Chapter 1

The Evidence-Based Paradigm

1.1 What do we mean by evidence? 4
1.2 Emergence of the evidence-based movement 7
1.3 The systematic review 10
1.4 Some limitations of an evidence-based view of the world 14

Since this is a book that is about the use of evidence-based research practices, we feel that it is only appropriate to begin it by considering what is meant by *evidence* in the general sense. However, because this is also a book that describes how we acquire evidence about software engineering practices, we then need to consider some of the ways in which ideas about evidence are interpreted within the rather narrower confines of science and technology.

Evidence is often associated with *knowledge*. This is because we would usually like to think that our knowledge about the world around us is based upon some form of evidence, and not simply upon wishful thinking. If we go to catch a train, it might be useful to have evidence in the form of a timetable that shows the intention of the railway company to provide a train at the given time that will take us to our destination. Or, rather differently, if we think that some factor might have caused a 'population drift' away from the place where we live, we might look at past census data to see if such a drift really has occurred, and also whether some groups have been affected more than others. Of course the link between evidence and knowledge is rarely well-defined, as in our second example, where any changes in population we observe might arise from many different factors. Indeed, it is not unusual, in the wider world at least, for the same evidence to be interpreted differently (just think about global warming).

In this chapter we examine what is meant by evidence and knowledge, and the processes by which we interpret the first to add to or create the second. We also consider some limitations of these processes, both those that are intrinsic, such as those that arise from the nature of the things being studied, and of data itself, and also those that arise from the inevitable imperfections of research practice. In doing so, we prepare the ground for Chapter 2, where we look at how the discipline of software engineering interprets these concepts, and review the characteristics of software engineering that influence the nature of our evidence—and hence the nature of our knowledge too.

1.1 What do we mean by evidence?

As noted above, evidence can be considered as being something that underpins *knowledge*, and we usually expect that knowledge will be derived from evidence through some process of *interpretation.* The nature of that interpretation can take many forms. For example, it might draw upon other forms of knowledge, as when the fictional detective Sherlock Holmes draws upon his knowledge about different varieties of tobacco ash, or about the types of earth to be found in different parts of London, in order to turn a clue into evidence. Interpretation might also be based upon mathematical or statistical procedures, such as when a scientist gathers together different forms of experimental and observational data—for example, using past medical records to demonstrate that smoking is a cause of lung cancer. Yet another, less scientific, illustration of the concept is when the jury at a criminal trial has to consider the evidence of a set of witnesses in order to derive reasonable knowledge about what actually happened. Clearly these differ in terms of when they arise, the form of knowledge derived, and the rigour of the process used for its derivation (and hence the *quality* of the resulting knowledge). What they do have in common though, is that our confidence about the knowledge will be increased if there is more than one source (and possibly form) of evidence. For the fictional detective, this may be multiple clues; for the clinical analysis it might involve using records made in many places and on patients who have different medical histories; for the jury, it may be that there are several independent witnesses whose statements corroborate each other. This process of *triangulation* between sources (a term derived from navigation techniques) is also an important means of testing the *validity* of the knowledge acquired.

Science in its many forms makes extensive use of these concepts, although not always expressed using this vocabulary. Over the years, particular scientific disciplines have evolved their own accepted set of empirical practices that are intended to give confidence in the validity and quality of the knowledge created from the forms of evidence considered to be appropriate to that discipline, and also to assess how strong that confidence is. Since this book is extensively concerned with different forms of *empirical* study, this is a good point to note that such studies are ones that are based upon *observation* and *measurement.* Indeed, this is a reminder that, strictly speaking, scientific processes never 'prove' anything (mathematics apart), they only 'demonstrate' that some relationship exists between two or more factors of interest. Even physicists, who are generally in the best position to isolate factors, and to exclude the effect of the observation process, are confronted with this issue. The charge on an electron, or the universal gravitational constant, may well be known to a very high level of precision, and with high confidence, but even so, some residual uncertainty always remains. For disciplines where it can be harder to separate out the key experimental characteristics and where (horrors), humans are involved in roles other than as observers, so the element of variability will inevitably increase. This is of course the situation that occurs for many software engineering research studies, and we will look at some of the consequences in the next chapter.

When faced with evidence for which the values and quality may vary, the approach generally adopted is to use repeated observations, as indicated above, and even better, to gather observations made by different people in different locations. By pooling these, it becomes easier to identify where we can recognise repeated occurrences of patterns in the evidence that can be used to provide knowledge. This repetition also helps to give us confidence that we are not just seeing something that has happened by chance.

The assumption that it is meaningful to aggregate the observations from different studies and to seek patterns in these is termed a *positivist* philosophy. Positivism is the philosophy that underpins the 'scientific method' in general, as well as almost all of the different forms of empirical study that are described in this book.

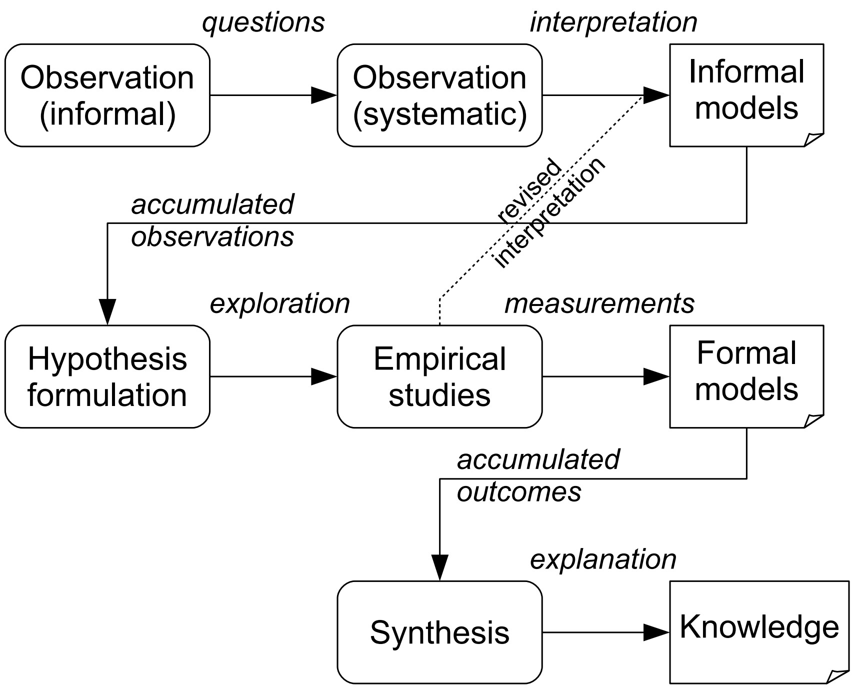

FIGURE 1.1: A simple model of knowledge acquisition.

Figure 1.1 shows a simple model that describes how these concepts relate to one another in a rather general sense. The top row represents how, having noticed the possible presence of some effect, we might begin gathering observations to create a rather informal model to describe some phenomenon. This model might well identify more than one possible cause. If this looks promising, then we might formulate a hypothesis (along the lines that "factor X causes outcome Y to occur") and perform some more systematically organised studies to explore and test this model, during which process, we may discard or revise our ideas about some of the possible causes. Finally, to confirm that our experimental findings are reliable, we encourage others to repeat them, so that our knowledge is now accumulated from many sources and gathered together by a process that we refer to as *synthesis*, so that the risk of bias is reduced. Many well-known scientific discoveries have followed this path in some way, such as the discovery of X-rays and that of penicillin.

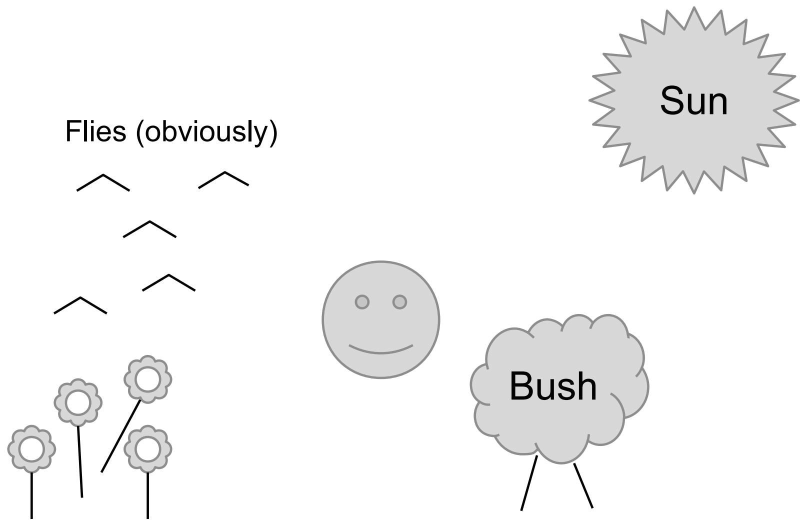

FIGURE 1.2: Does the bush keep the flies off?

Since this is rather abstract, let's consider a simple (slightly contrived but not unrealistic) example. This is illustrated (very crudely) in Figure 1.2. If we imagine that, while sitting out in a garden one day in order to enjoy the summer sunshine, we notice that we are far less bothered by flies when sitting near a particular bush, then this provides an example of informal observation. If we get enough good weather (we did say this example was contrived), we might try repeating the observation, perhaps by sitting near other bushes of that variety. If we continue to notice the effect, then this now constitutes an informal model. Encouraged by visions of the royalties that could arise from discovering a natural insecticide, we might then go on to pursue this rather more systematically, and of course, in so doing we will probably find all sorts of other possible explanations, or indeed, that it is not really an effect at all. But of course, we might also just end up with some systematically gathered knowledge about the insect-repellent nature of this plant (or perhaps, of this plant in conjunction with other factors).

This book is mainly concerned with the bottom two layers of the model shown in Figure 1.1. In Part I and Part III we are concerned with how knowledge from different sources can be 'pooled', while in Part II we provide a subject-specific interpretation of what is meant by the activities in the middle layer. In particular, we will be looking at ways of gathering evidence that go beyond just the use of formal experiments.

In the next section we examine how the concepts of *evidence-based* knowledge and of *evidence-informed* decision-making, have been interpreted in the 20th and 21st centuries. In particular, we will discuss the procedures that have been adopted to produce evidence that is of the best possible quality.

1.2 Emergence of the evidence-based movement

It is difficult to discuss the idea of evidence-based thinking without first providing a description of how it emerged in clinical medicine. And in turn, it is difficult to categorise this as other than a movement that has influenced the practice and teaching of medicine (and beyond). At the heart of this lies the *Cochrane Collaboration*[1], named after one of the major figures in its development. This is a not-for-profit body that provides both independent guardianship of evidence-based practices for clinical medicine, and also custodianship of the resulting knowledge.

So, who was Cochrane? Well, Archie Cochrane was a leading clinician, who became increasingly concerned throughout his career about how to know what was the best treatment for his patients. His resulting challenge to the medical profession was to find the most effective and fairest way to evaluate available medical evidence, and he was particularly keen to put value upon evidence that was obtained from randomised controlled trials (RCTs). Cochrane's highly influential 1971 monograph "Effectiveness and Efficiency: Random Reflections on Health Services" (Cochrane 1971) particularly championed the extensive use of randomisation in RCTs, in order to minimise the influence of different sources of potential bias (such as trial design, experimenter conduct, allocation of subjects to groups, etc.). Indeed, he is quoted as saying that "you should randomise until it hurts", in order to emphasise the critical importance of conducting fair and unbiased trials.

Cochrane also realised that even when performed well, individual RCTs could not be relied upon to provide unequivocal results, and indeed, that where RCTs on a given topic were conducted by different groups and in different places, they might well produce apparently conflicting outcomes. From this, he concluded in 1979 that "it is surely a great criticism of our profession that we have not organised a critical summary by speciality or subspeciality, adapted periodically, of all relevant randomised controlled trials".

Conceptually, this statement was at complete variance with accepted scientific practice (not just that in clinical medicine). In particular, the role of the *review paper* has long been well established across much of academia, with

[1]www.cochrane.org

specialist journals dedicated to publishing reviews, and with an invitation to write a review on a given topic often being regarded as a prestigious acknowledgement of the author's academic standing. However, a problem with this practice was (and still is) that two people who are both experts on a given topic might well write reviews that draw contrasting conclusions—and with each of them selecting a quite different set of sources in support of their conclusion.

While this does not mean that an expert review is necessarily of little value, it does raise the question of how far the reviewer's own opinions may have influenced the conclusions. In particular, where the subject-matter of the review requires interpretation of empirical data, then how this is selected is obviously a critical parameter. A widely-quoted example of this is the review by Linus Pauling in his 1970 publication on the benefits of Vitamin C for combatting the common cold. His 'cherry-picking' of those studies that supported his theory, and dismissal of those that did not as being flawed, produced what is now regarded as an invalid conclusion. (This is discussed in rather more depth in Ben Goldacre's book, *Bad Science* (2009), although Goldacre does observe that in fairness, cherry-picking of studies was the norm for such reviews at the time when Pauling was writing—and he also observes that this remains the approach that is still apt to be favoured by the purveyors of 'alternative' therapies.)

Finding the most relevant sources of data is, however, only one element in producing reviews that are objective and unbiased. The process by which the outcomes (findings) from those studies are *synthesised* is also a key parameter to be considered. Ideas about synthesis have quite deep roots—in their book on literature reviews, Booth, Papaioannou and Sutton (2012) trace many of the ideas back to the work of the surgeon James Lind and his studies of how to treat scurvy on ships—including his recognition of the need to discard 'weaker evidence', and to do so by using an objective procedure. However, the widespread synthesis of data from RCTs only really became commonplace in the 1970s, when the term *meta-analysis* also came into common use[2].

Meta-analysis is a statistical procedure used to pool the results from a number of studies, usually RCTs or controlled experiments (we discuss this later in Chapters 9–11). By identifying where individual studies show consistent outcomes, a meta-analysis can provide much greater statistical authority for its outcomes than is possible for individual studies.

Meta-analysis provided one of the key elements in persuading the medical profession to pay attention. In particular, what Goldacre describes as a "landmark meta-analysis" looking at the effectiveness of an intervention given to mothers-to-be who risked premature birth, attracted serious attention. Seven

[2]One of us (DB) can claim to have had relatively early experience of the benefits of synthesis, when analysing scattering data in the field of elementary particle physics (Budgen 1971). Some experiments had suggested the possible presence of a very short-lived Σ particle, but this was conclusively rejected by the analysis based upon the composite dataset from multiple experiments.

trials of this treatment were conducted between 1972 and 1981, two finding positive effects, while the other five were inconclusive. However, in 1989 (a decade later) a meta-analysis that pooled the data from these trials demonstrated very strong evidence in favour of the treatment, and it is a “Forest Plot” of these results that now forms a central part of the logo of the *Cochrane Collaboration*, as shown in Figure 1.3[3]. With analyses such as this, supported by the strong advocacy of evidence-based decision making from David Sackett and his colleagues (Sackett, Straus, Richardson, Rosenberg & Haynes 2000), clinicians became more widely persuaded that such pooling of data could provide significant benefits. And linking all this back with the ideas about evidence, Sackett et al. (2000) defined *Evidence-Based Medicine* (EBM) as “the conscientious, explicit and judicious use of the current best evidence in making decisions about the care of individual patients”.

FIGURE 1.3: The logo of the Cochrane Collaboration featuring a forest plot (reproduced by permission of the Cochrane Collaboration).

The concept has subsequently been taken up widely within healthcare, although, as we note in Section 1.4, not always without some opposing arguments being raised. It has also been adopted in other disciplines where empirical data is valued and important, with education providing a good example of a discipline where the outcomes have been used to help determine policy as well as practice. A mirror organisation to that of the Cochrane Collaboration is the Campbell Collaboration[4], that “produces systematic reviews of the effects of social interventions in Crime & Justice, Education, International Development, and Social Welfare”. And of course, in the following chapters, we will explore how evidence-based ideas have been adopted in software engineering.

So, having identified two key parameters for producing sound evidence from an objective review process as being:

- objective selection of relevant studies
- systematic synthesis of the outcomes from those studies

[3]We provide a fuller explanation of the form of Forest Plots in Chapter 11. The horizontal bars represent the results from individual trials, with any that are to the left of the centre line favouring the experimental treatment, although only being statistically significant if they do not touch the line. The results of the meta-analysis is shown by the diamond at the bottom.

[4]www.campbellcollaboration.org

we can now move on to discuss the way that this is commonly organised through the procedures of a *systematic review.*

1.3 The systematic review

At this point, we need to clarify a point about the terminology we use in this book. What this section describes is something that is commonly described as a process of *systematic review* (SR). However, in software engineering, a commonly-adopted convention has been to use the term *systematic literature review* (SLR). This was because when secondary studies were first introduced into software engineering, there was concern that they would be confused with code inspection practices (also termed reviews) and so the use of 'literature' was inserted to emphasise that it was published studies that were being reviewed, not code.

Now that secondary studies as a key element of evidence-based software engineering (EBSE) are part of the empirical software engineer's toolbox, the likelihood of confusion seems much less. So we feel that it is more appropriate to use the more conventional term 'systematic review' throughout this book. However, we do mention it here just to emphasise that when reading software engineering papers, including many of our own, a systematic literature review is the same thing as a systematic review.

The goal of a *systematic review* is to search for and identify all relevant material related to a given topic (with the nature of this material being determined by the underlying question and the nature of the stakeholders who have an interest in it). Knowledge about that topic is then used to assist with drawing together the material in order to produce a collective result. The aim is for the procedures followed in performing the review to be as objective, analytical, and repeatable as possible—and that this process should, in the ideal, be such that if the review were repeated by others, it would select the same input studies and come to the same conclusions. We often refer to a systematic review as being a *secondary study*, because it generates its outcomes by aggregating the material from a set of *primary studies.*

Not surprisingly, conducting such a review is quite a large task, not least because the 'contextual knowledge' required means that much of it needs to be done by people with some knowledge of the topic being reviewed. We will encounter a number of factors that limit the extent to which we can meet these goals for a review as we progress through the rest of this part of the book. However, the procedures followed in a systematic review are intended to minimise the effects of these factors and so even when we don't quite meet the aim as fully as we would like, the result should still be a good quality review. (This is not to say that expert reviews are not necessarily of good quality, but they are apt to lack the means of demonstrating that this is so, in contrast to a systematic review.)

So, a key characteristic of a systematic review is that it is just that, *systematic*, and that it is conducted by following a set of well-defined procedures. These are usually specified as part of the *Review Protocol*, which we will be discussing in more detail later, in Chapter 4. For this section, we are concerned simply with identifying what it is that these procedures need to address. Figure 1.4 illustrates how the main elements of a systematic review are related once a sensible question has been chosen. Each of the ovals represents one of the processes that needs to be performed by following a pre-defined procedure. Each process also involves making a number of decisions, as outlined below.

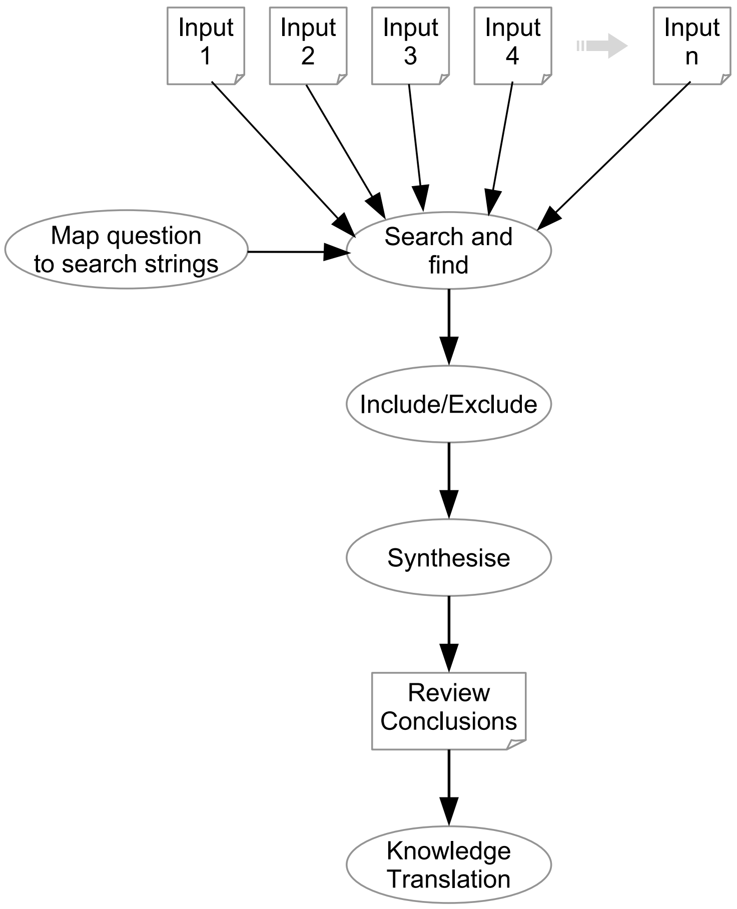

FIGURE 1.4: The systematic review process.

What searching strategy will be used? An important element of the review is to make clear *where* we will search, and *how* we will search for appropriate review material. In addition, we need to ensure that we have included all the different keywords and concepts that might be relevant. We address this in detail in Chapter 5.

What material is eligible for inclusion? This relates to both the different *forms* in which material (usually in the form of the outcomes of empirical studies) might occur, and also any characteristics that might affect its *quality*. Indeed, we often have more detailed specifications for what is to be *excluded* than for what is to be included, since we want to ensure that we don't miss anything that could be in a form that we didn't anticipate, or expect to encounter. Again, these issues will be considered more fully in Chapters 6 and 7.

How is the material to be synthesised? This addresses the analytical procedures that are to be followed. These may be fairly simple, as we explain below, or quite complex. Chapters 9, 10 and 11 consider the relevant issues for a software engineering context.

How to interpret the outcomes of the review? This is not necessarily a single process, since the outcomes might need to be interpreted differently when used in specific contexts. The processes involved are termed *Knowledge Translation* (KT), and are still the topic of extensive discussion in domains where evidence-based practices are much more established than they are in software engineering. However, in Chapter 14, we do examine how KT can be applied in a software engineering context.

The point to emphasise though, is that all of these activities involve *procedures* that need to be applied and interpreted by human beings, with many of them also needing knowledge about the topic of the review. While tools can help with managing the process, the individual decisions still need to be made by a human analyst. In particular, because there will almost certainly be a wide variation of potential inputs to a review, it is possible that some of these will be interpreted differently by different people. To minimise the effects of this, systematic reviews are often conducted by two (or even more) people, who compare results at each stage, and then seek to resolve any differences (again in a systematic manner).

As indicated, because systematic reviews have different forms, the process of synthesis can also take many forms. (A very good categorisation of the wide range of forms of synthesis used across those disciplines that employ systematic reviews is provided in the book by Booth et al. (2012).) At its most simple, synthesis can consist mainly of classification of the material found, identifying where there are groups of studies addressing a particular issue, or equally, where there is a lack of studies. We term this a *mapping study*, and software engineering research has made quite extensive use of this form. A value of a mapping study lies partly in identifying where there is scope to perform a fuller review (the groups of related studies), and also where there is a need for more primary studies (the gaps). At the other extreme, where the material consists mostly of RCTs, or good quality experiments, synthesis may be organised in the form of a statistical meta-analysis. Meta-analyses do exist in the software engineering literature, but only in small numbers. Most software engineering studies use less rigorous forms (and sometimes forms that are less rigorous than could actually be used), and again, we will examine this in much more detail in Chapters 9, 10 and 11.

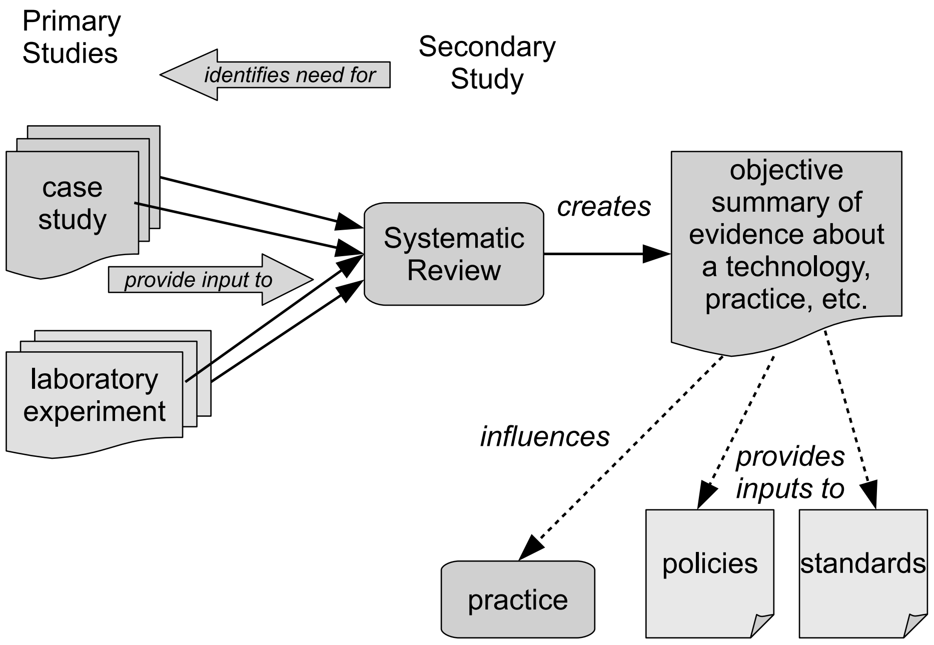

FIGURE 1.5: The context for a systematic review.

Figure 1.5 illustrates the wider context for a systematic review. So far we have mainly described the things that affect a review, but as we can see, the review itself also has some quite important roles. One of these is in providing a context for primary studies. Until the adoption of the evidence-based paradigm, these were mostly viewed as essentially being isolated studies that formed ‘islands’ of knowledge. When primary studies are viewed in terms of their role as inputs to a systematic review, there are two new factors that may influence the way that they are organised. One is the choice of topic—perhaps because a review has identified the need for further studies. The other is the way that primary studies report their results—one of the frequent complaints from analysts who conduct a systematic review is that important information is apt to be omitted from papers and reports. So designing and reporting of primary studies now needs to be more influenced by this role as an input to a secondary study than was the case in the past. Reviews also influence policies, standards and decisions about practice—and while this is still less likely to be the case in software engineering than in disciplines such as education and clinical medicine, consideration of these aspects should increasingly be a goal when performing systematic reviews.

The systematic review is the main instrument used for evidence-based studies and so will be discussed in depth through most of this book, and certainly in the rest of Part I. So, to conclude this introductory chapter, we need to consider some of its limitations too. This is because an appreciation of these is really needed when designing and conducting reviews as well as when seeking to understand what the outcomes of a review might mean to us.

1.4 Some limitations of an evidence-based view of the world

Not surprisingly, there has been a growing tendency for researchers, at least, to consider that knowledge that has been derived from an evidence-based process must inevitably be better than 'expert' knowledge that has been derived, albeit less systematically, from experience. And as the preceding sections indicate, we would to some degree support such a view, although replacing "inevitably" with the caveat "depending upon circumstances".

In clinical medicine and in wider healthcare, it has been argued that evidence-based research practices have become the "new orthodoxy", and that there are dangers in blind acceptance of the outcomes from this. Some of the arguments for this position are set out in a paper by Hammersley (2005). In particular, he questions whether professional practice can be wholly based on research evidence, as opposed to informed by it, noting that research findings do themselves rely upon judgement and interpretation. While many of the arguments focus upon how to interpret outcomes for practice, rather than upon the research method itself, the appropriateness of this form of research for specific topics does need to be considered. Even for systematic reviews, the two well known adages of "to a person with a hammer everything looks like a nail" and "garbage in–garbage out" may sometimes be apt.

So here we suggest some factors that need to be kept in mind when reading the following chapters. They are in every sense 'limitations', in that they do not necessarily invalidate specific evidence-based studies, but they might well limit the extent to which we can place full confidence in the outcomes of a systematic review.

A systematic review is conducted by people. There is inevitably an element of *interpretation* in the main activities of a systematic review: performing searches; deciding about inclusion and exclusion; and making various decisions during synthesis. All of these contain some potential for introducing *bias* into the outcomes. The practice of using more than one analyst can help with constraining the degree of variability that might arise when performing these tasks, but even then, two analysts who have the same sort of background might arrive at a set of joint decisions about which primary studies to include that would be different from those that would be made by two analysts who come from different backgrounds. Both the selection of studies, and also the decisions made in synthesis, can affect the outcomes of a review.

The outcomes depend upon the primary studies. The *quality* of the primary studies that underpin a systematic review can vary quite considerably. A review based upon a few relatively weak primary studies is hardly likely to be definitive.

Not all topics lend themselves well to empirical studies. To be more specific, the type of empirical study that is appropriate to some topics may well offer poorer scope for using strong forms of synthesis than occur (say) when using randomised controlled experiments. We will examine this more fully in Part II.

All of these are factors that we also need to consider when planning to perform a systematic review. And in the same way that a report of a primary study will usually make an assessment of the limitations upon its conclusions imposed by the relevant “threats to validity” (we discuss this concept further later), so a report of the outcomes from a systematic review needs to do the same. Such an assessment can then help the reader to determine how fully they can depend upon the outcomes and also how limited or otherwise the scope of these is likely to be.

In the next chapter we go on to look at the way that systematic reviews are performed in software engineering, and so we also look at some of these issues in rather more detail and within a computing context.

Chapter 2

Evidence-Based Software Engineering (EBSE)

2.1 Empirical knowledge before EBSE 17
2.2 From opinion to evidence 19
2.3 Organising evidence-based software engineering practices 23
2.4 Software engineering characteristics 25
2.5 Limitations of evidence-based practices in software engineering 27
2.5.1 Constraints from software engineering 27
2.5.2 Threats to validity 28

Although this chapter is mainly about how evidence-based ideas can be used in software engineering, we actually begin by examining some prior activities that helped pave the way for an acceptance of evidence-based thinking. To do so, we first examine some of the 'challenges' that empirical software engineer researchers were already posing, as well as some of the other factors that helped to make it the right time to introduce evidence-based thinking to software engineering in the years after 2004. We then describe a few examples of how evidence-based research has contradicted some widely-held beliefs about software engineering practices, after which we discuss what the concept of EBSE implies for software engineering and what the use of a systematic review might expect to achieve within a software engineering context. Finally, we examine some limitations that apply to evidence-based practices as used in software engineering research.

2.1 Empirical knowledge before EBSE

From around the mid-1990s there was a perceptible growth in the use of empirical studies to assess software engineering practices. In particular, some of these studies looked more widely at what was happening in software engineering research, and so we first look briefly at three such studies that have been quite widely cited, and at what they found.

- Zelkowitz & Wallace (1998) developed a classification of empirical validation forms, and to test this, they used it to categorise 612 papers published in the three years: 1985, 1990 and 1995. These were taken from a major conference, ICSE (International Conference on Software Engineering); an archival journal, (IEEE Transactions on Software Engineering); and a 'current practices' magazine (IEEE Software). After removing 50 papers because they addressed topics for which a validation was not appropriate, they then classified the remaining 562. They observed that about a third of the papers had no validation at all (although the percentage of these dropped from 36% in 1985 to 19% in 1995), and that a third relied upon informal 'assertions' (in effect, "we tried it out on a sample and it worked"). They also noted that "experimentation terminology is sloppy".

- At around the same time Walter Tichy raised the question "should computer scientists experiment more?" (Tichy 1998), and addressed many of the fallacies that were apt to be raised in opposition whenever the use of empirical studies was advocated. He particularly argued that computing in general was sufficiently well established to justify wider use of empirical validation than was being observed, and also observed that "experimentation can build a reliable base of knowledge, and thus reduce uncertainty about which theories, methods and tools are adequate".

- Somewhat later, Glass, Vessey and Ramesh conducted a series of classification studies of the ways that research was being conducted in the three major branches of computing: computer science, information systems and software engineering. These were based on papers published in a range of journals over the period 1995–1999. Their consolidated overview was published as (Glass, Ramesh & Vessey 2004), and showed that each branch had quite distinct characteristics. Once again though, based upon a sample of 369 papers, software engineering research methods were predominantly non-empirical, with 44% of the papers being classified as being "(non-mathematical) concept analysis" and 17% being "concept implementation" (loosely interpreted as "we built it and it worked").

So, when the idea of employing the evidence-based paradigm in software engineering research was proposed in 2004 by Kitchenham, Dybå & Jørgensen, this created considerable interest among researchers. We can suggest several reasons why this was well-timed.

- Firstly, there was the influence of the concerns raised in the studies of practice described above. These played an important role in widening awareness of the poor evidential basis available for software engineering techniques and practices.

- Secondly, empirical software engineering had also been making an increasing impact upon the academic software engineering community over the previous decade or so. How can we tell this was so? Two good indicators are:

- The establishment of a specialist journal (*Empirical Software Engineering*) in 1996, together with the publication of increasing numbers of empirical papers in many other journals.
- The establishment of two successful conference series. The first of these was the IEEE-sponsored ISESE (*International Symposium on Empirical Software Engineering*)—which began in 2002, and in 2006 merged with the *Metrics* conference to form the ESEM (*Empirical Software Engineering & Measurement*) series. The second was the smaller and more informal EASE (*Evaluation & Assessment in Software Engineering*) series of conferences, which began in 1996.

Taken together, these helped to promote an interest in, and better understanding of, empirical studies among researchers, as well as providing a useful corpus of material for secondary studies.

- A further factor in favour of the acceptance of the concepts of EBSE has been the growing recognition that the results from individual empirical studies are often inconclusive, and that such studies are difficult to replicate successfully (Sjøberg, Hannay, Hansen, Kampenes, Karahasanović, Liborg & Rekdal 2005, Juristo & Vegas 2011). Since software engineering researchers are partly motivated by the goal of providing input to both software engineering practitioners and also policy-makers, an approach that offers the potential for creating more convincing demonstrations to these audiences is likely to be favourably received.

The rest of this chapter examines some of the ways in which evidence-based thinking has begun to influence software engineering research. We begin by looking at a number of examples of where evidence-based studies have contradicted 'expert' opinion and established practice to explain some of the challenges this approach has created. We then look at how EBSE is organised; consider some aspects of software and software engineering practices that influence its effectiveness; and finally look at some examples of how evidence-based studies can provide guidelines for using some specific software engineering practices.

2.2 From opinion to evidence

Expert opinion and experience are often linked in software engineering. Techniques that have proved effective in one context are apt to be extrapolated to others, without this necessarily being appropriate. Expert opinion can also easily become linked to what might loosely be termed 'academic dogma', such as the belief that something that uses mathematically based formalisms or algorithms will be 'better' in some way. For some situations, it is certainly true that mathematical forms of reasoning are appropriate of course (the design of compilers is a good example). However, given that software engineering can be characterised as a 'design discipline', the associated non-deterministic nature of many software engineering activities (and the corresponding absence of 'right' or 'wrong' solutions) means that we need to be careful of overly emphasising any assumptions about rigour that the use of mathematical formalisms can confer. Indeed, and in contrast, one of the strengths of evidence-based studies is the rigour with which they can be conducted, although this is not conventionally 'mathematical' in its form. Their use of systematic and well-defined procedures provides an appropriate means for both linking experience to knowledge and also addressing the non-deterministic nature of software engineering activities.

One consequence of the formulation of ideas about EBSE has been the proliferation of published secondary studies over the following decade. A series of three broad 'tertiary' studies (a *tertiary* study is a secondary study that performs a mapping study of other secondary studies) identified over 100 published systematic reviews in the period up to 2009 (Kitchenham, Brereton, Budgen, Turner, Bailey & Linkman 2009, Kitchenham, Pretorius, Budgen, Brereton, Turner, Niazi & Linkman 2010, da Silva, Santos, Soares, França, Monteiro & Maciel 2011). Keeping up with this proliferation of secondary studies and indexing them has proved to be quite a challenge[1], but we can estimate that there have been over 200 secondary studies published in the first decade of EBSE. Inevitably, some of these have contradicted expert opinion (or "common wisdom" if you prefer) based on experience and expertise. Here we briefly examine three examples that highlight particular aspects of the clashes that can occur between evidence and opinion.

Estimating software development effort. Project planning for software projects, like all planning, is a challenging exercise. Over the years, algorithmic cost modelling approaches, such as that employed by the well-known COCOMO model (Boehm 1981) has often been viewed as the 'right' approach to predicting project costs. In part, this may well be because it is much more tractable to teach about using models than about using experience when teaching students about software engineering, and so greater emphasis has been placed upon the former. Anyway, whatever the reason, this belief is clearly challenged by the findings of Jørgensen (2004), who, from a set of 15 primary studies comparing models with expert judgement, found that:

- For one third of them, a formal cost model worked best;

[1]We do maintain a database on our website at www.ebse.org.uk, but this is inevitably always well behind the 'current' position.

- In another third, expert cost estimation was most effective;
- The remaining third identified no difference between expert judgement and model-based forms.

From this, and from examining similar studies in other disciplines, Jørgensen observed that "there is no substantial evidence supporting the superiority of model estimates over expert estimates". He noted that there were "situations where expert estimates are more likely to be more accurate, e.g. situations where experts have important domain knowledge not included in the models". And conversely, that "there are situations where the use of models may reduce large situational or human biases, e.g. when the estimators have a strong personal interest in the outcome".

So, here we see an example of how an evidence-based approach can be used to resolve the different outcomes from a range of studies with outcomes that may appear to be contradictory, and can synthesise the results in order to provide useful guidelines on how to use such techniques.

Our next example again highlights the point that the benefits claimed for well-known software engineering techniques are not always found to occur upon closer inspection.

Pair-Programming. The emergence of agile methods for software development, and of *extreme programming* in particular, has popularised the use of *pair programming*, with this often being used as a technique outside of an agile context. In pair programming, two programmers work together with a single keyboard, mouse and screen, taking it in turns to be the 'driver' and the 'observer' or 'navigator'. The perceived benefits include roles such as training of novices, producing better quality code, and speeding up the development process.

Pair programming does lend itself to experimentation. However, the range of experiments that have been performed is quite wide, and making any form of comparison with 'solo programming' is something of a challenge (it is easier to specify what pair programming involves, but not quite so easy to do so for solo programming). The meta-analysis of the outcomes from 18 primary studies reported in Hannay, Dybå, Arisholm & Sjøberg (2009) demonstrates this very clearly—although with some caveats about the possible existence of reporting bias[2]. After looking at the effects of pair programming upon measures of quality, duration and effort, the authors advise that:

> "If you do not know the seniority or skill levels of your programmers, but do have a feeling for task complexity, then

[2] *Reporting bias* occurs when we find the outcomes of studies with inconclusive or negative results do not get published, either because the authors do not think them of interest, or referees reject the submitted papers because they do not show significant results.

> employ pair programming either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important."

So, this example also shows that while there may be benefits to using a particular technique, they are unlikely to be universal, nor will they necessarily be consistent with every claim made for it.

Finally, we look at an example that shows that the benefits claimed for a technique on the basis of early studies may not be supported when a fuller set of studies is taken into account.

Inspections. The practice of performing inspections has long been accepted as being a useful technique for validating software and related documents. So not surprisingly, efforts have been made to optimise the benefits, usually by structuring the reading technique, with one of these being *perspective-based reading* or PBR (Basili, Green, Laitenberger, Lanubile, Shull, Sorumgard & Zelkowitz 1996). Early studies of its use suggested that, when compared to other forms, it was possible to achieve a 35% improvement when using this approach to inspection.

However, the systematic review performed by Ciolkowski (2009) found that, when used with requirements documents, there was no significant difference between PBR and any of the ad-hoc code inspection techniques in terms of their effectiveness. Further, PBR was less effective as a way of structuring inspections than the use of checklists.

One concern was that many of the studies were effectively replications of the original study that used the same dataset as the original study. However, one of the independent studies did also find positive results for PBR, and there are other factors that might explain some of the variation in results.

In many ways this third example shows that initial claims for the benefits of new software engineering techniques need to be treated carefully, and that the developers of a technique may not be the most appropriate people to conduct such studies, however carefully they try to avoid being biased.

We should add an important caveat here, that systematic reviews (in any discipline) provide evidence that represents the best knowledge available at a given point in time. If and when more (and hopefully better quality) primary studies become available, a later extended systematic review may well be able to refine and revise the original findings. So we should always view the outcomes of any systematic review as representing the "best knowledge" that is currently available, and indeed, one of the tasks in reporting a review is to assess the quality of the primary studies used, and also the effect this may have upon any conclusions (see Chapter 7).

Two key aims of evidence-based studies are to avoid bias and encourage objectivity, and in the next section we examine how the procedures of a systematic review can be organised for use with software engineering topics.

2.3 Organising evidence-based software engineering practices

The previous section looked at what an evidence-based approach can tell us about software engineering practices. In this section we discuss how this is done, and why this should be able to give us confidence in its objectivity.

In proposing the adaptation of evidence-based practices for use in software engineering, Kitchenham, Dybå & Jørgensen (2004) suggested that this could be structured as a five-step process.

1. Convert the need for information into an answerable question.
2. Find the best evidence with which to answer the question.
3. Critically appraise the evidence for its validity (how close it comes to the truth), its impact (the 'size' of the effects observed), and its applicability (how useful it is likely to be).
4. Integrate the critical appraisal with software engineering expertise and stakeholders' values.
5. Evaluate the effectiveness and efficiency in the previous steps 1–4, and seek ways to improve them.

The first three steps are essentially the role of the *systematic review*, while the fourth is that of *Knowledge Translation* (we explained what was meant by KT in the previous chapter, and discuss KT later in Chapter 14). The fifth is one of ensuring that the research procedures are themselves subject to constant scrutiny. (An illustration of this is the use of a systematic process to produce the revised *Guidelines* that form Part III of this book.)

Our concern here is with the first three steps. These tasks can be structured as a set of nine *activities*, grouped as three *phases* (note that the phases do not map directly on to the steps, although the number is the same). This is illustrated in Figure 2.1—although we should note that this shows a somewhat idealised model, and that in practice, there is likely to be some iteration between the different activities. We will not discuss each activity in detail here, since all of them are described more fully in the following chapters. So the main task here is to identify what each one involves. Practical guidance on performing these activities is also provided in Part III.

Phase 1: Plan the review. The first phase addresses the task of designing how the study is to be performed, with this being documented through the *review protocol.* Planning a review involves three important activities.

1. *Specify the research question.*

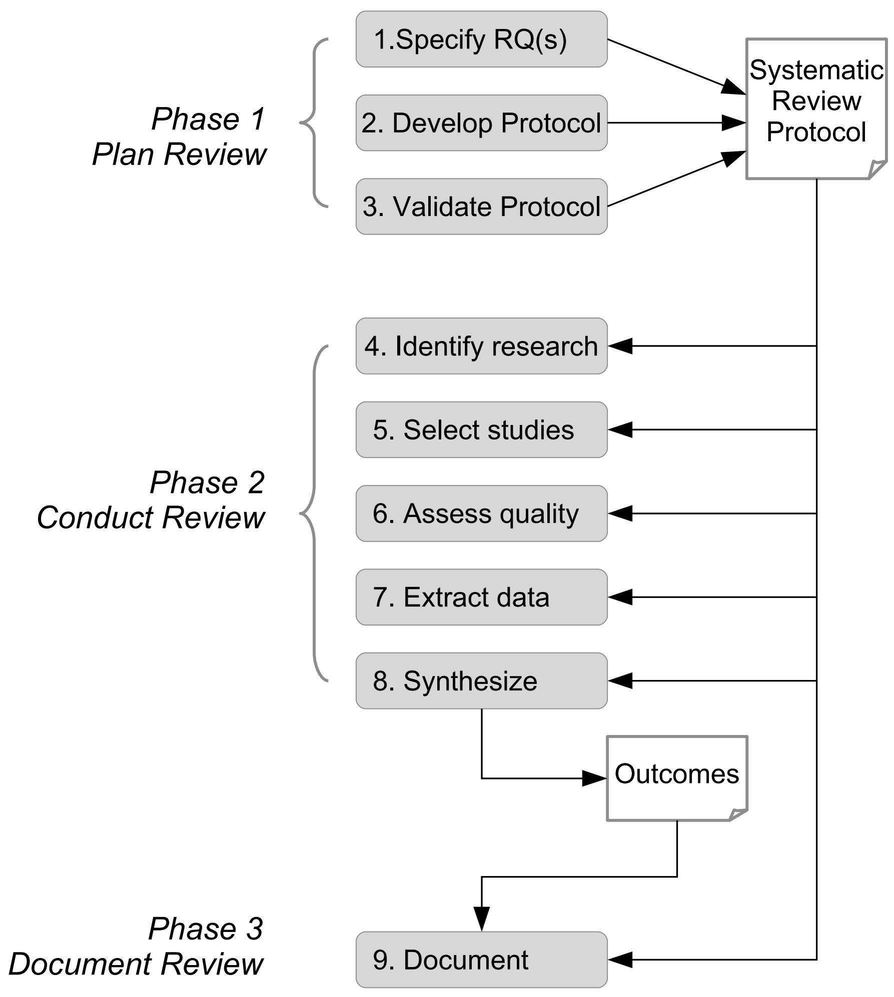

FIGURE 2.1: Overview of the systematic review process.

2. *Develop the review protocol.*
3. *Validate the review protocol.*

We discuss the activities of this phase in more detail in Chapter 4.

Phase 2: Conduct the review. In this phase we put the plan into action. Phase 2 is very much driven by the research protocol, and any *divergences* that occur, requiring that we change the plan to reflect unexpected or other circumstances, need to be carefully documented.

4. *Identify relevant research.* We discuss this activity in more depth in Chapter 5.
5. *Select primary studies.* A fuller explanation of this activity is provided in Chapter 6.

While all of the different forms of systematic review that we discuss in the next chapter should involve performing the first two activities of this phase, not all will necessarily undertake the remaining three in full detail.

6. *Assess study quality.* We discuss these issues further in Chapter 7.
7. *Extract required data.* This is discussed in more detail in Chapter 8.
8. *Synthesize the data.* This is a challenging task that we examine in much greater detail in Chapters 9, 10 and 11.

Phase 3: Document the review. Reporting about the processes and outcomes of a review is discussed in Chapter 12.

We should note here that applying evidence-based ideas in software engineering is not necessarily confined to conducting systematic reviews, although this is the form that has largely been taken so far, and that we focus upon in this book. In the *Further Reading* section provided at the end of Part I, we discuss the study reported by Kasoju, Peterson & Mäntylä (2013) which used EBSE practices as the means of investigating a particular industry problem (related to the software testing processes used in the automotive industry). The important distinction here is that whereas a systematic review forms a *topic*-specific application of evidence-based ideas, the approach used in that study was *problem*-specific, and employed a multi-stage process involving a mix of empirical forms (including a systematic review).

2.4 Software engineering characteristics

At this point, it is useful to consider how the review process outlined in the previous section is influenced by some characteristics of the software engineering discipline—and sometimes by the characteristics of its practitioners too.

The challenges posed by the characteristics of software were outlined by Fred Brooks Jr. in one of software engineering's seminal papers (Brooks Jr. 1987). These are obviously important to any researcher conducting a primary study, and so clearly do have influence upon a secondary study in terms of the likely spread of results they create. Here we look at some factors that are in part consequences of the main characteristics identified by Brooks (invisibility, changeability, mix of static and dynamic properties) and in part consequences of the way that the discipline has evolved.

- *Primary studies involve active participation.* In software engineering it is common to refer to the people who take part in primary studies as *participants* rather than *subjects.* This is because they perform active tasks (coding, reviewing, classifying, etc.) rather than simply receiving some form of treatment (as occurs in much of clinical medicine). Not only does this make it impractical to conduct Randomized Controlled Trials (RCTs) in software engineering, since 'blinding' of participants and experimenters is virtually impossible, it also means that the outcomes of primary studies may well be quite strongly influenced by the characteristics of the particular set of participants involved, by the skills that they have, and by their previous experiences. Like some of the other characteristics we consider here, this one complicates the task of *synthesis.* We examine some aspects of this further in Part II when we look at how primary studies are organized.

- *Software engineering lacks strong taxonomies.* The terms that we use are often imprecise, and software engineers are rather prone to create new terms to describe ideas that may well be closely related to existing ones. This can complicate *searching* since we need to consider all possible forms of terminology that might have been used in the titles and abstracts of papers. Snowballing may help with this, but essentially it stems from the constraint of studying procedures and artifacts.

- *Primary studies lack statistical power.* Because software engineering studies usually need specialist skills and knowledge, it is often difficult for experimenters to recruit enough participants to provide what is generally regarded as an acceptable level of statistical power (Dybå et al. 2006). This in turn reduces the strength of the *synthesis* that can be achieved in a systematic review.

- *There are too few replicated studies.* There may be many reasons for this, not least the problem of getting a paper describing a replicated study published, particularly one that is considered to be a *close* replication (Lindsay & Ehrenberg 1993). Although this view may be inaccurate, if researchers think it is so, then they will be reluctant to conduct replicated studies. There is also debate about what exactly constitutes a 'satisfactory' replication study (we will examine this issue in Chapter 21). Again, this presents a problem for *synthesis* in particular.

- *Reporting standards are often poor.* Many primary studies are reported in a manner that effectively ignores the likelihood that, at some time in the future, a systematic reviewer will attempt to extract data from the paper. While this might have been more excusable in the past, that really is not the case now. Another form of reporting problem is related to our culture of refereed conferences, which can lead to researchers publishing more than one paper that uses the same set of results—requiring the systematic reviewer to take care not to count such studies more than once. Similarly, some papers also describe more than one experiment, complicating separation of individual studies when the analyst conducting a systematic review is performing *data extraction*. We address reporting needs for primary studies in Part II.

2.5 Limitations of evidence-based practices in software engineering

We touched on some factors that constrained the use of the evidence-based paradigm in Chapter 1. In this section we discuss these in the context of EBSE, and look at how they may be affected by the characteristics of software and software engineering. We also introduce the concept of *threats to validity* in rather more detail.

2.5.1 Constraints from software engineering

We begin by considering how these factors are influenced by the nature of our discipline, as characterised in the preceding section.

A systematic review is conducted by people. As we identified earlier, a major risk arising from this aspect is that of *bias*. This can arise in various stages, for example when searching using electronic forms, our choice of search engines and of search terms may favour our finding some studies and perhaps missing others. (As we mentioned in the preceding section, software engineering does lack strong taxonomies.) Equally, when searching manually, our choice of journals and conferences may influence the outcomes. Similarly, our inclusion/exclusion criteria might lead to bias — for example, in software engineering, replicated studies are probably less likely to be published than original ones, but may be available as technical reports. Equally, the analyst needs to be aware that the common practice of expanding conference papers into journal papers can easily lead to a study being counted twice. The issue of bias is discussed quite extensively by Booth et al. (2012) in their discussion of analysis, where they also discuss some strategies that might be used to cope with this.

The outcomes depend upon the primary studies. Even when it is systematic, the main contribution of any review will arise from its *synthesis* of the outcomes from the primary studies. For software engineering these primary studies typically:

- exhibit poor statistical power arising from having small numbers of participants;
- address a wide variety of research questions;

- employ a range of empirical forms;
- have a tendency to employ student participants for tasks that might actually be performed rather differently by more experienced practitioners.

These are all factors that impede the production of reliable outcomes from a secondary study, or at least, constrain the scope of any outcomes.

Not all topics lend themselves well to empirical studies. In software engineering research we are concerned with the study of *artefacts*, which we create, rather than of 'physical' entities. Glass et al. (2004) identified a wide range of forms for evaluation that were used in software engineering, and while we might feel that our discipline could make more use of empirical evaluation, we need to also recognise that forms such as 'concept implementation' are valid approaches to research, and may sometimes be more appropriate than empirical studies.

Not only do we create and study *artefacts*, these are also often being subject to continuous change and evolution. This turn means that different studies that make use of a given artefact in some way may actually all be based upon different versions. This can also apply to our conceptual tools—for example, the UML (Unified Modeling Language) has gone through a number of versions, adding new diagrammatical forms as it evolves. So different studies based on using the UML may not always be directly comparable.

2.5.2 Threats to validity

The concept of limitations upon the rigour of an empirical study, expressed as *threats to validity*, is well established for primary studies, and we discuss them in that context in Part II. However, the concept does apply to secondary studies too, and indeed, they are often discussed when reporting a systematic review. The factors that influence the validity of a study are largely those discussed above, but cast into a slightly different perspective within the structure of a systematic review. Shadish et al. (2002) identify four major forms of threat arising for primary studies, and here we briefly discuss how each of these might be interpreted in the context of a secondary study.

Construct Validity is concerned with how well the *design* of the study is able to address the research question. Essentially this relates to the consistency and comparability of the operationalisation of the outcome measures as used in the primary studies.

Internal Validity is concerned with the *conduct* of the study, particularly related to data extraction and synthesis, and whether there are factors that might have caused some degree of bias in the overall process.

Conclusion Validity is concerned with how reliably we can draw conclusions about the link between a treatment and the outcomes of an empirical study (particularly experiments). For a secondary study, we can therefore relate this to the *synthesis* element of a systematic review, and how well this supports the conclusions of the review. Hence for secondary studies there is little distinction between internal and conclusion validity.

External Validity is concerned with how widely a cause-effect relationship holds, given variations in conditions. For a secondary study this should be based upon an assessment of the range covered by the primary studies in terms of their settings, materials and participants.

Taken together, these provide a framework that can be employed to assess the possible limitations that apply to the outcomes of a review. They need to be reported by the systematic review team, mainly because they are the people who are in the best position to assess whether these factors are likely to have had any effect upon the outcomes, and if so, how significant this might be. In turn, this knowledge may be important to anyone wanting to make use of the outcomes in making decisions about practice or policy, as well as to anyone who might in the future wish to extend and update the review.

But do note that...

None of these issues are likely to make it impossible to conduct a useful systematic review, although they may well limit its scope and usefulness, since we are often studying differences in practice that have fairly small effects upon the outcomes. However, where possible, it is important to anticipate the influence of the likely 'threats' when writing the research protocol.

This page intentionally left blank

Chapter 3

Using Systematic Reviews in Software Engineering

3.1 Systematic reviews 32
3.2 Mapping studies 34
3.3 Meta-analysis 37

We conduct secondary studies in order to answer a variety of research questions, so it is not surprising that we need to adapt the way that such a study is organised according to the question being addressed. Some reviews seek to answer questions about software engineering practices (for example, “in what situations is pair programming likely to be a good strategy to adopt?”); others may examine research trends (such as “what have been the ‘hot topics’ in cloud technology research and how have they changed with time?”); while a ‘broad’ form of review might be used to help determine whether a more focused review of a topic is feasible—or whether it is first necessary to perform more primary studies on that topic.

The way that we conduct a secondary study, and in particular, the way that searching is organised and the choice of procedures used to synthesize the results, will therefore vary substantially. Indeed, the very concept of ‘synthesis’ is apt to have many interpretations, especially when conducting studies of research trends, where the reviewers may well choose to include non-empirical forms of input.

Booth et al. (2012) catalogue a range of forms that are used for organising secondary studies across a range of disciplines. Since not all of these are very relevant to software engineering, in this chapter we briefly examine those forms that software engineers do use, and what they use them for. (Each of these forms will also be discussed in much greater depth in the following chapters.) The key forms are as follows.

- *Systematic reviews* (using both qualitative and quantitative inputs). In some cases, it is also possible to perform a meta-analysis for a quantitative review.
- *Mapping studies* (used both for secondary and }tertiary studies).

Note that our use of this terminology does differ a little from that used in Booth et al., largely reflecting the way that the use of secondary studies has evolved in software engineering.

Figure 3.1 shows a simple summary of the roles of, and relationships between, the different forms. In the rest of this chapter we say a little more about each of these, and in particular, about the way that each form is used in software engineering.

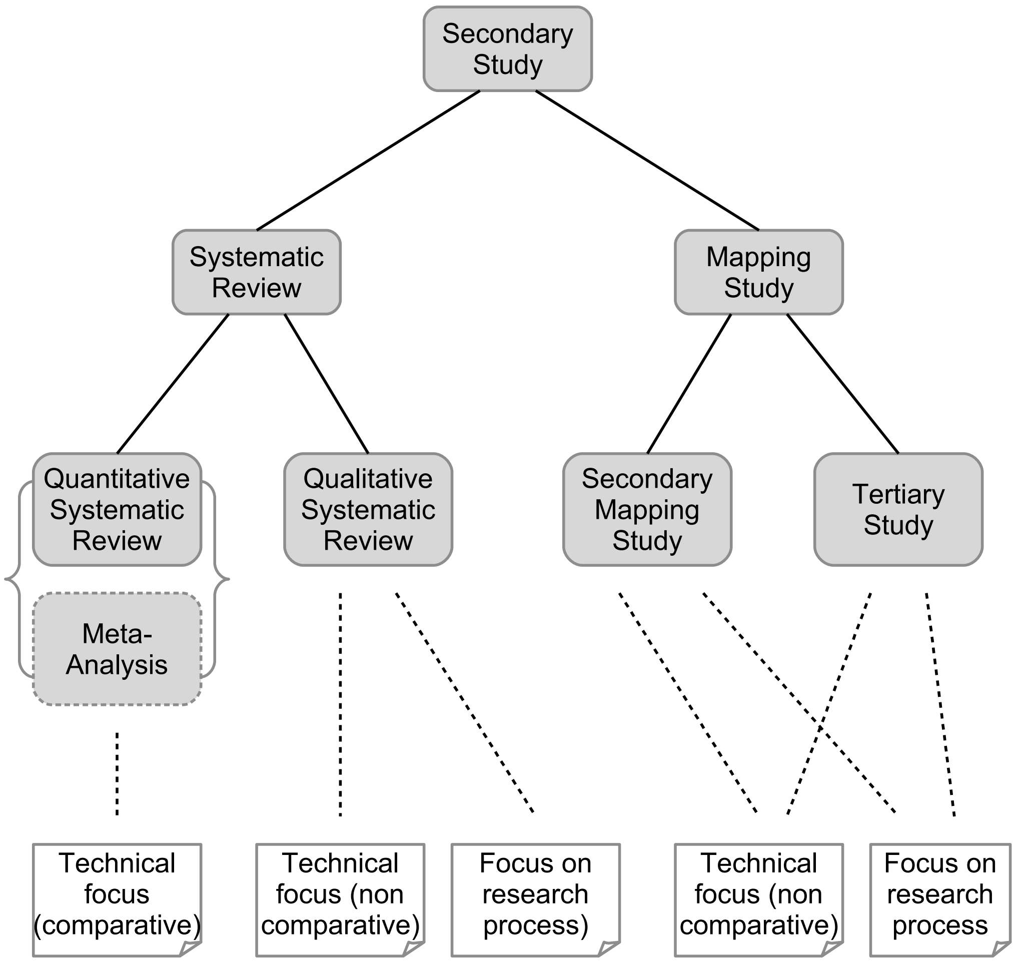
Secondary Study
Systematic Review
Mapping Study
Quantitative Systematic Review
Meta-Analysis
Qualitative Systematic Review
Secondary Mapping Study
Tertiary Study
Technical focus (comparative)
Technical focus (non comparative)
Focus on research process)
Technical focus (non comparative)
Focus on research process

FIGURE 3.1: The hierarchy of study forms.

3.1 Systematic reviews

While 'systematic review' is often used as a generic term for all types of review conducted using evidence-based practices, a systematic review is also a well-defined form of study used to answer a specific research question. Systematic reviews can be further sub-classified according to whether they involve synthesizing qualitative or quantitative forms of data. In turn, this will determine how both data extraction and synthesis need to be organised. In particular, for a quantitative systematic review it may well make it possible to perform a statistical meta-analysis as the means of synthesis, providing greater confidence in both the statistical significance and also the statistical power of the outcomes. We discuss this a bit more in Section 3.3.

Since the organisation and use of systematic reviews is covered extensively in the rest of the book, this section will be confined to discussing how systematic reviews are commonly used in software engineering.

A matter of classification

We should observe here that throughout Part I of this book, our classification of different publications as being mapping studies or systematic reviews may not always agree with those used by the original authors. This is because we use the way that synthesis is performed in a study as our key criteria for differentiating between these forms, and so consider some studies that have been described as "systematic reviews" when published, to be more correctly classified as mapping studies for that reason.

One role of a systematic review in software engineering is to establish whether particular techniques or practices work better than others, and if so, under what conditions this will be true. Existing systematic reviews span many activities and forms (agile methods, project estimation, design patterns, requirements elicitation techniques, regression testing techniques, just to name a few). They are also used for other purposes, such as to evaluate how far particular techniques have been adopted by industry and commerce, or to identify the benefits of using tools in a particular context.

The purpose of a review will determine the type of input that is expected, and hence the way that the inputs from different studies can be synthesized. For example, a systematic review addressing topics such as pair programming, inspection techniques or the use of software design patterns might be expected to involve synthesizing the results from experiments and quasi-experiments. Studies looking at the adoption of tools in industry, or the take-up of agile methods, are more likely to be synthesizing the outcomes from observational studies and case studies. In this book we are mainly concerned with the following two classes of systematic review, as identified in Figure 3.1.

Quantitative reviews Inputs for these are likely to come from experiments or quasi-experiments, or from data mining using existing repositories, and the studies themselves may well be performing comparisons or producing estimates based on past profiles. Associated research questions are likely to address comparative aspects such as "does technique X perform better than technique Y?". Synthesis may take a range of forms ranging from tabulation of the different outcomes through to a statistical meta-analysis, depending on how much the primary studies vary in terms of topics and measures used. A good example of a quantitative review is that of Dieste & Juristo (2011), comparing the effectiveness of different requirements elicitation techniques.

Qualitative reviews These usually address questions about the specific use of a technology, and so are unlikely to involve making comparisons (and hence less likely to address questions that involve any sense of something being ‘better’). In a software engineering context they may well be used for such tasks as studying adoption issues, or perhaps more likely, the barriers to adoption, employing procedures for synthesis that can help to identify patterns in the data. This class of review also includes studies that look at research methodologies, not just practice, such as Kitchenham & Brereton (2013).

Systematic reviews, along with mapping studies are the two forms that have been most widely employed in software engineering to date. However, a few examples of the use of meta-analyses for software engineering topics do exist, and so we also discuss the role of meta-analysis in the final section.

3.2 Mapping studies

The goal of a mapping study is to survey the available knowledge about a topic. It is then possible to synthesise this by categorisation in order to identify where there are ‘clusters’ of studies that could perhaps form the basis of a fuller review, and also where there are ‘gaps’ indicating the need for more primary studies. Mapping studies may also be ‘tertiary’ studies, for which the inputs are secondary studies, so providing a higher level of categorisation of knowledge.

Mapping studies have found wide acceptance in software engineering, although this form of study appears to be less widely used in other disciplines. This may reflect the nature of software engineering and its vocabulary in particular. Software engineering is still not a truly ‘empirical’ discipline, even if it is slowly moving that way, and so we may often have both very limited knowledge about how widely a topic has been studied, and also relatively few studies that are empirical in form. Related to this, empirical studies may well be reported in many different venues, meaning that we need to search widely to find all relevant material. In addition, whereas the vocabulary of clinical medicine is based upon terms used for the parts of the human body and its conditions, which have been standardised over a long period of time, software engineering deals with artifacts, and so invents new terms to describe these. Unfortunately, software engineers are also apt to “reinvent the wheel” when doing this, sometimes using new terms to describe a concept that was developed earlier, but may not have been realised at the time because of (say) a lack of computational power.

If we return to the model of the systematic review process described in the previous chapter, we can describe a mapping study as a form that involves relatively little synthesis. A mapping study may also include an element of quality assessment, depending upon its purpose.

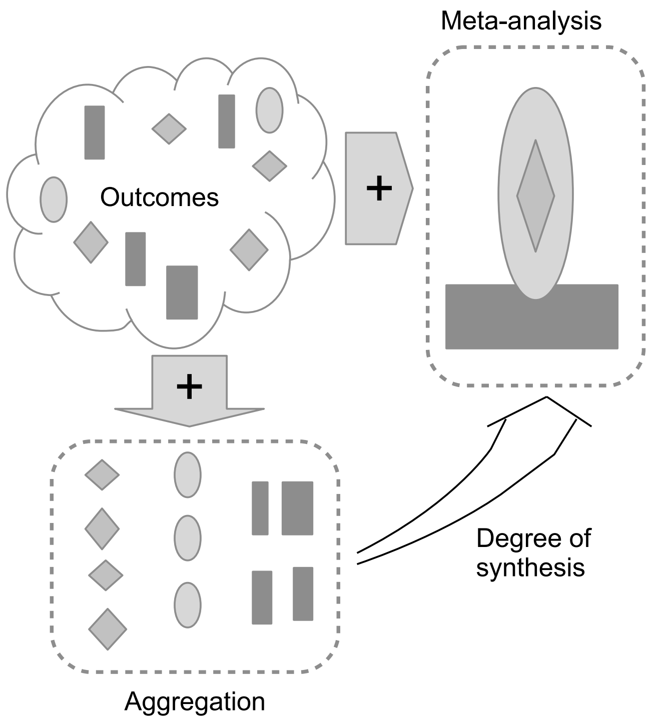

FIGURE 3.2: The spectrum of synthesis.

Figure 3.2 illustrates this issue. At one extreme, simple *aggregation* alone just groups together any data occurring within a given category, and while providing knowledge about the count for each category (such as number of papers published each year), it creates no new derived knowledge. Extending from that we have a spectrum of *synthesis*, extending through to a statistical meta-analysis, whereby new knowledge is derived about such aspects as patterns and associations in the data. The issues related to performing synthesis across the spectrum of study forms are discussed more fully in Section 10.2.

Searching may well use quite a broad set of terms in order to ensure that we do not overlook any particular group of studies. It also needs to be as thorough as possible in order to obtain a clear and useful picture of the topic.

The activity of categorisation may employ a number of different schemes, intended to reflect specific characteristics of the primary studies. It might use an existing scheme—for example, we might want to classify the studies found in terms of the research method employed for each primary study, using a set of categories such as experiment, quasi-experiment, survey, observational, and case study. Employing an existing categorisation scheme also provides a useful basis for identifying where there are 'gaps'. We might also derive the categories for a given characteristic by looking at the set of studies found, and grouping these to reflect the patterns observed.

So, in what context do we find it useful to conduct a mapping study? Below, we briefly examine two examples of situations where mapping studies may be particularly relevant.

Studying research trends. A mapping study may be useful as a means of analysing how research in a given topic has evolved over a period of time (so one of the categories used for the studies will need to be publication date). Such a study may focus upon identifying the "hot issues", or the techniques used, or even the countries where the research has been performed.

One example of its use in this role, mentioned earlier, is that of the *tertiary study*. To recap, a tertiary study is a form of systematic review for which the inputs are secondary studies. A *broad tertiary study* is organised as a mapping study where the purpose is to categorise these and to observe trends. The earliest tertiary study conducted in software engineering was that reported in (Kitchenham, Brereton, Budgen, Turner, Bailey & Linkman 2009). This study identified a set of secondary studies and categorised them by type (such as research trends) and topic. Later broad tertiary studies such as (Kitchenham, Pretorius, Budgen, Brereton, Turner, Niazi & Linkman 2010) and (da Silva et al. 2011) also included a quality assessment of the secondary studies. Viewed as a series, the value of these studies was therefore to index the emerging field of evidence-based studies, identifying those areas of software engineering where most activity was taking place. This is also an example of where aggregation is an appropriate form of analysis.

PhD literature review. Preparation for PhD study almost always requires a candidate to undertake a 'literature review' of the state of the art related to the intended topic. Traditionally this is conducted using informal searching, with expert guidance provided by the supervisor. However, for PhD projects involving empirical studies in particular, the use of a mapping study may well provide a very useful initial stage for a study, as examined in (Kitchenham, Budgen & Brereton 2011). Using this approach is not just appropriate for empirical topics of course, definitions and research trends may usefully be studied in this way too, as demonstrated in the review of different definitions of 'service oriented architecture' or SOA, provided in (Anjum & Budgen 2012). Here, it is appropriate to employ a degree of synthesis in analysing the outcome data.

While these are by no means the only roles that can be performed by a mapping study, they are fairly representative of the ways that this form has been used in software engineering.

3.3 Meta-analysis

For clinical medicine, where a secondary study may well be drawing together the results from a number of Randomized Controlled Trials (RCTs), the use of a statistical meta-analysis (which we discuss more fully in Chapter 11) is often appropriate. This is because primary studies of a new clinical treatment are likely to use the same baseline(s) for their comparisons and ask similar research questions about the effective use of a treatment. For software engineering however, even when a review is drawing together the outcomes of a set of experiments, these may well vary widely in form, as well as in their research questions.

Where the use of meta-analysis is an option, this is usually because there is a reasonable number of primary studies with similar forms, and that these also use comparable response variables. This was the case for the meta-analysis of pair programming studies performed by Hannay et al. (2009), although as here, the analysts may still have to cope with wide variation in the characteristics of the primary studies. In addition, many primary studies in software engineering have poor statistical power, as we observed earlier. However, one benefit of being able to use meta-analysis is that any outcomes can then be assigned a quantitative measure of confidence based on the use of inferential statistics.

One question we might well ask is whether this position is likely to change in the future so that we will see more use of meta-analysis. One of the aims of this book is to provide guidance for the use of evidence-based practices in software engineering, and in doing so, to encourage the use of more rigorous forms of synthesis. As we explain in Chapter 11, meta-analysis is possible in a variety of circumstances and there seem to be no reasons why it should not be employed more widely in software engineering. Certainly, if EBSE is to make greater impact upon the software engineering profession, then an increased use of meta-analysis is likely to be an important way of helping with providing potential users with appropriate levels of confidence in the findings from our studies.