Empirical Paradigm – The Role of Experiments

Barbara Kitchenham

Abstract. This article discusses the role of formal experiments in empirical software engineering. I take the view that the role of experiments has been overemphasised. Laboratory experiments are not representative of industrial software engineering tasks, so do not provide us with a reliable assessment of the effect of our techniques and tools. I suggest we need to concentrate a larger proportion of our research effort on industrial quasi-experiments and case studies. Methodologies for these empirical methods are well-understood in the social science and would appear to be appropriate mechanisms for investigating many software engineering research questions. In addition, I believe we need to make the results of empirical software engineering more visible and relevant to practitioners. To influence practitioners I suggest that we need to produce evidence-based text books and evidence-based software engineering standards.

1 Introduction

In this paper, I discuss the role of formal experiments in empirical software engineering. I believe that we may have over-emphasised the role of formal experiments in empirical software engineering and as a result we have both failed to identify the limitations and risks inherent in software engineering experiments and given insufficient consideration to other empirical methods.

My basic assumption is that the goal of empirical software engineering is to influence the practice of software engineering. This implies that we need empirical methods that provide us with insights into how software engineering works in practice and how changes to the process can result in changes to the outcomes of the process.

In order to explain my concern about formal experiments, I will identify some areas where the nature of software engineering practice is at odds with the requirements of formal experiments and discuss some of the risks that arise because of this. Then I will suggest that quasi-experimental design and case studies might be better suited to some types of empirical study than formal experiments. Finally, I will indicate how we might make the results of empirical studies more visible to practitioners.

2 Software Engineering Practice and Experimental Methodology

Software engineering in practice involves coordinating and integrating many different tasks (analysis, design, coding, testing, quality assurance, project management etc.) that rely heavily on human expertise often in the context of developing innovative products using new technologies. For large scale software engineering, this basic complexity is compounded by the involvement of many different engineers and

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 25 – 32, 2007.

managers working in cooperating teams (sometimes distributed) within one or more industrial cultures. In general it is difficult to identify one task or a single decision and consider its impact in total isolation from its surrounding context.

In comparison, formal experiments abstract tasks away from industrial contexts in order to study in detail specific isolated elements of a process, an event or an artefact. In general the more isolated the object of study is from its environment the easier it is to manipulate and study, but there is a risk that the results will not apply in more complex industrial situations.

Software engineering researchers often debate the use of student subjects, but in my opinion the choice of subjects is far less critical that the selection of materials, tasks and contexts. If our materials are small scale documents with known solutions, our tasks are restricted to those that take less than 2 hours, and the rich industrial context in which software tasks are planned and performed is removed, what is the value of the outcomes of our formal experiments? Clearly there are some cases when we can rely on formal experiments but there are significant risks. We need to consider more that just the scale-up problem, or the student subject problem, for example:

- We may fail to recognise the value of techniques that are not cost effective for small scale tasks but would be valuable for large scale activities (techniques that increase overheads such as documentation, project management or quality assurance would fit this category).
- We may not be able to define realistic *control* situations leading to experimental results that cannot be interpreted by practitioners (e.g. comparing task results based on training people with a new technique with results obtained from people given no training is poor experimental practice; new techniques are best compared with current best practice).
- We may over estimate the impact of our techniques when they are used in controlled situations without the variety inherent in industry practice. This may lead to over-optimistic ROI estimates.
- We may find ourselves examining phenomena that are a result of abstracting the technology away from its usage context not characteristics of the technology itself.
- We may blame practitioners for failure to use our methods when the real problem is our failure to understand the complexity of the context in which our techniques will be used.

If we look at what happens in other human intensive disciplines, we observe that either they are able to perform realistic experiments such as randomised controlled trials in medicine or they use quasi-experimental methods such as those developed by social scientists and educationalists.

The critical property of a randomised controlled trial is that it is a real trial of a treatment (e.g. a new drug or other health care intervention) in a real hospital (or health centre) involving real patients and real doctors, with outcomes that directly affect the health and well-being of the participants. It is extremely rare that we are able to undertake trials of such direct relevance to practitioners in software engineering. In fact I am aware of only one experiment (undertaken by Jørgensen and Carelius [1]) that incorporated a genuine randomised experiment within actual practice. Thus, I do not see software engineering being able to adopt randomised controlled trials as a standard experimental protocol.

It might be argued that we are better served by considering our laboratory experiments to be *exploratory* studies. However, formal experiments were designed with hypothesis testing in mind. It is not clear that they are as well suited to exploratory studies as other empirical methods such as industrial case studies.

Adding a qualitative element to formal experiments does not overcome the objection that they were designed for hypothesis testing, certainly not in the context of laboratory experiments with student subjects. Petticrew and Roberts [2] suggest qualitative research is more appropriate that randomised controlled trials for purposes of *salience* (whether the technology/service matters), *process of service delivery*, *acceptability* (whether the technology/service will be taken up by potential users), *appropriateness* (whether the technology/service is right for the proposed users, *satisfaction* (whether users are satisfied with the technology or service). However, to investigate these issues, researchers would need to obtain the opinion of potential users in a realistic context not surrogate users such as students trying out a small scale task in a laboratory.

I conclude that we should be more ready to perform industrial studies using quasi-experimental designs to support hypothesis testing (or *confirmation*) and qualitative studies (particularly case studies) to support hypothesis generation (or *exploration*). I discuss these approaches in more detail in the next section.

3 Quasi-experiments and Case Studies

The social sciences have developed a large number of quasi-experimental designs for large-scale field experiments, and have a clear understanding of the strengths and weaknesses of these designs. Quasi-experimental designs are designs in which it is impossible to allocate subjects/participants to treatment conditions at random. I suggest that empirical researchers in software engineering need to become more familiar with these types of designs and more open to the opportunities they offer to improve the rigour of large-scale industrial studies.

Quasi-experimental designs began with simple before and after designs which immediately confound treatment effects with the passage of time, but have evolved into far more robust designs. Shadish et al. [3] provide a catalogue of basic quasi-experimental designs incorporating multiple pre- and post-measures and control groups. They also describe designs such as interrupted time-series analysis and regression discontinuity that are almost as rigorous as formal experiments, but have the ability to monitor the impact of large-scale social interventions.

The rigour of quasi-designs has improved as researchers have continued to criticize and improve them. A major element of the criticism will be familiar to most empirical software engineers since it is based on an assessment of study validity. Indeed the validity terms that we use in software engineering have been obtained directly from validity issues associated with quasi-experiments undertaken in education and social policy (not formal experiments). For example, *maturity* validity is particularly important in studies that deal with children since the impact of various social and education programs will be confounded with the children's basic skills increasing as they grow older. Similar a *history* threat is a major problem if families living in poverty that are eligible for one support program (e.g. housing support) may also be

receiving another (e.g. food stamps). However, the studies of validity threats do not end with generic threats applicable to any design but have been refined to identify validity threats specific to particular types of quasi-design. For example, Shaddish et al. [3] provide a detailed list of validity threats for case control studies, and discuss validity issues for other quasi-designs.

3.1 Case Control and Cohort Studies

As examples of fairly common quasi-experimental design consider *case control* studies and *cohort* studies. In *case control* studies, we identify experimental units (e.g. humans, organizations, artifacts) that exhibit some undesirable characteristic (e.g. a project that significantly overruns its budget and timescale). We then match one or more *control*s with each case. The controls are units that do not exhibit the undesirable property but in all other respects match one of the cases. Differences between each case and its control(s) are investigated to look for possible reasons for the undesirable characteristic.

Retrospective case control studies are the standard design used to identify risk factors associated with medical conditions. They would seem an obvious candidate for determining project risk factors. This design has many limitations (see Shaddish et al. [3] Table 4.3 for a complete list). A major problem with such designs is to find the correct characteristics to match the cases and the control. Another problem is that case-control studies are usually backward looking (*retrospective*) studies. Other problems associated with data collection include:

- Underlying cause bias: Project managers of failing projects may reflect about possible causes and thus exhibit different recall than project managers of controls.
- Expectation bias: Observers may systematically err in measuring and recording data so that they concur with prior expectations.
- Exposure suspicion bias: Knowledge of the status (i.e. case or control) may influence the intensity and outcome of a search for exposure to a risk factor.
- Recall bias: Questions about specific exposures may be asked several times of cases and only once of control.

One approach to reducing bias resulting from questioning people about past events is to ensure that interviewers are kept "blind" to case status (i.e. the interviewers who interrogate project staff should not know whether the project was a failure or a success). Although this sounds strange, it is the standard practice for studies that interrogate people about their exposure to medical risk factors.

An alternative design is a forward looking (*prospective*) study. Cohort studies are often prospective. In this type of study we identify a sample of experimental units and observe their progress over time. Medical cohort studies involve millions of subjects over long periods of time (up to 20 years) so this type of design is suitable for large-scale, long-term studies. They are often used to identify the incident rate of diseases in the general population, so they would seem to be appropriate for issues such as the rate of project failures. However, there have been no prospective studies of this type performed in software engineering.

3.2 Evaluating Technology Impact

Two recent studies of the impact of ISO/IEC 15504 (SPICE) have been based on correlation studies ([4], [5]). Correlation studies are observational studies which are weaker methodologically than experiments or quasi-experiments. They always suffer from the problem that they cannot confirm causality. Significant correlations may occur by chance (particularly when a large number of variables are measured), or as a result of a "latent" variable (i.e. an unmeasured variable that affects two measured variables and gives rise to an apparent correlation between the measured variables).

A more reliable approach is to monitor the impact of technology adoption in individual organizations by measuring project achievements before and after adoption utilizing multiple measurement points before and after technology changes. This approach has been adopted by several researchers for CMM evaluations. For example, Dion [6] recorded cost of quality and productivity data for 18 projects, undertaken during a five year process improvement activity. The first two projects were started before the process changes were introduced; the subsequent projects were started as the series of process changes were introduced. Simple plots of the results show an ongoing improvement over time consistent with an ongoing process improvement exercise. However, the provision of data on projects started prior to the process changes gives additional confidence that the effect was due to the process change rather than other factors. Steen [7] provides another endorsement of Dion's methodology. He reviewed 71 experience reports of CMM-based SPI and identified Dion's study as the only believable report of Return on Investment (ROI) of CMM-based SPI.

In another study, McGarry et al. [8] plotted project outcomes before and after the introduction of CMM level 2. The data spanned a 14-year period and included 89 projects. The graphs showed that improvements in productivity and defect rates were not due to the introduction of CMM. The same improvement rate had been observed prior to the introduction of CMM and could be attributed to the general process improvement activities taking place before and during CMM adoption not specifically the adoption of CMM (i.e. McGarry observed a *history* effect). In contrast, improvements in estimating accuracy did appear to be a result of adopting CMM.

3.3 Industrial Case Studies

Quasi-experimental designs allow us to perform quantitative studies investigating factors such as the effectiveness of techniques, or the relative importance of project risk factors. Industrial case studies in contrast allow us to look in detail at the *how* and *why* of software engineering phenomena [9].

It is important to identify what I mean by a *case study*. A case study should be a genuine industrial software engineering project (or project activity), not a toy project, nor a special project performed for the purpose of evaluating a technology or training new staff. All too often researchers use the term case study when they mean *example* (i.e. recreating a previously constructed software artefact using a new technology). In principle, an industrial software project should act as a *host* for a case study. In fact, as Yin points out the most convincing case studies are those that have a strong rationale for case selection. This means that the host project should have

characteristics that make it suitable to address the issues being investigated by the case study. If we are concerned about investigating the way technology works in practice and its impact on practitioners, industrial case studies are likely to be a more reliable methodology than small-scale experiments with an added qualitative element.

4 Visibility of Empirical Software Engineering Results

Several recent publications have made the point that software engineering academics and practitioners trust expert opinion more than objective evidence ([10], [11]). I conclude that that empirical software engineering will not have much relevance to practitioners, if empirical studies have no visibility. For this reason, we need to find a suitable outlet for our results. There are two areas that empirical software engineering should address to make empirical ideas visible to practitioners: text books, which can influence software engineers during their training, and international standards, which are likely to impact industrial practitioners.

We need software engineering text books that incorporate empirical studies to support their discussion of technologies that identify the extent to which technologies have been validated, or under what conditions one technology might be more appropriate than another. Endres and Rombach [12] have made a start at this type of text book, but we need more general software engineering text books that include empirical evidence. Furthermore, text books require summarised evidence not simply references to individual empirical studies, so I we need more systematic literature reviews to provide rigorous summaries of empirical studies ([13], [2]).

We also need evidence-based standards. In my experience the quality of international software engineering standards is woeful. I have no objection to standards related to arbitrary decisions, such as the syntax of a programming language, which are simply a matter of agreement. However, standards that purport to specify best practice are another issue. Software standards of this type often make unsupported claims. For example ISO/IEC 2500 [14] says:

> "The purpose of the SQuaRE set of International Standards is to assist developing and acquiring software products with the specification and evaluation of their products. It establishes criteria for the specification of software product quality requirements, their measurement and evaluation."

I imagine a large number of researchers and practitioners would be surprised to find that the means of specifying, measuring, and evaluating software quality is so well-understood that it can be published in International Standard.

Compare this "International Standard" with the more modestly named "Research-based web design and usability guidelines" [15]. The web-guidelines not only explicitly reference the scientific evidence that supports them; they also define the process by which the guidelines were constructed. Each individual guideline is rated with respect to its importance and the strength of evidence supporting it. The software engineering industry deserves guidelines and standards of the same quality.

5 Conclusions

In this article, I have argued that we have overemphasised the role of formal experiments in empirical software engineering. That is not to say that there is no place for formal experiments in software engineering. Formal experiments can be used for initial studies of technologies such as proof of concept studies. There are also undoubtedly occasions when formal experiments are the most appropriate methodology to study a software engineering phenomenon (for example, performance studies of alternative coding algorithms). However, the nature of industrial software engineering does not match well with the restrictions imposed by formal experiments. We cannot usually perform randomised controlled trials in industrial situations and without randomised controlled trials we cannot assess the actual impact of competing technologies, nor can we assess the context factors that influence outcomes in industrial situations.

To address the limitations of formal experiments, I suggest that empirical software engineering needs to place more emphasis on industrial field studies including case studies and quasi-experiments. In addition, we need to make empirical results more visible to software engineers. I recommend that the empirical software engineering community produce evidence-based text books and campaign for evidence-based standards. Evidence-based text books would prepare future software engineers to expect techniques to be supported by evidence. Evidence-based standards might help practitioners address their day-to-day engineering activities and lead to a culture in which evidence is seen to benefit engineering practice.

Acknowledgements

National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. At Keele University, Barbara Kitchenham works on the UK EPSRC EBSE project (EP/C51839X/1).

References

1. Jørgensen, M. and G. J. Carelius. An Empirical Study of Software Project Bidding, IEEE Transactions of Software Engineering 30(12):953--969, 2004.
2. Petticrew, Mark and Roberts, Helen. Systematic Reviews in the Social Sciences. A practical Guide. Blackwell Publishing, 2006.
3. Shaddish, W.R., Cook, T.D., and Campbell, D.T. Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin Company, 2002.
4. El Emam, K. and Birk, A. Validating ISO/IEC 15504 measures of software development process capability. The Journal of Systems and Software, 51, 2000a, pp 119-149.
5. El Emam, K. and Birk, A. Validating ISO/IEC 15504 measures of software Requirements Analysis Process Capability, IEEE Transactions on Software Engineering, 26(6), 2000b, pp 541-566.
6. Dion, Raymond. Process Improvement and the Corporate Balance Sheet. IEEE Software, 10(4), 1993, pp 28-35.

7. Steen, H.U. Reporting framework-based software process improvement. A quantitative and qualitative review of 71 experience reports of CMM-based SPI. Master Thesis, Simula Research Laboratory & Department of Informatics University of Oslo, 29th October 2004.
8. McGarry, F., Burke, S. and Decker, B. Measuring the Impacts Individual process Maturity Attributes have on Software Products. Proceedings Fifth International Software Metrics Symposium, IEEE Computer Society, 1998, pp 52-60.
9. Yin, Robert K. Case Design and Methods. Third edition, Sage Publications, 2004.
10. Barbara Kitchenham, David Budgen, Pearl Breton, Mark Turner, Stuart Charters and Stephen Linkman, Large Scale Software Engineering Questions – Expert Opinion or Empirical Evidence? Experience and Methods from Integrating Evidence-based Software Engineering into Education, Proceedings 4th International Workshop WSESE2003, Fraunhofer IESE-Report No., 068.06/E, 2006
11. Rainer, A., Jagielska, D. and Hall, T. Software Practice versus evidence-based software engineering research. In Proceedings of the Workshop on Realising Evidence-based Software Engineering, ICSE-2005, 2005, http:/cfm.portal.acm.org/dl.
12. Endres, Albert and H. Dieter Rombach. Empirical Software and Systems Engineering: A Handbook of Observations, Laws and Theories, Addison Wesley, 2003.
13. Kitchenham B. (2004). Procedures for Undertaking Systematic Reviews, Joint Technical Report, Computer Science Department, Keele University (TR/SE-0401) and National ICT Australia Ltd (0400011T.1)
14. ISO/IEC 25000. International Standard. Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. 2005-08-01.
15. Koyani, S.J., Bailey, R.W. and Nall, J.R. (2003) Research based web design and usability guidelines. National Cancer Institute, Available for download at http://usability.gov/ pdfs/guidelines.html.