Has Twenty-five Years of Empirical Software Engineering Made a Difference?

Ross Jeffery and Louise Scott
CAESER
School of Computer Science and Engineering
University of New South Wales
Sydney 2052 Australia
R.Jeffery@unsw.edu.au

Abstract

Our activities in software engineering typically fall into one of three categories, (1) to invent new phenomena, (2) to understand existing phenomena, and (3) to facilitate inspirational education. This paper explores the place of empirical software engineering in the first two of these activities. In this exploration evidence is drawn from the empirical literature in the areas of software inspections and software cost modelling and estimation. This research is then compared with the literature published in the Journal of Empirical Software Engineering. This evidence throws light on aspects of theory derivation, experimental methods and analysis, and also the challenges that we face as empirical software engineering evolves into the future.

1. Introduction

Empirical software engineering is concerned with the scientific measurement, both quantitative and qualitative, of software engineering process and product.

Activities in software engineering research typically fall into one of three categories,

(1) To invent new phenomena,

(2) To understand existing phenomena, and

(3) To facilitate inspirational education.

Empirical software engineering has a core place in all of these activities. In the first of these, empirical software engineering should be used in conjunction with programs that develop new phenomena to provide evidence of the benefits of the phenomena. The second activity is largely industrially based and driven by industry needs and vision. In this area it could be said that empirical software engineering is concerned with theory building based on empirical evidence. The third category is where empirical techniques are used in the education process so that measurement occurs in the classroom and understanding is provided through that measurement. In this paper we focus on the first two of these areas.

Although we will not attempt to identify any "starting point" for empirical software engineering, this paper discusses work carried out since 1977 and published in the software engineering literature since around that time. Two areas of research are the focus for the early part of this paper: (1) the work in software cost modelling and cost estimation, and (2) work in software inspections. The former is one of the more mature empirical research areas and the latter one that has received considerable attention in recent years.

The questions we ask are:

- How has empirical software engineering been carried out?
- What impact has empirical software engineering had in both the industrial and academic field?
- How could empirical software engineering practices be improved?

We answer these questions by using data in detail from the two research areas mentioned above and in general from publications in the Journal of Empirical Software Engineering.

Based on the investigation we find that:

- The most common paradigms used are understanding (exploration) and evaluation. Replication, theory revision and development of empirical SE methods are poorly represented.
- There has been relative success in cost estimation with a healthy academic community, healthy relationship with industry but poor penetration in the application of empirical results.
- Empirical software engineering could be improved with more replication and theory revision, more explicit theory statements and independent evaluation, and more reflection and development of empirical SE methods.

In the next two sections we will explore from an historical perspective two quite different examples of empirical software engineering, that of software cost modelling and estimation and that of software inspections. In Section 4 we put each investigation into the context of a model of scientific enquiry and explore the wider empirical software engineering literature to determine how empirical software engineering has been carried out. In Section 5 we discuss our results and Section 6 concludes the paper.

2. Software cost modelling and estimation

Since 1977 the empirical software engineering research group at UNSW have sought to better understand the process of software development. In the initial studies we sought to measure programming productivity [1, 2], those factors which influenced it, and the extent to which it varied between organizations. From this work we moved to the software project level and carried out similar studies [3]. This work was concerned largely with understanding the industrial relationships between effort, size, elapsed time and cost drivers for effort. From here we moved to understanding the productivity impact of specific technologies including CASE tools and fourth generation languages [4-7]. One aspect that arose from all of this earlier research was the need for an early lifecycle size metric, which led to the work on function points and other related metrics [8-13]. Other metric related work was pursued in the area of software complexity [14] and also in estimation [15] and cost modelling [16, 17].

It is interesting to stand back and summarise the achievements of the twenty-five years of research into cost models and estimation. The most significant work of the late 1970s in the cost modelling area was clearly that of Barry Boehm [18] (COCOMO). This work had enormous academic and industrial impact. It provided the foundation for academic work in testing both the cost model itself and the cost drivers identified in that model. A very large number of software development organisations around the world used the cost model and instantiated the drivers for their own environments. Interestingly, this work was based on a study of only sixty-three systems in TRW. Although this may seem a small data set given the wide-ranging influence of the work, at that time we could not identify even a single company in Australia that had collected the kind of data that was need for this work.

Our work at The University of New South Wales at this time was not concerned with the entire software development process, but rather just the programming process. We were able to establish simple insights such as that programming productivity averaged around ten lines of code per hour from receipt of program specification to delivery of unit-tested code. This was the case for some thirty or so organisations studied. We also found that the only factor that significantly modified this rate was reuse and that an organisation could achieve a productivity increase of around twenty percent through code reuse. An interesting insight is that although these results in 1977 clearly suggested reuse as a fruitful line of further enquiry, significant research into reuse did not occur until later and we did not pursue research into reuse at all.

At around the same time Larry Putnam was publishing his work into cost modelling in IEEE Transactions on Software Engineering [19]. His model was quite different in form to that of Boehm, combining effort and schedule so that the trade-off between them was explicitly recognised in the one model. The work in [3] tested this model in the Australian industrial context and found that, for this environment, the relationships between size, effort and elapsed time were much more complex than represented in the model derived by Putnam. The data revealed that elapsed time could be shortened or lengthened with varying impact on effort. The picture shown in Figure 1 reveals part of the reason for this. In this figure we see that an organisation can experience increasing or decreasing returns to scale. The size at which the change occurs will depend on many factors, such as the technology in use, the organisational experience with the size, the project management regime in place, and the extent of reuse. In terms of Brooks law, adding more staff to a late project may or may not make it later depending on many factors, not the least of which is the particular turning point for the project at hand. In the Australian context, with hindsight it was possible to find projects that were beyond the turning point. The Canberra Mandata project (Australian Government) was one of these. Perhaps CS90 (Westpac) was another. On these projects, more people probably did slow the work further. However, the data used in [3] did not have a single case where decreasing returns to scale were evident.

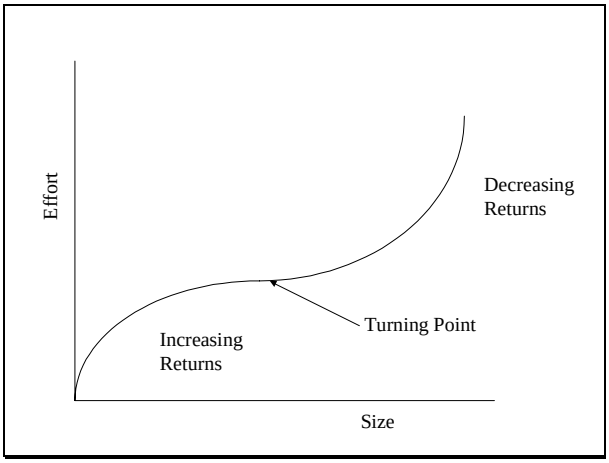

Figure 1: Returns to scale in software development.

One organisation that we studied in the 1970s showed a productivity rate that was much higher than others in the sample of around thirty organisations. When the development manager was asked to explain, we expected to find that the data was inaccurate. Instead we got a response "of course". He said, "I follow a principle of staff quarantine. Once we have a well-developed architecture, rather than have a team of eight people on

the project, I use three or four smaller teams working on independent elements of the project. These smaller groups do not need to communicate with each other, as the components they are working on are clearly defined and independent of each other except as defined in the overall architecture. Avoiding this communication overhead provides us with a productivity advantage." Doesn't this sound a little like "Agile" methods? Yet it was the 1970s. How slow we are to learn! Or is it just that we forget?

From these early studies we learnt that the "rules" of software engineering economics were far from simple and that developing general cost models was certainly possible but that their application in an organisation would require identification of appropriate cost drivers and their values, and identification of appropriate functional form in the unique organisation. However we were far from understanding how we could generalise the process so that it could be applied in any organisation.

As we stated above, one issue that arose from this work was the need for an early lifecycle size metric. This led to our investigation into the function point metric that was begun in 1985 by Loo Chong Chai in her undergraduate thesis [20]. Later work involving Graham Low and John Stathis resulted in an excellent understanding of the strengths and weaknesses of the metric and formed the basis for measurement research, which continues today. Of particular note is the manner in which this measurement research was instrumental in the formation of the Australian Software Metrics Association which later led to the formation of the International Software Benchmarking Standards Group (ISBSG) (see http://www.isbsg.org.au/html/index2.html).

Thus by the late 1980s you could say that we had a good understanding of software cost drivers, cost models and size measurement which allowed us to turn our attention to cost estimation which was the observed industrial need. In [15] we summarised the state of the art in software development cost estimation and argued for the need for both analogical methods as well as algorithmic methods. The cost modelling work had revealed the importance of individual difference and the difficulty in collecting organisational data for either model development or tailoring. It seemed to us that a method was needed that could take advantage of the relatively consistent appearance of certain cost drivers in the models, such as team experience with the technology and maximum project staffing levels, and add to this input from the experience of the management of projects in the organisational setting. The goals were to lower the data entry hurdle, to provide a method for tailoring the models, to facilitate calibration to the local environment, and to minimise the organisational cost of using the estimation process while maximising the accuracy of the estimates. These have been achieved we feel in the recent application of the CoBRA method [21] to the environment in Allette Systems in Sydney. In this work we modified the CoBRA methodology in several ways to (1) allow sizing of web applications using "Web Objects", (2) introduce a cost driver review meeting to check on the managers understanding of the cost drivers and their weightings, (3) introduce analogy into the sizing task for future estimates and (4) remove cost factor interdependencies through the use of verified orthogonal cost driver definitions and measurement. With this method we have been able to obtain high regression accuracy of estimates with quantitative data collected on only a small number of projects (around ten).

Thus if we look back over this period of work by us and many others we see a picture of focused and successful research usually involving an academic/industry partnership, the development of theory and models which lead to understanding, and over the years many instances of the successful adoption of the research outcomes to the industrial software engineering setting.

3. Software inspections

Academic empirical research into software inspections has occurred more recently than the work in cost modelling. The majority of the work in this area has been laboratory based, with much of it concerned with testing new techniques rather than aimed at understanding or developing theory to explain phenomena. In this sense the work is largely concerned with research area (1) above.

The seminal empirical work in this area was probably the 1995 paper by Porter et al. [22]. Although there had been much written about inspections since the time of Michael Fagan's work in 1976 there had been almost no published empirical study of the benefits of the process. Parnas and Weiss [23] argued that systematic processes were needed for effective reviews. In the 1995 Porter paper, student subjects inspected specification documents for a cruise control system and a water level monitoring system. An elevator control system specification was used for training. Comparison was made between ad hoc inspection, checklists and scenarios. It was found that scenarios improved defect identification by roughly 35%. These scenarios were defect based. Checklists were no more effective than ad hoc methods. Moreover, collection meetings contributed "nothing to fault detection effectiveness". In the 1996 paper [24] perspective based reading is explored in inspecting requirements documents. The results in this experiment are not clear however, with many experimental issues observed in the paper. In 1997 Fusaro et al. [25] replicated the 1995 Porter et al. work, again with student subjects. In this experiment only the finding concerning the collection meeting was confirmed.

The 35% improvement reported in the original experiment was not confirmed. This was also the case for the 1998 experiment of Sandahl et al. [26]. Again in 1998 Johnson and Tjahjono found that the group meeting did not identify significantly more defects [27]. A replication of the 1995 study was also completed in 1998 by Miller et al. [28] but again the results are not clear and the support for the first improvement result is only weak. The authors describe the results as "ambiguous".

At this point the empirical research gets really interesting. In 1998 Porter and Votta published a paper [29] that replicated the original 1995 work and confirmed the results! The paper makes no reference to the intervening studies. Why is this so? It appears that an attempt at theory formulation and, importantly, theory revision, has been omitted from the research.

In January 2000 we published a paper devoted entirely to the construction of a theory of group software inspections [30]. This was based on an earlier technical report published in 1996. Although the theory is not complete, in the sense of explaining all aspects of software inspections, it focuses on the behavioural aspects of individual and group work in the inspection setting. Figure 2 shows a schematic summary of the behavioural theory presented. In this theory we posit that review performance of a group is a function of group expertise, group process and the social decision scheme used in the group meeting. Group process influences the social decision scheme. Group expertise, in turn, is determined by individual expertise in the group and group size. In turn, task training and group member selection determine individual task expertise available to the group. In this paper we present eleven research propositions derived from the theory and associate with each proposition the published evidence that had appeared to that date. Interestingly some of the propositions had never been tested, while others had shown mixed evidence. Perhaps more surprising was the fact that at least one published paper that had made use of the theory as presented in an earlier technical report [31], did not make reference to the published theory. It is clear that in this sub-field of empirical software engineering there is a misdirected focus on the experiment rather than the theory that supports the experiment.

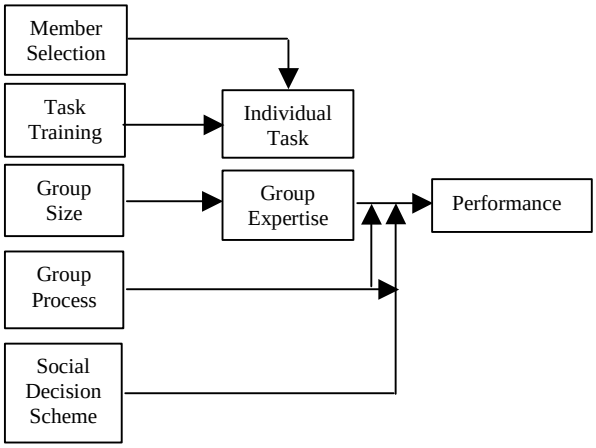

Figure 2. Behavioural Theory of Group Inspection Performance

4. Theory derivation and refinement

The history and philosophy of science is a well-developed field of enquiry and one that has been able to set out the agreed principles of scientific enquiry. Recently, Kitchenham et. al. [32] have published "Preliminary Guidelines for Empirical Research in Software Engineering", a first attempt to explicitly state research guidelines specifically for empirical software engineering research based on accepted methods from other scientific fields (most notably Medicine). Figure 3 presents our interpretation of a scientific approach to studying software engineering phenomena. The process begins when a scientist becomes aware of a phenomenon that is poorly understood. The phenomenon may be, for example, a practice that is occurring in industry (e.g. cost estimation) or a new proposal for a software engineering tool or method (e.g. perspective-based reading). The phenomenon has been invented (proposed) and may even be practiced in the real world but the understanding of it is characterised by hearsay, common beliefs and experience reports. Development and modelling of the phenomenon is characterised by the same unfounded belief system. Experience over hundreds of years of enquiry has shown that only a limited amount of development and improvement can be achieved in this context. XP is a good current example of development in this fashion.

The scientist is naturally uneasy that the phenomenon is not properly understood and at first seeks to understand more about the phenomenon. This can be done by using empirical methods including observational techniques such as case studies or by using exploratory experiments. Once the scientist understands more a tentative theory can be formulated. The theory is then open to testing and evaluation by the software engineering community.

Experiments that evaluate the theory by comparison, for instance, are common here (e.g. perspective-based reading vs ad hoc approaches). The result is a tested theory. Ideally, all experiments should be independently replicated to prove or disprove the results and, if necessary, theory revision should take place to improve the theory. Finally, the theory can be applied to improve the phenomenon.

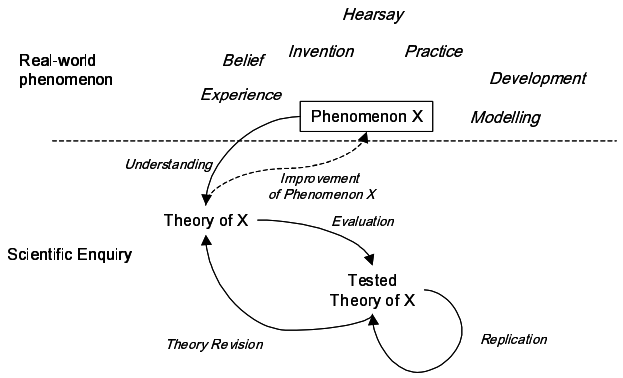

Figure 3: A model of scientific enquiry.

It is interesting to place the research in estimation and inspections into this model and to compare the approaches and outcomes of the research in both areas.

In the estimation research we can see a large number of studies that have been concerned with the process of understanding the phenomenon of software costs and sizing in order to build better theory of software costs. Most of the data studied has been industrial data and in the early years there were generally only correlation studies that sought to understand the relationships between effort, schedule, size and the various cost drivers. In later years we saw the studies that evaluated different types of estimation models or methods; for example analogy versus algorithmic studies or ordinary least squares regression versus robust regression, and so on. Another observation is that the understanding has been increasing. In the early studies to test the accuracy of the models the Magnitude of Relative Error (MRE) was as high as 700% but today we see combination algorithmic and knowledge-based methods returning an MRE of around 10-20%.

This improvement process resulted from the early publication of the COCOMO model that embodied the simple theory that development effort was a function of size and a series of cost drivers. Over the years of experimentation we have come to understand the importance of tailoring not only the value of the cost drivers, but also the selection of those cost drivers. Our understanding of early lifecycle size measures has increased and our ability to model the uncertainty and risks of a development project has improved. So too has our understanding of the modelling techniques that we might use to construct organisational-specific cost models.

This is a research area that has progressed very well and in line with the normal scientific process of:

1. Understanding,
2. Theory building
3. Theory testing
4. Replication
5. Theory revision, and
6. Possible re-evaluation of the phenomena.

Hopefully this process will continue as new technologies and development methods are developed.

The research in the area of inspections has not yet followed this path. Although theory may have been created through an understanding of the phenomenon, the empirical evaluation has not been focused on theory testing at all. Rather it has been concerned with evaluating new ideas, replicating the evaluation, but NOT revising the theory or feeding back into our understanding of the phenomenon. Perhaps this explains the current attempt to carry out a worldwide study of inspection practice. It appears that at present there is confusion in the empirical inspections literature. Replication experiments have been unable to verify earlier results. We propose that this is a result of insufficient expression of theory, a consequent lack of models, and too little attention in the experiments to the justification for the hypotheses under test.

It is also interesting to consider the wider body of empirical software engineering literature in the context of the model and see how it compares to the experiences above. Our source of data is a review of the content of the Journal of Empirical Software Engineering (EMSE) from 1996 - 2002 (sixty-eight full-length, peer-reviewed papers). We believe that, although empirical software engineering is published in other journals, this journal is a valid reflection of the general state of empirical software engineering.

Of the sixty-eight papers reviewed, twenty-eight are concerned with understanding software engineering phenomena, twenty with evaluating theories and seven with replicating evaluation experiments. Of the other fourteen papers, seven were concerned with empirical software engineering methods and seven we deemed not to be true empirical software engineering papers. This shows healthy activity in understanding and evaluating phenomena, but rather little work on replication and theory revision. It also shows a disappointing small amount of reflection and method development in empirical software engineering. This is especially worrying since this journal is typically the only forum for such development.

Of further concern is the number of areas (topics) of software engineering to which the empirical research is applied; for example, requirements engineering, software

quality or process improvement. Over the sixty-eight papers, thirteen different areas of software engineering were investigated. The most popular area was inspections with thirteen papers followed by metrics with eleven papers. This broad spread of topics makes it difficult to identify a body of knowledge in any one area. With the exception of inspections and metrics, no other area of software engineering even begins to address empirical investigation from understanding through to theory revision, theory acceptance and, finally, theory application.

Other observations also raise concern about the practice of empirical software engineering research. For example, many of the evaluation papers contain only implicit references to theory which, again, makes it difficult to identify a body of knowledge. Moreover, many evaluations are carried out by the inventor of the theory (indeed, often by the inventor of the phenomenon itself), which is scientifically fraught. Improvements could be made in empirical software engineering by better articulating well-formed theories and by providing independent evaluation of theories.

But the picture is not all bleak. The balance of investigations using industrial data compared to those using student data was very even (twenty-three industrial/twenty-four student). This suggests a healthy relationship between empirical researchers and software engineering industry, which should result in work that is relevant to industry.

Also encouraging is the wide range of empirical methods used, ranging from exploratory case studies through surveys, literature reviews and quasi-experiments all the way up to controlled laboratory experiments. The popularity of the case study method to investigate industrial phenomena (fourteen out of twenty-three papers) reflects the difficulty of carrying out controlled experiments in industrial settings. Surveys and quasi-experiments were also used with industrial data and two controlled experiments were carried out with professional software developers.

Where more rigorous experimental methods were used, students were the more common source of data. Ten of the twelve controlled experiments reported used student data and only two case studies were performed on students.

In an attempt to judge how healthy the academic community in empirical software engineering is, we looked at the citation rate for papers in the Journal of Empirical Software Engineering and compared them to what is widely regarded as the premier journal in software engineering (IEEE Transactions on Software Engineering (TSE)) and what we regard as a mid-range journal (Information and Software Technology (IST)). The source of citation data is the "Web of Science" citation index, which contains most of the popular journals in computing, software engineering and information systems.

Fifty percent of the papers published in EMSE from 1996 to 2002 were cited at least once in the following years. This compared with 66% for TSE and 38% for IST. The citation rate for all journals dropped off markedly after 1998 reflecting the lag time between publication of a paper and the publication of work that may reference the paper. In 1996 the citation rate for EMSE was 70% compared with 94% for TSE and 53% for IST. We note here that a third of the papers cited in EMSE were cited only by their own authors - however we do not have self-citation data to compare for TSE and I&ST. Our conclusion is that a publication in EMSE is not likely to be as influential as in TSE, but is as good if not better than a mid-quality journal like IST. These citation rates suggest a reasonably healthy interest in the papers published in EMSE.

Table 1: The percentage of papers cited at least once from EMSE, TSE and IST.

Year	% Cited		
	EMSE	TSE	IST
1996	70	94	53
1997	63	94	58
1998	77	88	44
1999	58	57	44
2000	20	43	21
2001	14	21	14

5. Conclusions

Experiences show that empirical software engineering can have an impact in developing sound and practical theories of software engineering phenomena (e.g. cost estimation). It also shows that many years of investigation may not result in a well-understood theory (e.g. inspections). One way of understanding this is within the framework of scientific enquiry. To develop a sound and practical theory a wide range of enquiry is necessary. Experience shows that concentrating the investigation on just one of understanding or evaluation and without independent evaluation, replication and theory revision it is unlikely that a useful theory will result. Unfortunately, most of the topics presented in the Empirical Software Engineering Journal do not show this depth of enquiry. Conspicuous by their absence are well-formed theories, independent evaluation, replication, theory revision and empirical software engineering method development.

6. Acknowledgments

We would like to thank Lucila Carvalho for helping to collect the data on papers from the Journal of Empirical Software Engineering.

7. References

[1] D. R. Jeffery and M. J. Lawrence, "An Inter-Organizational Comparison of Programming Productivity," *Proceedings of the 4th International Conference on Software Engineering*, pp. 369-377, 1979.

[2] D. R. Jeffery and M. J. Lawrence, "Managing Programming Productivity," *Journal of Systems and Software*, vol. 5, pp. 49-58, 1985.

[3] D. R. Jeffery, "Time-Sensitive Cost Models in the Commercial MIS Environment," *IEEE Transactions on Software Engineering*, vol. 13, pp. 852-859, 1987.

[4] D. R. Jeffery, "A Comparison of Programming Productivity - The Influence of Program Type and Language," *Proceedings of the ACC86*, pp. 382-400, 1986.

[5] D. R. Jeffery, "Software Development Models for a Fourth Generation Language Environment," presented at TIMS XXVII, Australia, 1986.

[6] G. C. Low and D. R. Jeffery, "The Productive Use of Current CASE Tools in Software Development," *Proceedings of the ACC89*, 1989.

[7] G. C. Low and D. R. Jeffery, "Productivity Issues in the Use of Current Back-end CASE Tools," *Proceedings of the 3rd International Workshop on CASE Conference*, vol. supplementary, pp. 12-38, 1989.

[8] D. R. Jeffery and G. C. Low, "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering*, vol. 16, pp. 64-71, 1990.

[9] D. R. Jeffery and G. C. Low, "A Comparison of Function Point Counting Techniques," *IEEE Transactions on Software Engineering*, vol. 19, pp. 529-532, 1993.

[10] J. Stathis and D. R. Jeffery, "Specification Based Software Sizing: An Empirical Investigation of function Metrics," *Proceedings of the 18th Annual Software Engineering Workshop*, pp. 97-115, 1993.

[11] J. Stathis and D. R. Jeffery, "An Empirical Study of Albrecht Function Points," *Proceedings of the ASMA Australian Conference on Software Metrics*, pp. 96-117, 1993.

[12] R. Lo, R. Webby, and D. R. Jeffery, "Sizing and Estimating the Coding and Unit Testing Effort for GUI Systems," *Proceedings of the 3rd International Software Metrics Symposium*, pp. 166-173, 1996.

[13] D. R. Jeffery and J. Stathis, "Function Point Sizing: Structure, Validity, and Applicability," *Empirical Software Engineering: An international Journal*, vol. 1, pp. 11-30, 1996.

[14] S. Cant, D. R. Jeffery, and B. Henderson-Sellers, "A Conceptual Model of Cognitive Complexity of Elements of the Programming Process," *Information &Software Technology*, vol. 37, pp. 351-362, 1995.

[15] F. Walkerden and D. R. Jeffery, "Software Cost Estimation: A Review of Models, Process, and Practice," *Advances in Computers*, pp. 59-125, 1997.

[16] D. R. Jeffery, M. Ruhe, and I. Wieczorek, "A Comparative Study of Two Software Development Cost Modelling Techniques Using Multi-Organizational and Company-Specific Data," *Information &Software Technology*, pp. 239-248, 2000.

[17] D. R. Jeffery, M. Ruhe, and I. Wieczorek, "Using Public Domain Metrics to Estimate Software Development Effort," *Proceedings of the 7th International Software Metrics Symposium*, pp. 16-27, 2001.

[18] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentics-Hall Inc., 1981.

[19] L. H. Putnam, "A general empirical solution to the Macro software sizing and estimating problem," *IEEE Transactions on Software Engineering*, vol. 4, pp. 345-360, 1978.

[20] L. C. Chai, "An Evaluation Study of the Function Point Measure." Unpublished B. Sc. Honours Thesis, Sydney, Australia: University of New South Wales, 1985, pp. 130.

[21] L. C. Briand, K. El Emam, and F. Bomarius, "COBRA: A Hybrid Method for Software Cost Estimation, Benchmarking and Risk Assessment," *Proceedings of the 20th International Conference on Software Engineering*, pp. 390-399, 1998.

[22] A. A. Porter, L. G. Votta, and V. R. Basili, "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," *IEEE Transactions on Software Engineering*, vol. 21, pp. 563-575, 1995.

[23] D. L. Parnas and D. M. Weiss, "Active Design Reviews: Principles and Practices," *Proceedings of the 8th International Conference on Software Engineering*, pp. 215-222, 1985.

[24] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz, "The Empirical Investigation of Perspective-Based Reading," *Empirical Software Engineering: An international Journal*, vol. 1, pp. 133-164, 1996.

[25] P. Fusaro, F. Lanubile, and G. Visaggio, "A Replicated Experiment to Assess Requirements Inspection Techniques," *Empirical Software Engineering: An international Journal*, vol. 2, pp. 39-57, 1997.

[26] K. Sandahl, O. Blomkvist, J. Karlsson, C. Krysander, M. Lindvall, and N. Ohlsson, "An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections," *Empirical Software Engineering: An international Journal*, vol. 3, pp. 327-354, 1998.

[27] P. M. Johnson and D. Tjahjono, "Does Every Inspection Really Need a Meeting?," *Empirical Software Engineering: An international Journal*, vol. 3, pp. 9-35, 1998.

[28] J. Miller, M. Wood, and M. Roper, "Further Experiences with Scenarios and Checklists," *Empirical Software Engineering: An international Journal*, vol. 3, pp. 37-64, 1998.

[29] A. A. Porter and L. G. Votta, "Comparing Detection Methods for Software Requirements Inspections: A replication Using Professional Subjects," *Empirical Software Engineering: An international Journal*, vol. 3, pp. 355-380, 1998.

[30] C. Sauer, D. R. Jeffery, L. Land, and P. Yetton, "The Effectiveness of Software Development Technical Reviews: A Behaviourally Motivated Program of Research," *IEEE Transactions on Software Engineering*, vol. 26, pp. 1-14, 2000.

[31] C. Sauer, D. R. Jeffery, L. Lau, and P. Yetton, "A Behaviourally Motivated Programme for Empirical Research into Software Development Reviews," CAESER, Sydney, Technical Report 96/5, 1995.

[32] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, pp. 721-734, 2002.