Chapter 3
Cost Prediction and Software Project Management

Martin Shepperd

Abstract This chapter reviews the background and extent of the software project cost prediction problem. Given the importance of the topic, there has been a great deal of research activity over the past 40 years, most of which has focused on developing formal cost prediction systems. The problem is that presently there is limited evidence to suggest formal methods outperform experts, therefore detailed consideration is given to the available empirical evidence concerning expert performance. This shows that software professionals tend to be biased (optimistic) and over-confident, and there are a number of deep cognitive biases which help us understand why this is so. Finally, the chapter describes how this might best be tackled through a range of simple, practical and evidence-based methods.

3.1 Introduction

Cost estimation[1] has been viewed as a challenging and important part of software project management for almost 60 years. Interestingly, Benington (1956) writes of his experiences developing, what was back in the mid-1950s, a large air defense system comprising half a million lines of code (LOC). In it he tabulates what he termed 'reasonable production costs' and although the headings such as computer

[1] There is something of a proliferation of terminology. Whilst the majority of writers refer to cost modelling or prediction, strictly speaking the usual focus is upon labour or effort which forms the dominant part of costs and is usually the hardest to predict. Such costs may or may not be reflected in the price charged to the client or user. This chapter will use the term in this particular sense. Likewise, estimation and prediction are used interchangeably since we're only concerned with future events.

M. Shepperd (✉)
Department of Computer Science, Brunel University, Middlesex UB8 3PH, UK
e-mail: martin.shepperd@brunel.ac.uk

G. Ruhe and C. Wohlin (eds.), *Software Project Management in a Changing World*,
DOI 10.1007/978-3-642-55035-5_3,

and paper costs might no longer be seen as relevant, others such as specification, coding and testing remain pertinent. The outcome was 'the schedule slipped by a year', something that remains distressingly familiar!

So how bad is the problem? Apart from the anecdotal, evidence is surprisingly elusive probably due to the commercially sensitive nature of poor project cost estimation. Jørgensen and Moløkken-Østvold (2006) reviewed multiple sources of evidence and concluded that a typical cost estimation error was 'in the range of about 30 %'. Another indicator that not all is well comes from the 2005 and 2007 surveys conducted by El Eman and Koru (2008) who from a total of 388 responses found 'the most critical performance problem in delivered software projects is therefore estimating the schedule and managing to that estimate'. An independent study by the European Services Strategy Unit of 105 large public ICT projects (Whitfield 2007) found more than half to show cost overruns with the average cost being 30.5 %, a figure very much in line with Jørgensen and Moløkken-Østvold (2006).

The question therefore arises, as to why are software project costs difficult to estimate? There are many reasons. First and foremost is complexity. Many projects are extremely large undertakings with multiple stakeholders in a setting characterised by uncertainty, inconsistency and change. Second, software development is best viewed as a design type activity and it is emphatically not concerned with production. This means the sub-tasks and activities are not routine so simple linear extrapolation is seldom a safe guide. Third, estimates are required at a very early stage when little is known and requirements are still to be discovered, arbitrated, let alone documented. Finally, there are many subtle, and not so subtle, social and political pressures upon those responsible for cost modelling. In his analysis of a wide range of projects, Flyvbjerg refers to this tendency to under-estimate costs and over-estimate benefits in order to secure funding for a proposed project as 'strategic misrepresentation' (Flyvbjerg 2008).

Clearly, these problems with predicting software project costs have significant ramifications. First, we see a tendency for errors in one direction, i.e., bias or a propensity for over-optimism. Second, poor cost prediction will severely hamper meaningful cost-benefit analysis and the consequent unnecessary cancellation of projects that should not have been commissioned in the first place. Conversely, under-estimation might lead to missed opportunities or sub-optimal procurement decisions.

3.2 A Review of State-of-the-Art Techniques

The first thing to consider is what is an estimate? Although it can easily be forgotten, it must be stressed that an estimate is a probabilistic statement (DeMarco 1982; Kitchenham and Linkman 1997), and consequently, to simply report an estimate as a point value masks important information. As an example, if a project manager makes a prediction that the Integration Testing will take 150 person-hours,

we do not know with what confidence he or she makes this statement; it could be with near certainty or it could be a wild guess with almost no certainty. Thus there are two components. Jørgensen and Sjøberg (2003) recommend a simple approach based on an interval and a confidence level. Based on the Integration Testing example, the project manager (if highly confident) might state 140–160 person-hours at 90 % confidence, or (if lacking confidence) 50–250 person-hours at 50 % confidence. Note the trade-off between the interval size so it is possible to increase confidence by enlarging the interval or to decrease the confidence value and reduce the interval accordingly.

An alternative approach sometimes used in industry is derived from the critical path analysis technique Program Evaluation and Review Technique (PERT) (Willis 1985) and is known as 3-point estimation. It is based on the idea that an estimate is actually a probability distribution, and a simple characterisation is as a triangle based on the best case, worst case and most likely case or mode.

Figure 3.1 shows an example of a 3-point estimate depicted as a triangular probability distribution. The shaded area shows the region within which the true or actual effort value will fall (assuming of course that the distribution is correctly estimated). The estimation interval is the range between the worst case (i.e., the highest possible value) for effort and the best case (i.e., the lowest possible value) for effort. In addition, the distribution shows likelihood or the probability p on the y-axis. This reveals that the highest or modal point on the distribution is the most likely, i.e., it has the greatest chance of actually occurring. The distribution also reveals another interesting property, that it is skewed or biased since the region above or to the right of the most likely value is considerably greater than the region below the mode. The implication is that even if the distribution were accurately estimated, to use the most-likely value as the actual estimate will lead to a tendency to under-estimate over time. This is a phenomenon that we observe (as noted in Sect. 3.1).

Although thinking of an estimate as a distribution enables a far richer analysis, empirically we are hindered by the fact that we are obliged to construct the distribution from a single observation. The situation can be further complicated by the fact that projects are seldom static and so one has to be clear whether the estimates refer to a project as intended at its inception as compared with the actual project as delivered which could conceivably have functionality added or removed. These problems are further explored by Grimstad et al. (2006).

There is surprisingly little systematic analysis of what software practitioners actually do. Studies such as those by Heemstra (1992) and Hughes (1996) have reported that expert judgment is the dominant method amongst software practitioners and there is little to suggest matters have changed radically since the 1990s.

One source for identifying what is perceived as good practice is the Software engineering Body of Knowledge (SWEBOK; Abran and Bourque 2004), which was the culmination of the work of a team of software development experts. Interestingly, the section on effort, schedule and cost estimation is relatively brief; however, a number of principles emerge:

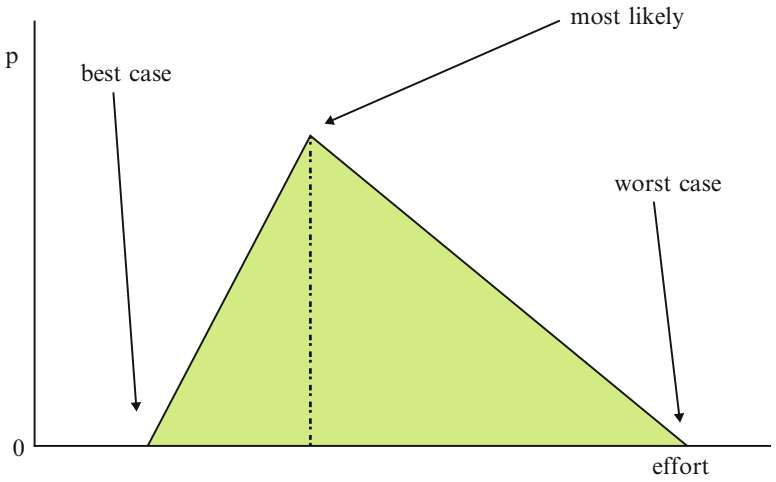

Fig. 3.1 Three point estimates as probability distribution

1. Estimates can be derived *top-down* or by means of some breakdown of tasks.
2. For each such task the expected effort [cost] *range* can be derived from a cost model which needs *calibration* to the local environment using *historical* data if available. Otherwise, an alternative is needed such as expert judgment.
3. The individual estimates should be summed across the entire project.
4. Estimates need to be *revised* iteratively until agreement is reached amongst all stakeholders, which the SWEBOK identifies as principally software engineers and management.

In this list of steps I have italicised some key concepts, which will be explored in more detail.

The idea behind a top-down or a decomposition approach to cost estimation is that of divide and conquer. In other words, it is easier to estimate the cost of a small task than a large one. Moreover, it is easier to match a smaller task to some repertoire of previously completed tasks, than it is for a large task where the combinatorial explosion militates against this possibility. Often, the idea is formalised into work breakdown charts. The chief difficulty is the fact that some activities do not easily fit into neat hierarchical breakdowns.

The next point of note is the SWEBOK recommendation to consider representing an estimate as a range. As previously discussed, in order to view an estimate as a probabilistic statement a point value is inadequate. However, to attach additional meaning minimally, we need a confidence level in the range. Provision of 3-point estimate provides an even richer picture.

SWEBOK also recommend the use of formal models, and although no examples are specified, widely used models include COCOMO 81, which is based on a non-linear relationship between estimated LOC and effort, implying diseconomies of scale. The fundamental relationship is modified by the type of project and,

initially, 14 cost drivers. This was subsequently modified and extended as COCOMO II, although, unfortunately unlike COCOMO 81, the database from which this model is derived is not in the public domain.

Although COCOMO is widely used and there are many free implementations, it has come in for criticism. Firstly, accurate estimates of LOC may not be available at an early stage of a software project. Secondly, there is mixed empirical evidence as to whether software projects exhibit diseconomies (as many commentators assert), economies or simple linearity with respect to scale (Kitchenham 2002). Third, there is limited evidence that COCOMO performs well using the off the shelf settings on data other than that with which it was developed, for example, Kocaguneli et al. (2012b) reported that the model was ranked 92 out of 102 different combinations of models and pre-processors that were evaluated in a major empirical study. Likewise Kemerer (1987) reported mean absolute relative errors in excess of 600 % for a different data set of 15 software projects. Interestingly, he found that COCOMO performed best (least badly?) in its simplest form, and additional sophistication of the model harmed its accuracy. This has led many researchers, in line with the SWEBOK, to recommend tailoring and calibration to a local environment. Gulezian (1991) describes how multiple regression analysis can be used to calibrate the weights for the various cost drivers. The systematic review by Jørgensen (2004) identified individual primary studies and the *only* ones that showed formal prediction systems to outperform experts involved the use of calibration. More recently, Yang et al. (2013) described a calibration procedure to handle local bias, thereby improving the usability of cross-company data sets and demonstrated this with respect to COCOMO II. The value of calibration was again highlighted by the analysis of Menzies et al. (2013). Nevertheless, despite some of the reservations COCOMO or a similar approach is often used as some form of sanity check.

Another important part of the SWEBOK recommendations is the need to revisit any prediction. This has often been neglected by researchers who tend to see a software project as a static snapshot, which of course does not reflect the realities of (1) a growing understanding of the requirements and challenges as the software project plays out, converging upon certainty on the day of delivery and (2) the changing environment in which the project is embedded. MacDonell and Shepperd (2003a) in a rare study of re-estimation in a commercial setting found no support for the idea that there are 'standard proportions' of effort for particular development stages, e.g., specification and design. However, in most cases simple linear regression combined the managers' estimates led to improvements in predictive accuracy. These results indicate that, in this organisation, prior-phase effort data is useful and revising estimates worthwhile.

3.3 A Review of Cost Estimation Research

Because of the need for effective software cost estimation, this has been the subject of a good deal of research. From the outset, the aim has been to replace the subjectivity of project managers and other professionals, generally referred to as

expert judgment with more objective and formal approaches. This was, or still is, seen as a good thing because this provides opportunities for scrutiny, it is more repeatable and can militate against the loss of knowledge and insight if experts leave an organisation.

Early approaches tended to be based on some function between size either measured as estimated LOC or Function Points (Albrecht and Gaffney 1983) and a variant known as Mk II Function Points (Symons 1988). Generically, these take the form:

$$E = f(S^a)$$

where E is effort or cost, S is size (typically measured by LOC or Function Points) and *a* an exponent representing economies or diseconomies of scale. Typically, this overall relationship is then modified by a set of productivity or cost factors. COCOMO 81 (as described in Sect. 3.2) is a good example of this approach. An interesting recent study by Kocaguneli et al. (2012a) has suggested that in many cases, the use of a size measure may be less important than previously supposed. It may be that other features act as a proxy for size, e.g., the different application types may tend to be of different sizes. Nevertheless, it is an thought-provoking point that size may be less essential than has been previously supposed.

Early models were postulated based on the beliefs of the inventor, however, the 1990s heralded a more data-driven approach to modelling. Often, multiple regression methods sometimes using a stepwise approach[2] were deployed in order to isolate the important factors, specific to some software development environment as captured by a data set of historical project data. Kitchenham and Kansala (1993) used multiple regression to re-estimate weightings for the standard values for Function Points with considerable benefit. They also reminded researchers of the dangers of constructing models when many of the components are strongly correlated, i.e., multicollinearity is present which if uncorrected leads to highly unstable models.

Given the emphasis of learning from historical data, different machine learning techniques became popular from the 1990s onwards. In all cases the underlying principle is to reason inductively from the particular to the general. For cost prediction the idea is to learn from past, completed software projects in order to predict for new, unseen projects. One technique is lazy learning[3] based on analogical or case-based reasoning (Shepperd and Schofield 1997; Keung et al. 2008) which is often referred to as Estimation by Analogy (EBA). The simplicity of the idea—that history repeats itself, but not exactly—has attracted a good deal of attention not least because to be acceptable to practitioners, prediction systems

[2] The regression model is constructed one independent variable at a time or iteratively until no new variable *significantly* contributes to the model fit.

[3] A lazy learner only makes an inductive generalization when actually presented with the new problem to solve. This can be advantageous when trying to learn in the face of noisy training cases and much uncertainty.

benefit from good explanatory value since decisions arising from the prediction will be of high value (Mair et al. 2000). Despite these strengths, EBA was not found by a Systematic Review (Mair and Shepperd 2005) of all available empirical studies to outperform simpler regression models with 9 studies supporting, 4 equivocal and 7 against.

Because of the relative ease of fitting regression models these are now often used as a benchmark with which to compare more elaborate methods, e.g., Mair et al. (2000) compared various machine learning methods (artificial neural nets (ANNs), case-based reasoners (CBR) and rule induction) with stepwise regression. Interestingly, the basic regression approach outperformed the rule induction algorithms although not CBR or ANNs.

The last decade could be characterised by research that has explored more advanced prediction systems. Examples include through the use of ensembles of learners coupled with some decision making logic (Minku and Yao 2013) and new approaches like Grey Relational Algebra (Song and Shepperd 2011). This has been supported by more research into such things as data pre-processing as many prediction methods are vulnerable to excessive noise, extreme outliers and missing observations. Consequently, appropriate pre-processing can have a substantial impact upon predictive performance, Strike et al. (2001), Song and Shepperd (2007), Liu and Mintram (2005).

Another area of concern and of some progress is developing frameworks as to how we meaningfully compare the proliferating number of cost estimation approaches. Until the empirical studies of Myrtveit and Stensrud (1999) which set out to independently compare regression modelling, EBA and the unaided human expert, it was not customary to perform any statistical testing. Subsequently, inferential test such as *t*-tests and Mann–Whitney U became the norm, however, methodological problems such as correcting[4] the α threshold for null hypothesis significance testing in the face of large numbers of tests and using inappropriate measures of predictive accuracy remained. Mittas and Angelis have proposed a method that is not too conservative but reduces the number of tests required by means of clustering the results into groups (Mittas and Angelis 2013). More generally, various authors have proposed remedies and strong arguments as to why to proper procedures are required in order to derive sound conclusions. For example, Shepperd and MacDonell (2012) show that inappropriate evaluation hid the fact that various published prediction techniques such as regression to the mean coupled with EBA actually performed worse than guessing!

After the event, when evaluating the quality of a prediction there are three dimensions that need to be assessed (1) error (2) bias and (3) variance or scatter.

[4] Essentially, the point is that when conducting a significance test for a hypothesis, there are two dangers: One can wrongly reject the null hypothesis or wrongly fail to reject the null hypothesis. It is customary to set the chances of wrongly rejecting the null hypothesis (denoted by α) at 0.05. However, if many tests are performed, the probability of at least once committing such an error grows with the number of tests. For this reason, the α threshold needs to be reduced to take this danger into account.

Even accuracy is often misunderstood in the software engineering community and inappropriately assessed by accuracy statistics such as Mean Magnitude of Relative Error (MMRE). Elsewhere researchers show how this is flawed both theoretically as it is merely an asymmetric measure of spread (Kitchenham et al. 2001) and empirically through Monte Carlo simulation (Foss et al. 2003). Without a clear conceptual understanding of accuracy it is difficult for the community to review or improve their prediction practice since there is no systematic basis for evaluating different approaches to cost estimation. Indeed, MMRE has the rather perverse characteristic of favouring optimistic predictions over pessimistic ones. Given the widespread use of MMRE this may be another contributor to the biases we observe in industry practice described in Sect. 3.1. Therefore, unless there is good reason to the contrary, it is recommended (Shepperd and MacDonell 2012) that researchers seek to minimise the absolute sum of the residuals, consider performance relative to guessing and be aware of the effect size. The effect size is a means of capturing the practical or real world effect of the particular intervention, for example by moving from cost estimation technique A to B what actual benefit does this yield? This is a very different question from how likely is the effect to have arisen by chance since large numbers of observations will render even small effects highly significant (Armstrong 2007; Ellis 2010).

The final development, and one that warrants a section in its own right, is the realisation that formal prediction or cost models have not succeeded in replacing humans and therefore there is a need to research into how practitioners make predictions. This section has of necessity been brief. For a more detailed overview see the mapping studies in Jørgensen (2004) and Jørgensen and Shepperd (2007).

3.4 The Interaction Between People and Formal Techniques

As the previous section has shown there has been no shortage of ideas or research into constructing formal prediction systems for software project costs. Unfortunately, as systematic reviews (Mair and Shepperd 2005; Jørgensen and Shepperd 2007; and simulation work Shepperd and Kadoda 2001) demonstrate, no single technique dominates. In particular, formal model performance seems closely linked with the specific characteristics of the historical data that are used to train or calibrate the prediction system (Shepperd and Kadoda 2001). This has led some researchers such as Menzies et al. to suggest that we should focus on finding prediction systems that are ‘good enough’ rather than the ‘best’ (Menzies et al. 2010). Nevertheless, Jørgensen (2004) reported that formal models do not consistently outperform their human counterparts and frequently do less well. Specifically, in his systematic review of 15 primary studies he reports that 5 favoured formal models, 5 were equivocal and 5 favoured expert judgement over the formal model. Looking in more detail, Jørgensen suggests that those

studies using local calibration or where the estimators lacked expertise yielded the best results for formal models. Similarly, in a software maintenance setting, the systematic review of Riaz et al. (2009) found that 'there is little evidence on the effectiveness of software maintainability prediction techniques and models'. Moreover, formal models do not appear to be very widely used in practice and expert judgement remains the dominant estimation technique (Jørgensen 2004). Consequently, Jørgensen and his co-workers have been exploring over the past decade why this might be so.

The first thing to appreciate is the nature and use of cost estimates. Software projects are generally high value and relatively infrequent events since typical durations are many months through to years. Therefore the estimate matters and in a way that predicting if a supermarket customer chooses a cabbage will they also purchase carrots, does not. The career prospects of an individual may be impacted by an estimate and the associated decision-making, e.g., to initiate/cancel a software project. In extremis the financial health or viability of the software development may be impacted. Such awareness may skew the estimation process of individuals. More than 20 years ago Lederer and Mendelow (1999) in their study of cost estimation within information systems projects observed how organisational politics can be inimical to good estimation. Flyvbjerg et al. (2003), Flyvbjerg (2008) in a study of a number of major projects—whilst not specifically related to software—found considerable evidence to support the notion of strategic misrepresentation. This typically manifests itself as a tendency to under-estimate costs and over-estimate benefits because of the desirability of the end goal. In terms of software it may be that professionals might see the potential opportunities of a new project, e.g., improved work prospects, personal development or intellectual challenge. The interesting thing is that formal models may not offer any protection against such phenomena since these models require inputs, many of which must be estimated, for instance COCOMO (as previously indicated) requires the user to estimate delivered LOC which will not normally be known at the point of prediction. Likewise many machine learning techniques are heavily parameterised with little deep theory to guide the user, thus rendering such methods rather experimental in their approach. This can encourage a 'suck it and see' philosophy. Jørgensen and Gruschke (2005) termed this 'expert judgment in disguise'.

The problem of obtaining useful predictions is compounded by the strong tendency for professionals to display both over-optimism, e.g., Buehler et al. (1994) and over-confidence, e.g., Jørgensen (2010). Because these phenomena are so widespread the causes of bias have been extensively investigated by cognitive psychologists in various domains over the past three decades since the seminal work of Kahneman and Tversky (1979). This has led to the identification of a number of cognitive biases that appear to be both deeply ingrained and widespread. Four such biases are now considered.

One problem is the so-called 'planning fallacy' which is the tendency to underestimate project completion times as a consequence of spending time on detailed planning aspects. Buehler et al. (1994) examined the underlying cognitive processes and found that a narrow focus on future plans for the target task led to

neglect of other useful sources of information. In other words, an illusion of control leads to significant over-optimism. Therefore we might expect detailed top-down planning methods such as work breakdown to be vulnerable to this particular bias.

Another source of bias is a preference for case-specific (and recent) evidence over distributional evidence (Tversky and Kahneman 1974; Griffin and Buehler 1999). For example, data suggesting that 8 out of 10 projects are delivered late (i.e., costs and schedule have been under-estimated) might be neglected in preference to evidence suggesting this specific project will be different because staff will be motivated to work harder or because there will be reuse of some software components. This helps us understand why professionals struggle to learn lessons from the past because deep down we believe it will be different next time. The problem is the distributional or frequency related evidence says otherwise and this is usually correct!

A closely related phenomenon is the peak-end rule where the most recent experience dominates even when it is highly atypical. This has been demonstrated in many different arenas including the experiment described in Kahneman et al. (1993) where participants were subjected to modest pain (a hand in icy water) and preferred the worse (in terms of temperature and duration) experience when for the final period the water temperature was raised. In terms of software projects, professionals may recall the final experiences of getting software to work, as opposed to the lengthy previous experiences of failures and debugging. Again this bias can lead to distributional evidence being ignored or neglected and the consequent impact upon estimates.

A third, relevant cognitive theory is the dual-process theory of cognition which leads to a tendency to trust analytic justifications (explanations) over intuitive ones yet to prefer intuitive judgments over analytic ones. One implication is that this is another reason why formal prediction systems can turn into 'expert judgment in disguise' (Jørgensen and Gruschke 2005) as the estimator is seeking 'objective' evidence to support his or her intuitive judgement.

A fourth bias is known as anchoring where data in the request for an estimate can be highly influential even when the estimator is told to ignore it. An example is the experiment by Jørgensen and Grimstad (2012) where professional participants were randomly allocated to two groups, one of which was primed with a high anchor and another with a very low anchor. They were then asked to estimate the same task, namely their own productivity in LOC per work-hour over their last project. Remarkably, the difference in median response between the two groups was almost sevenfold (15 LOC per hour versus 100 LOC per hour). This stable finding—repeated by a number of independent studies—indicates just how vulnerable humans are to these biases and is clearly a major contributor to the some of the cost estimation problems reported at the beginning of this chapter.

These biases are common to many problem domains, and seem independent of individual differences, e.g., the traits of optimism and procrastination (Buehler and Griffin 2003). The limited work investigating de-biasing strategies, e.g., utilising previous experience, such as past project databases, Personal Software Process

(Humphrey 2000) and lessons learned sessions, have not been all that successful, particularly in the field of software engineering prediction. Interestingly, Jørgensen and Grushka found that software professionals were better able to learn lessons for the estimates of others than for their own estimates (Jørgensen and Gruschke 2009).

There are both theoretical and empirical reasons why software practitioners make consistently sub-optimal predictions within software engineering. However, the vast bulk of the psychological research has been conducted using student participants working on problems that are not industry-related (Mair et al. 2009) and therefore Jørgensen's work using software developers has been quite unusual. In addition, the literature has predominantly focused upon understanding factors that contribute to bias. We need to also explore factors that promote de-biasing in realistic settings. In parallel, much research has been undertaken into meta-cognition (i.e., thinking about thinking), particularly in the domain of learning. There is a considerable body of evidence showing that increased metacognitive awareness leads to increased learning and enhanced performance, e.g., Coutinho found a relationship between metacognitive awareness and educational performance (Coutinho 2007). Other researchers have shown that metacognitive skills can be taught (Borkowski et al. 1987; Dawson 2008) and these can potentially militate against some of the cognitive biases described above.

Metacognition can be divided into metacognitive knowledge and metacognitive skills. The former relates to declarative knowledge of the interactions among self, task, and strategy characteristics (Flavell 1979) that can be inaccurate and resistant to change. Clearly, this will be an inhibitor to improving prediction performance. Metacognitive skills on the other hand refer to procedural knowledge for self-regulating problem solving and learning activities and include feedback (reflection) on metacognitive knowledge. This division between metacognitive knowledge and skills is related to that of single and double loop learning popularised by Argyris and Schön (1996).

'Single-loop learning' occurs when goals, values, plans and rules are taken for granted and put into operation rather than questioned. It reduces risk and affords greater control, but severely limits growth and learning. By contrast, 'double-loop learning' involves questioning the fundamental systems that underlie goals and strategies. It results in the questioning of governing variables and may lead to fundamental changes. This double-loop learning is necessary if practitioners and organisations are to make informed decisions in changing and uncertain contexts.

Reflection is a metacognitive skill important for personal and professional development, see for example, Schön (1983), Moon (1999), and it plays a key role in both single and double loop learning. However critical reflection, as demonstrated in double loop learning, is essential for growth and change. Critical reflection demands focusing on the cognitive aspects and challenging the strategies that led to particular actions, and the outcomes and lessons learned from those actions for future application.

Unfortunately, previous studies of software project cost prediction suggest that feedback on performance and the typical methods for reflecting on experience, e.g., unaided lessons learned sessions, do not necessarily lead to improvement in

accuracy or assessment of the uncertainty (Jørgensen and Gruschke 2009). The lack of training in both reflecting on one's own thinking and the fundamental causes of suboptimal outcomes (double-loop learning) can be a major obstacle. As an illustration, in a previous study where software professionals described reasons for their estimation errors (Jørgensen and Gruschke 2009; Moløkken and Jørgensen 2004), most were shallow and corresponded to single-loop learning. In particular the participants (all software professionals) exclusively focused on reasons for their estimation inaccuracy and at the expense of their confidence. Indeed, participants only identified means to improve their accuracy (e.g., add more time for unknown events). The alternative, which would have been to change their level of confidence in the effort estimates, was not considered in terms of documented reflections. This lack of double-loop learning would seem to be a key contributor to the robust findings on over-optimism and over-confidence among software developers (Note, in contrast Chap. 7 takes a more organisational perspective to learning. It also uses the device of a decision rationale to support future learning.)

Hence it is important to consider estimation approaches that are underpinned by theories of meta-cognition and double-loop learning. Specifically, we need to better understand the impact of enhanced metacognitive awareness on the ability to improve project cost prediction and confidence (uncertainty assessment) within a software engineering context. To summarise,

1. Formal prediction systems are not consistently reliable or superior to the unaided human expert. Moreover, their inputs and parameters must be manipulated by humans with a consequent loss of their raison d'être, i.e., objectivity.
2. There is a strong tendency for professionals to display over-optimism and over-confidence. A number of experiments and empirical studies help us to understand the cognitive basis for this bias.
3. De-biasing strategies based upon utilising previous experiences, such as lessons learned sessions, have not led to noticeable improvement in prediction accuracy or the realism of uncertainty assessment.
4. There are opportunities to apply recent results from metacognition research to counteract this natural bias and consequently improve performance.

It is therefore evident that more attention needs to be paid both by researchers and practitioners into the cognitive aspects of cost estimation. To ignore this aspect is to severely limit the reach and impact of any initiatives to improve cost estimation practice. As has already been noted, formal models such as those based on machine learning algorithms have their place, but they still depend upon inputs and parameters supplied by, and outputs utilised by, software professionals who are subject to the same cares, concerns and biases of all human beings.

3.5 Practical Recommendations

Thus far, this chapter has noted the importance of effective cost estimation for software projects and contrasted this with the widespread challenges that are faced. In particular, endemic over-optimism has led to costs being systematically under-estimated and over-confidence, causing estimators to believe they are more accurate than they really are. We have then reviewed the problems that are currently being experienced in terms of cost estimation, most notably the tendency to be over-optimistic (i.e., under-estimate costs) and to be over-confident (i.e., be less accurate than anticipated). This has triggered a good deal of research to try to overcome these problems, in particular through proposing formal prediction systems or models. After initial work based on the idea of generally applicable models such as COCOMO (Boehm 1984) and COCOMO II (Boehm et al. 2000), the dominant idea driving formal models has been to derive them from historical data either through statistical analysis such as regression modelling or through induction using one of the many machine learning techniques available. Despite this activity, it is not possible to strongly recommend any one formal technique, for the simple reason of a lack of consistent evidence. Thus, any recommendations must be grounded in the understanding that human judgement plays a substantial contribution.

Whilst not intended to be exhaustive, the following is a list of six practical recommendations that are supported by empirical evidence and could usefully be deployed in real-life projects:

1. Data driven
2. Sensitivity analysis
3. Multiple techniques
4. Group estimates
5. Training and reflection
6. Estimation and confidence

Data-driven estimation requires the availability of historical data on previously completed projects. Such data can be useful in three different ways. First, for analogical reasoning that can be formalised as case-based reasoning (Shepperd and Schofield 1997) or used more informally. Second, local historical data can be used for calibration purposes since there is widespread evidence to indicate that off-the-shelf approaches are problematic and that general purpose models benefit from calibration to the specific or local problem domain (Cuelenaere et al. 1987; Jeffery and Low 1990; Gulezian 1991; Yang et al. 2013). Third, for direct predictive model building, relevant, local data is necessary for training, i.e., inductive learning purposes. Naturally, the question arises about the situation when no local data is available. This might be because the software development organisation is new or because no relevant past data exists. Is the assumption that global data is inferior to local data well founded? This has vexed researchers for some time and two systematic reviews (Kitchenham et al. 2007; MacDonell and Shepperd 2007)

have concluded that the evidence is mixed, and from primary studies available no definitive answer is possible. In some ways, the question of local vs. global data is somewhat artificial and more relevant is how relevant is the global or cross-company data? However, a recommendation is to inform any cost estimation with local data, including past estimation performance data wherever possible. If circumstances do not allow this, then global data, after careful consideration of its relevance, is the next best option.

Sensitivity analysis is not common practice, yet in the face of uncertainty, it is a very useful means of determining the vulnerability of an estimate to particular assumptions and the level of confidence that can be placed in that estimate. Such analysis can be highly sophisticated (Saltelli et al. 2000) or use simple Monte Carlo methods (Fishman 1996). Wagner (2007) illustrates how these ideas can be deployed using a COCOMO model and finds that the code size estimate dominates the effort prediction, but less obviously that there are significant second order effects between the different cost drivers due to the multiplicative nature of the model. This kind of analysis can also be valuable when the uncertainty surrounding an estimate is unacceptable, thereby helping the estimator identify the most important sources of variability and could then take steps to reduce this uncertainty through further investigation, simulation, etc. of the key parameters or inputs.

Using more than one estimation method or *multiple techniques* is another important consideration. Although an obvious recommendation for practitioners, this has not been widely researched and the evidence base is quite limited. Kitchenham et al. (2002) conducted an empirical study of 145 projects at a large software house where estimators were required to use a minimum of two techniques and then select one estimate to be the basis of client-agreed budget. The advantage, over simply using the mean is that if one estimate is misleading it will not 'contaminate'. MacDonell and Shepperd (2003b) explored a similar question and also found that not only was no one technique best but using the mean was also sub-optimal. By selecting one technique, or perhaps investigating more deeply, requires more consideration and discussion than the formulaic application of an averaging technique.

Group estimates should also be considered as a practical estimation technique. Again, surprisingly considering they have been promoted since Boehm's seminal *Software Engineering Economics* (Boehm 1981) described a wideband Delphi process, but there has been limited research and therefore evidence. Taff et al. (1991) proposed a related approach that they termed Estimeetings, however, little empirical support is offered in terms of their effectiveness. Passing and Shepperd (2003) investigated the impact of group discussion and iterated estimates and found that both checklists and group discussions significantly contribute to improved estimation. The limitation of this study was that it involved Masters students rather than professionals and was in an artificial setting. Reporting similar results, Moløkken and Jørgensen (2004) found a significant and substantial effect in terms of the tendency for group estimates to be less optimistic both for group decisions and the individual post-group discussion decisions to be less optimistic than the original estimates.

The lack of systematic *training and reflection* is another improvement opportunity. As Jørgensen puts it 'the focus on learning estimation skills from software development experience seems to be very low' (Jørgensen 2004). The challenges are that the various cognitive biases described in Sect. 3.4 are deeply ingrained and de-biasing strategies not necessarily effective. Consequently, emphasis should be given to reflection but structured in order to guide estimators beyond the shallow reflections that some researchers have found, such as 'the estimate was too low because insufficient time was allocated'! Researchers have also found that emphasising metacognitive skills can also significantly improve performance.

Finally, practitioners need to keep in mind that because an estimate is a probabilistic statement, it has two dimensions (the *estimate* and *confidence*) and therefore, it is not well represented by a single point value even if this is required as the final outcome of the decision making process, e.g., the bid value. To give an example, estimating 1,000 person-hours $\pm$ 10 person-hours is a very different proposition to 1,000 person-hours $\pm$ 500 person-hours. Even this may not be adequate since it is unclear whether it means that an actual effort of 1,510 person-hours is deemed impossible or merely very unlikely. Moreover such a formulation imposes a symmetric distribution which may not properly reflect the estimator's beliefs. Jørgensen recommends a confidence value in a range, e.g., 80 % confidence between 500 and 1,500 person-hours. This allows some simple trade-offs between precision and confidence to be exploited. A richer picture still is obtained by describing the estimate as a probability distribution, e.g., as a 3-point estimate and a triangular distribution. Either way, failing to regard estimates as probabilities indicates a failure to appreciate their true nature and therefore the opportunity to learn and improve.

The above list contains some simple, practical, general and evidence-based recommendations for software cost estimation. It is not a panacea, and there are many other challenges that have not been fully addressed. Nevertheless, given the importance of software, software projects and effective cost management, they may offer some useful steps forward.

3.6 Follow-Up Sources of Information

There are several comprehensive systematic reviews on research into cost estimation. Jørgensen and Shepperd (2007) gives general coverage of the different research activities being undertaken and Simula have continued to update the database of sources subsequent to its publication.[5] A second, more specialised on the role of human experts, and slightly older systematic review is by Jørgensen (2004). The review by Riaz et al. (2009) focuses on cost estimation in a software maintenance context.

[5] The bibliographic database can be found at www.simula.no/BESTweb

Cost estimation generally takes place in the wider setting of a software project. There are many good textbooks, such as Hughes and Cotterell (2009) on project management and Sommerville (2010) on software engineering and the set of guidelines published as the SWEBOK (Abran and Bourque 2004).

In terms of making sense of published empirical research comparing different formal models, and for designing new experiments, Shepperd and MacDonell (2012) set out a framework based on three research questions that need to be addressed.

Glossary

Absolute residuals a simple and robust means of assessing the predictive accuracy of a prediction system. It is defined simply as: $|y_i - \hat{y}_i|$ where y_i is the true value for the ith project and $\hat{y}_i$ the estimated value. This gives the error, irrespective of direction, i.e., an under- or over-estimate. The mean residual (keeping the direction of error) gives a measure of the degree of bias.

Cognitive bias these are patterns of thinking about problem solving or decision-making that distort and lead people to 'sub-optimal' choices. Because of the ubiquity of many such biases, they are classified and named, e.g., the anchoring bias. See the pioneering work of Tversky and Kahneman (1974).

Double loop learning this differs from ordinary or single-loop learning in that one not only observes the effects of the process, but also understands the external factors that influence the effects. This was initially promoted by Argyris and Schön as a way of promoting effective organisational behaviour (Argyris and Schön 1996).

Estimation by Analogy (EBA) uses some form of case-based reasoning where a new or target case which is to be solved is plotted in feature space (one dimension per feature) and some distance metric used to determine past proximal cases from which a solution can be derived. For a general account of CBR see the pioneering work by Kolodner (1993) and for its application to software engineering see Shepperd (2003).

Expert Judgement this is something of a catch all description for a range of informal approaches to estimation. Jørgensen describes it as 'unaided intuition ("gut feeling") to expert judgment supported by historical data, process guidelines and checklists ("structured estimation")' (Jørgensen 2004). Despite it being a widespread estimation approach, it can still be criticised for its reasoning not being open to scrutiny since the reasoning process is 'non-recoverable' (Jørgensen 2004), not repeatable or easily transferable from existing experts to others.

Formal prediction system or formal model for cost prediction is characterised by repeatability so that different individuals applying the same inputs should generate the same outputs (with the exception of prediction systems based on

stochastic search [also see Chap. 15 on search-based project management] where this will tend to be true over time (Clark et al. 2007), but not necessarily for a single utilisation). Examples of formal systems range from simple algorithmic models, such as COCOMO, to complex ensembles of learners.

Machine Learning this is a branch of applied artificial intelligence based on inducing prediction systems from historical data, i.e., reasoning from the particular to the general. There are a wide range of approaches including neural networks, case-based reasoning, rule induction, Bayesian methods, support vector machines and population search methods such as genetic programming. Standard textbooks that provide overviews of these techniques include Witten et al. (2011).

Mean magnitude of relative error (MMRE) this is a widely used, although now heavily criticized (Kitchenham et al. 2001; Foss et al. 2003; Shepperd and MacDonell 2012), measure of predictive accuracy defined as:

$$MMRE = \sum_{1}^{n} \left(\left| {}^{(x_i - \hat{x}_i)} \Big/ {}_{x_i} \right| \right) \Big/ {}_{n}$$

where x is the true cost for the ith project, $\hat{x}$ is the estimated cost and n the total number of projects.

Metacognition this refers to 'thinking about thinking' (Flavell 1979) and is an awareness and monitoring of one's thoughts and performance. It encompasses the ability to consciously control the cognitive processes involved in learning such as planning, strategy selection, monitoring and evaluating progress towards a particular goal and adapting strategies as, and when, necessary to reach that goal (Ridley et al. 1992).

Over-confidence refers to the tendency of an estimator to value precision over accuracy. Typically, one might express confidence in an estimate as the likelihood that the true value falls within a specified interval. For example, stating that one is 80 % confident that the actual effort will fall within the range 1,000–1,200 person-hours implies that this will occur 8 out of 10 times. If the true value falls into the range less frequently this implies over-confidence. Jørgensen et al. (2004) reported that over-confidence was a widespread phenomenon and that at least one contributor was the fact that managers often interpret wide intervals as conveying a lack of knowledge and prefer narrow but less accurate estimates.

Over-optimism refers to the situation where the estimation error is biased towards an under-estimate. Many studies indicate that this is the norm in the software industry with a figure of 30 % being seen as typical (Jørgensen 2004).

Prediction whilst 'prediction' and 'estimation' are often used interchangeably, we use 'prediction' to mean a forecast or projection, and 'estimate' to connote a guess or rough and ready calculation.

Single-loop learning Argyris and Schön (1996) characterise this as focusing on restrictive feedback so that the individual or organisation only endeavours to improve a single metric without external reflection upon the process, i.e., double loop learning.

References

Abran A, Bourque, P (2004) SWEBOK: guide to the software engineering body of knowledge. IEEE Computer Society

Albrecht AJ, Gaffney JR (1983) Software function, source lines of code, and development effort prediction: a software science validation. IEEE Trans Softw Eng 9:639–648

Argyris C, Schön D (1996) Organizational learning II: theory, method and practice. Addison-Wesley, Reading, MA

Armstrong S (2007) Significance tests harm progress in forecasting. Int J Forecast 23:321–327

Benington HD (1956) Production of large computer programs. In: Symposium on advanced computer programs for digital computers, ACR-15

Boehm BW (1981) Software engineering economics. Prentice-Hall, Englewood Cliffs, NJ

Boehm BW (1984) Software engineering economics. IEEE Trans Softw Eng 10:4–21

Boehm B, Abts C, Brown W, Chulani S, Clark BK, Horowitz E, Madachy R, Reifer D, Steece B (2000) Software cost estimation with COCOMO II. Pearson/Prentice Hall, Englewood Cliffs, NJ

Borkowski JG, Carr M, Pressley M (1987) Spontaneous strategy use: perspectives from metacognitive theory. Intelligence 11:61–75

Buehler R, Griffin D (2003) Planning, personality, and prediction: the role of future focus in optimistic time predictions. Organ Behav Hum Decis Process 92:80–90

Buehler R, Griffin D, Ross M (1994) Exploring the "Planning Fallacy": why people underestimate their task completion times. J Pers Soc Psychol 67:366–381

Clark J, Dolado JJ, Harman M, Hierons RM, Jones B, Lumkin M, Mitchell B, Mancoridis S, Coutinho SA (2007) The relationship between goals, metacognition, and academic success. Educate 7:39–47

Coutinho SA (2007) The relationship between goals, metacognition, and academic success. Educate 7:39–47

Cuelenaere A, van Genuchten M, Heemstra F (1987) Calibrating a software cost estimation model - why and how. Inf Softw Technol 29:558–567

Dawson TL (2008) Metacognition and learning in adulthood. Developmental Testing Service LLC, Northampton, MA

DeMarco T (1982) Controlling software projects: management, measurement and estimation. Yourdon Press, New York

El Emam K, Koru G (2008) A replicated survey of IT software project failures. IEEE Softw 25:84–90

Ellis P (2010) The essential guide to effect sizes: statistical power, meta-analysis, and the interpretation of research results. Cambridge University Press, Cambridge

Fishman G (1996) Monte Carlo: concepts, algorithms, and applications. Springer, New York

Flavell JH (1979) Metacognition and cognitive monitoring: a new area of cognitive-developmental inquiry. Am Psychol 34:906–911

Flyvbjerg B (2008) Curbing optimism bias and strategic misrepresentation in planning: reference class forecasting in practice. Eur Plan Stud 16:3–32

Flyvbjerg B, Bruzelius N, Rothengatter W (2003) Megaprojects and risk: an anatomy of ambition. Cambridge University Press, Cambridge

Foss T, Stensrud E, Kitchenham B, Myrtveit I (2003) A simulation study of the model evaluation criterion MMRE. IEEE Trans Softw Eng 29:985–995

Griffin D, Buehler R (1999) Frequency, probability, and prediction: easy solutions to cognitive illusions? Cogn Psychol 38:48–78

Grimstad S, Jørgensen M, Moløkken-Østvold K (2006) Software effort estimation terminology: the tower of Babel. Inf Softw Technol 48:302–310

Gulezian R (1991) Reformulating and calibrating COCOMO. J Syst Softw 16:235–242

Heemstra FJ (1992) Software cost estimation. Inf Softw Technol 34:627–639

Hughes RT (1996) Expert judgement as an estimating method. Inf Softw Technol 38:67–75

Hughes RT, Cotterell M (2009) Software project management. McGraw-Hill, London
Humphrey W (2000) Introducing the personal software process. Ann Softw Eng 1:311–325
Jeffery DR, Low GC (1990) Calibrating estimation tools for software development. Softw Eng J 5:215–221
Jørgensen M (2004) A review of studies on expert estimation of software development effort. J Syst Softw 70:37–60
Jørgensen M (2010) Identification of more risks can lead to increased over-optimism of and over-confidence in software development effort estimates. Inf Softw Technol 52:506–516
Jørgensen M, Grimstad S (2012) Software development estimation biases: the role of interdependence. IEEE Trans Softw Eng 38:677–693
Jørgensen M, Gruschke T (2005) Industrial use of formal software cost estimation models: expert estimation in disguise? In: Proceedings of EASE, Keele, UK
Jørgensen M, Gruschke T (2009) The impact of lessons-learned sessions on effort estimation and uncertainty assessments. IEEE Trans Softw Eng 35:368–383
Jørgensen M, Moløkken-Østvold K (2006) How large are software cost overruns? A review of the 1994 CHAOS report. Inf Softw Technol 48:297–301
Jørgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33:33–53
Jørgensen M, Sjøberg DIK (2003) An effort prediction interval approach based on the empirical distribution of previous estimation accuracy. Inf Softw Technol 45:123–136
Jørgensen M, Teigen KH, Moløkken K (2004) Better sure than safe? Overconfidence in judgment based software development effort prediction intervals. J Syst Softw 70:79–93
Kahneman D, Tversky A (1979) Intuitive prediction: biases and corrective procedures. TIMS Stud Manag Sci 12:313–327
Kahneman D, Fredrickson B, Schreiber C, Redelmeir D (1993) When more pain is preferred to less: adding a better end. Psychol Sci 4:401–405
Kemerer CF (1987) An empirical validation of software cost estimation models. Commun ACM 30:416–429
Keung J, Kitchenham B, Jeffery R (2008) Analogy-X: providing statistical inference to analogy-based software cost estimation. IEEE Trans Softw Eng 34:471–484
Kitchenham BA (2002) The question of scale economies in software - why cannot researchers agree? Inf Softw Technol 44:13–24
Kitchenham BA, Kansala, K. (1993) Inter-item correlations among function points. In: 1st International symposium on software metrics. IEEE Computer Society Press, Baltimore, MD
Kitchenham BA, Linkman SG (1997) Estimates, uncertainty and risk. IEEE Softw 14:69–74
Kitchenham BA, MacDonell SG, Pickard L, Shepperd MJ (2001) What accuracy statistics really measure. IEEE Proc Softw Eng 148:81–85
Kitchenham BA, Pfleeger SL, McColl B, Eagan S (2002) An empirical study of maintenance and development estimation accuracy. J Syst Softw 64:57–77
Kitchenham B, Mendes E, Travassos G (2007) Cross versus within-company cost estimation studies: a systematic review. IEEE Trans Softw Eng 33:316–329
Kocaguneli E, Menzies T, Hihn J, Kang H (2012a) Size doesn't matter? On the value of software size features for effort estimation. In: Proceedings of the 8th international conference on predictive models in software engineering, New York
Kocaguneli E, Menzies T, Keung J (2012b) On the value of ensemble effort estimation. IEEE Trans Softw Eng 38:1403–1416
Kolodner JL (1993) Case-based reasoning. Morgan-Kaufmann, San Mateo, CA
Lederer A, Mendelow A (1999) The impact of the environment on the management of information systems. Inf Syst Res 1:205–222
Liu Q, Mintram R (2005) Preliminary data analysis methods in software estimation. Softw Qual J 13:91–115
MacDonell S, Shepperd M (2003a) Using prior-phase effort records for re-estimation during software projects. In: 9th IEEE international metrics symposium

MacDonell S, Shepperd M (2003b) Combining techniques to optimize effort predictions in software project management. J Syst Softw 66:91–98
MacDonell S, Shepperd MJ (2007) Comparing local and global software effort estimation models – reflections on a systematic review. In: 1st international symposium on empirical software engineering and measurement, Madrid
Mair C, Shepperd M (2005) The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In: 4th international symposium on empirical software Engineering (ISESE) Noosa Heads, Australia
Mair C, Kadoda G, Lefley M, Keith P, Schofield C, Shepperd M, Webster S (2000) An investigation of machine learning based prediction systems. J Syst Softw 53:23–29
Mair C, Martincova M, Shepperd M (2009) A literature review of expert problem solving using analogy. In: 13th international conference on evaluation and assessment in software engineering (EASE), British Computer Society, Swinton, UK
Menzies T, Jalili M, Hihn J, Baker D, Lum K (2010) Stable rankings for different effort models. Autom Softw Eng 17:409–437
Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F, Turhan B, Zimmermann T (2013) Local versus global lessons for defect prediction and effort estimation. IEEE Trans Softw Eng 39:822–834
Minku L, Yao X (2013) Ensembles and locality: insight on improving software effort estimation. Inf Softw Technol 55:1512–1528
Mittas N, Angelis L (2013) Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Trans Softw Eng 39:537–551
Moløkken K, Jørgensen M (2004) Group processes in software effort estimation. Empir Softw Eng 9:315–334
Moon J (1999) Reflection in learning and professional development: theory and practice. Kogan Page, London
Myrtveit I, Stensrud E (1999) A controlled experiment to assess the benefits of estimating with analogy and regression models. IEEE Trans Softw Eng 25:510–525
Passing U, Shepperd M (2003) An experiment on software project size and effort estimation. In: ACM-IEEE international symposium on empirical software engineering (ISESE 2003)
Riaz M, Mendes E, Tempero E (2009) A systematic review of software maintainability prediction and metrics. In: 3rd international symposium on empirical software engineering and measurement, ACM Computer Press, pp 367–377
Ridley D, Schutz P, Glanz R, Wernstein C (1992) Self-regulated learning: the interactive influence of metacognitive awareness and goal-setting. J Exp Educ 60:293–306
Saltelli A, Tarantola S, Campolongo F (2000) Sensitivity analysis as an ingredient of modeling. Stat Sci 15:377–395
Schön DA (1983) The reflective practitioner. Basic Books, New York
Shepperd M (2003) Case-based reasoning and software engineering. In: Aurum A, Jeffery R, Wohlin C, Handzic M (eds) Managing software engineering knowledge. Springer, Berlin
Shepperd MJ, Kadoda G (2001) Comparing software prediction techniques using simulation. IEEE Trans Softw Eng 27:987–998
Shepperd M, MacDonell S (2012) Evaluating prediction systems in software project estimation. Inf Softw Technol 54:820–827
Shepperd MJ, Schofield C (1997) Estimating software project effort using analogies. IEEE Trans Softw Eng 23:736–743
Sommerville I (2010) Software engineering. Pearson, Hemel Hempstead, UK
Song Q, Shepperd M (2007) Missing data imputation techniques. Int J Bus Intell Data Mining 2:261–291
Song Q, Shepperd M (2011) Predicting software project effort: a grey relational analysis based method. Expert Syst Appl 38:7302–7316
Strike K, El Emam K, Madhavji N (2001) Software cost estimation with incomplete data. IEEE Trans Softw Eng 27:890–908

Symons CR (1988) Function point analysis: difficulties and improvements. IEEE Trans Softw Eng 14:2–11

Taff LM, Borcering JWB, Hudgins WR (1991) Estimeetings: development estimates and a front-end process for a large project. IEEE Trans Softw Eng 17:839–849

Tversky A, Kahneman D (1974) Judgment under uncertainty: heuristics and biases. Science 185:1124–1131

Wagner S (2007) An approach to global sensitivity analysis: FAST on COCOMO. In: 1st International symposium on empirical software engineering and measurement (ESEM 2007). IEEE Computer Society, pp 440–442

Whitfield D (2007) Cost Overruns, delays and terminations: 105 outsourced public sector ICT contracts. The European Services Strategy Unit

Willis R (1985) Invited review: critical path analysis and resource constrained project scheduling—theory and practice. Eur J Oper Res 21(2):149–155

Witten I, Frank E, Hall M (2011) Data mining: practical machine learning, tools and techniques. Morgan Kaufmann, Burlington, MA

Yang Y, He Z, Mao K, Li Q, Nguyen V, Boehm B, Valerdi R (2013) Analyzing and handling local bias for calibrating parametric cost estimation models. Inf Softw Technol 55:1496–1511. Software Engineering Body of Knowledge (SWEBOK). Software Engineering Body of Knowledge (SWEBOK) Home. http://www.computer.org/portal/web/swebok/home

Biography Martin Shepperd received a Ph.D. in computer science from the Open University in 1991 for his work in measurement theory and its application to empirical software engineering. He is Head of Department and holds the chair of Software Technology and Modelling at Brunel University, London, UK. He has published more than 150 refereed papers and 3 books in the areas of software engineering and machine learning. He is a fellow of the British Computer Society.