practice or teaching. We excluded any that were purely concerned with research issues (and of course, almost all mapping studies).

To identify candidate systematic reviews we used a two-part search procedure, organised as follows.

1. List all systematic reviews found in the three published 'broad' tertiary studies that were available to us (Kitchenham, Brereton, Budgen, Turner, Bailey & Linkman 2009, Kitchenham, Pretorius, Budgen, Brereton, Turner, Niazi & Linkman 2010, da Silva et al. 2011). Together, these covered the period up to the end of 2009. We also included one paper that was subsequently known to have been missed by these studies.

2. List the systematic reviews found in five major software engineering journals between the start of 2010 and mid-2011. While recognising that this would be incomplete, it was felt that it would identify the majority of published studies for this period.

Together, this produced a set of 143 secondary studies.

We then excluded any studies that addressed research trends, those with no analysis of the collected data and those that were not deemed to be relevant to teaching. Conversely, we included studies that covered a topic considered to be appropriate for a software engineering curriculum, using *Knowledge Areas* (KAs) and *Knowledge Units* (KUs) from the 2004 ACM/IEEE guidelines for undergraduate curricula in software engineering. This left us with 43 secondary studies.

Our data extraction procedure then sought to categorise these studies against the KAs and KUs, to extract any recommendations provided by the authors, and in the absence of these, any that we felt were implied by the outcomes, since few authors provide explicit recommendations. (As we have seen, *knowledge translation* is not a trivial task.) Data extraction was performed by pairs of analysts using different pairings of the four authors in order to reduce possible bias.

14.5.2 Topic coverage

Table 14.2 provides a count of the number of studies we categorised against each Knowledge Area.

TABLE 14.2: Number of Systematic Reviews for Each Knowledge Area

KA code	Topic	Count
QUA	Software Quality	6
PRF	Professional Practice	2
MGT	Software Management	13
MAA	Modeling & Analysis	7
DES	Software Design	1
VAV	Validation & Verification	7
EVO	Software Evolution	2
PRO	Software Process	5
	Total Studies	43

Within these numbers, there are some substantial groupings for particular Knowledge Units. In particular, nine of the studies classified as MGT were in the area of project planning, with a preponderance of cost estimation studies among these. (As this is an important topic, and one that teachers may not always be particularly expert in, this can of course be seen as a useful grouping.)

Fuller details of the studies are provided in Appendix A.