Research Methods for the Empirical Assessment of Software Processes

Joost Schalken

Vrije Universiteit, Department of Computer Science,
De Boelelaan 1083a, 1081 HV Amsterdam, The Netherlands
`jjp.schalken@few.vu.nl`

Abstract. For the state-of-practice in software engineering to improve, industry needs to apply methods and tools from software engineering research. To enable industrial software engineers to select which results are useful for them, research institutions need to provide sound evidence about the effectiveness of proposed methods and tools in practice. Research methodology for empirical software engineering has not fully matured yet and no consensus exists on what is acceptable as proof in empirical software engineering. This paper describes what problems exists with current research approaches and proposes a research plan to improve insights in what standards need to be met by research methods.

1 Introduction

Software engineering can still not be seen as a mature profession [1]. Still almost a quarter of all software projects fail [2] and progress to improve the state of practice is slow. There is a need for better tools and methods to develop software systems. However, industry is slow to pick up improvements suggested by academia. Redwine and Riddle show in their study that on average acceptance of software technology by industry takes 10 to 15 years [3].

In their model of software technology maturation Redwine and Riddle state that substantial evidence on the value and applicability of a technology is needed before it will be accepted by industry [3]. In the software engineering community there is a growing awareness for the need of empirical validation of the effectiveness of proposed methods and tools. Without proper empirical validation one is unable to assess the effectiveness of the the proposed tools and methods in a real-life setting. The problem in the software engineering discipline is that there is no accepted standard for what constitute suitable methods to gather the required evidence [4] to convince both academic software engineering researchers and software engineers in practice. Although practitioners rarely directly question the evidence supporting a method or technique, the continuing religious wars regarding tools and methods could well be a consequence of the lack of solid evidence on the effectiveness of the proposed methods.

Historically the emphasis of software engineering research has been on the construction of new tools and the invention of new development methods, not on the empirical evaluation of methods and tools in complex, industrial settings. The social sciences have long struggled to find acceptable methods to perform research in complex settings and have developed a substantial body of knowledge regarding the methods of scientific enquiry. However also in the empirically well-established disciplines such as social sciences there is controversy over what methods of scientific enquiry are acceptable. On top of that it is not always straightforward to apply research methods of the social sciences on software engineering research (see for example [5, 6]).

Barbara Kitchenham et al. have suggested that the application of the evidence-based practice paradigm to software engineering offers an opportunity to improve the quality of current empirical research and to improve the diffusion of research results to practitioners [7]. Evidence-based software engineering has been inspired by evidence-based medicine, which has had substantial success in achieving the goal of transferring research results to practice in the medical domain.

Unfortunately there are some serious problems related to the application of the evidence-based practice paradigm to software engineering. One of the most significant problems has to do with the inability to consistently use randomized, double-blind experiments within empirical software engineering research. These problems and other problems with research methodology are explained in more detail in Sect. 3.2.

As no silver bullet has (yet) been found to solve all research methodological difficulties for software engineering research, the researcher has to choose a research approach that is appropriate for his or her problem at hand. The research approach has to include at least the research strategy (such as experiment, survey, archival analysis, history or case study [8, p. 5]), the research design (which experimental subjects are selected and how are treatments assigned to the subjects), data collection procedures (what to measure and how to measure it) and analysis methods (such as regression analysis, ANOVA or grounded theory [9]).

The research approach has to be adapted to the properties of the research problem at hand: the reason for performing the study and the context in which the research will be performed. The reason for performing the study, also called the research motive, has on impact on the research question. If a research project is undertaken as a critique on development practices the research question will most likely differ from a research project that is motivated to improve the state of development practice. Different research motives will result in different research questions for the same problem domain. These different research questions will in turn lead to different research approaches to investigate similar phenomena. Furthermore, properties of the context in which the research project will be performed will have a profound impact on the research approach taken. Performing long case studies with students in an academic setting might prove to be difficult, whereas in industry experiments might be harder to perform. Therefore the research approach must be suitable for the context of the study.

The research approach chosen by the researcher is not only influenced by the properties of the research problem, but also by the researcher's notion of knowledge. "Epistemological assumptions decide what is to count as acceptable truth by specifying the criteria and process of assessing truth claims. ... Methodological assumptions indicate the research methods deemed appropriate for the gathering of valid evidence." [10].

The goal of my Ph.D. project is to develop a framework to describe how epistemological assumptions, methodological assumptions and properties of the research problem at hand guide the selection of a research approach. This framework will be used as a starting point to perform a content analysis of published journal articles to determine what research approaches are current and what assumptions guide the selection of these research approaches. If software engineering researchers have inter-subjective agreements regarding the selection of research approaches, this consensus can be translated into methodological guidelines for software engineering researchers.

In the CAiSE Doctoral Consortium I would like to discuss the components of the analysis framework to study the choice of a researcher for certain research approach. In addition to feedback on the framework, I would like to discuss other potential strategies to analyze the strenghts and weakness of research approaches to perform empirical software engineering research.

The remainder of this paper is structured as follows; Section 2 describes some related work in the field of research methodology in software engineering. Section 3 describes the research I have performed so far and discusses the recent change of direction in my Ph.D. project from assessing the effects of software process improvement to investigating research methods for empirical software engineering research. In Sect. 4 the goals of my research project are explained. Section 5 discusses how I plan to proceed with my research and wraps up the paper.

2 Related Work

Remarkably little related work exists in the field of research methodology for software engineering, the study of the research methods that are used by software engineering researchers. Only recently Shaw observed that the software engineering discipline has no accepted standard for what constitute suitable methods to gather the required evidence [4]. Although much has been written about individual research methods (e.g. how to perform an ANOVA test or how to design a randomized experiment) little research has been performed to study how a researcher can choose the best research approach for the research problem at hand.

Adrion started the analysis of software engineering research methods by classifying software engineering research in four distinctive categories [11]: the scientific method, the engineering method, the empirical method and the analytical method.

Kitchenham provides some insights in teh strengths and weaknesses of distinctive research approaches used in software engineering research [12]. She however does not provide normative guidance on choosing a research approach for a given research problem nor does she compare the indicated use of a research method with its actual use by researchers.

Zelkowitz and Wallace [13] and Tichy, Lukowitz, Prechelt and Heinz [14] take a different approach; instead of focusing on which research approach would be most appropriate for a certain research problem, they focus on the actual use of a research approach by computer scientists in practice and place less emphasis on what research method is best. By performing content analysis on published papers they are able to show which research approaches are used in successful research.

Shaw takes this approach a little further by comparing the research approaches from papers that have been accepted for a prestigious conference with the research approaches from papers that have been rejected [4]. Is this manner she sheds some light on what are research approaches that are acceptable in the eyes of other researchers and what research approaches are clearly not acceptable.

Unfortunately no study has been found by the author that takes the full spectrum of questions into account regarding the choice of research approaches for software engineering research.

3 From Software Process Improvement to Research Methodology

This section describes why the focus of my Ph.D. project has recently changed from assessing the effects of software process improvement to research methodology for empirical software engineering.

3.1 Assessing Software Process Improvement

Two years ago my Ph.D. project started with the goal to assess and quantify the effects of model-based software process improvement. Under the supervision of prof. dr. J.C. van Vliet (Vrije Universiteit, Amsterdam) and prof. dr. S. Brinkkemper (Utrecht University) a study of a large software process improvement program was performed at the internal Information Technology department of a large Dutch financial institution (ABN AMRO Bank N.V.).

In this IT department over 1500 people are employed. The organization primarily builds and maintains large, custom-built, mainframe transaction processing systems, most of which are built in COBOL and TELON (an application-generator for COBOL). Besides these mainframe systems, a large variety of other systems are implemented, constructed and maintained by the organization. These systems are implemented in a large variety of different programming languages (such as Java and COOL:Gen), run under various operating systems (such as Microsoft Windows and UNIX) and are distributed over different platforms (batch, block-based, GUI-based and browser-based).

The organization has recently finished a major, four-year software process improvement program (SPI) to improve the internal IT processes and cooperation with the business. The SPI program included the introduction of the Dynamic Systems Development Method (DSDM) [15], a quality system that complies with the requirements of Software CMM [16] level 2, the introduction of a software metrics program, and a culture change program.

The empirical study of the SPI program has been conducted at two levels of abstraction: holistic analyses of the overall effects and in-depth analysis of individual processes. In the overall study we examined the specific productivity gains in different organization units, after the successful implementation of the Software CMM-compliant quality system [17,18], using the the single, embedded case study design [8,19] as the guiding research strategy. The results showed that the overall productivity of the IT organization consistently increased after the implementation of the improvement steps.

To gain more insight into the effects of SPI on the requirements gathering process, we compared the effectiveness of projects using the existing requirements gathering process using one-to-one interviews with projects that used facilitated workshops to gather requirements [20]. Facilitated workshops, which are a technique of the DSDM method [15], are intensive meetings in which technical staff, end-users and management collaborate on information systems development tasks, such as project planning, requirements specification and user interface design. Facilitated workshops fit in the general tendency to increase the involvement of stakeholders in the requirements engineering process. In the study the duration, effort and satisfaction of 49 DSDM projects using facilitated workshops have been compared with 20 projects that used the previous method, which use one-to-one interviews to gather the requirements. For small projects, one-to-one interviews were found to be more efficient, whereas for larger projects the efficiency of one-top-one interviews is surpassed by DSDM's facilitated workshops. Other quantitative effects were not found, and subjective ratings of stakeholders do not indicate a preference for DSDM projects either.

In another study we examined the organization's project post-mortem review database to find useful lessons learned that could provide opportunities for bottom-up software process improvement [21]. As these project post-mortem reviews were mostly recorded in plain text, is was difficult to derive useful overall findings using conventional analysis methods. For this study we developed a five-step method to transform the qualitative, natural language type information present in those reports into quantitative information, using a grounded theory [9] approach. This quantitative information can subsequently be analyzed statistically and related to other types of quantitative project-specific information. Currently, in a subsequent study, the same method is being applied to a different set of post-mortem reviews, in cooperation with researchers not involved in the development of the technique, to see how generally applicable the method is.

We have also designed a metric that can be used to measure the size of projects that install and configure COTS stand-alone software, firmware and hardware components [22], in order to assess the efficiency of the departments performing these functions. We call these components IT infrastructure, as these components often form the foundation of the information system that is built on top of it. Before the study started no accepted size metric existed for the installation and configuration of stand-alone software, firmware and hardware components. The proposed metric promises to be a viable instrument to assess the effectiveness and efficiency of IT infrastructure projects, yet requires further empirical study, which is underway.

3.2 Problems Assessing Software Process Improvement

During the design of the holistic studies of the efficiency benefits of SPI and the design of study of the benefits of facilitated workshops questions arose about the relative merits of different research strategies (such as experiment, survey, archival analysis, history or case study [8, p. 5]).

When choosing research methods, it seems as if one always needs to make a trade-off between relevance and rigor [23]. Interpretative case studies [24] and action research [23] and other interpretative approaches can give deep, relevant insights into what happens during a software engineering project. These research strategies are however often criticized for lacking sufficient scientific rigor (even with the guiding principles described by Klein & Meyers [24] and Davidson et al. [23]).

Experiments (and to a lesser degree quasi-experiments) on the other hand excel in scientific rigor, but the results of experiments with objective measurements often merely prove what is already known to be true by practitioners in the field. And when surprising results do appear, software engineering experiments often do not provide sufficient information to validate or explain all the results [25].

At the same time choosing research methods is also a trade-off between internal and external validity [26]. When using interpretive research, one can be rather confident that the obtained data really is related to the intended constructs (construct validity) and the deeper insights gained through interpretative research will usually make clear to the researcher if the theory will be applicable to other organizations. This advantage should be weighed against the disadvantages of subjectivism (other researchers can have a different interpretation of the same events and perhaps even the same data) and the risk of inferring causal relations where in reality none exist; the human mind is not good at distinguishing causal relationships and random correlations, one always runs the risk that the observed phenomenon could not be explained by chance alone.

Experiments and quasi-experiments with objective measures are however also not without problems. The measurement definitions can be contradictory and often be explained in different wyas (e.g., a book over 100 pages is needed to define a standard for consistently counting lines of code), leading to differences in observed data, just because the measurement procedures differed. This is even more problematic when comparing data between companies (such as in meta-analysis or benchmarks) as differences in measurement procedures will be even larger.

Another severe problem with experiments and quasi-experiments is that one cannot always be sure that the right operationalization of the variables is chosen. E.g., is developed function points per hour a good measure of productivity or should quality also be taken into account?

A procedure to improve the validity of experiments is to control as many environmental factors as possible and to use random double-blind assignment (in which both researcher and subject do not know to which treatment group they are assigned). Although this method works well for evidence-based practice in certain medical disciplines, software engineering is a skill-based profession in which the professionals will have to carry out the assignment. This makes double blinding impossible, the software engineer will always know what method he is practicing (which could influence the results).

The skill-based aspect also makes comparison between treatments more complicated; a pill can only be applied in a single manner, whereas the application of a skill can differ vastly between individuals [27]. And with complex, composite methods (such as Extreme Programming) not all organizations will employ the full method [28].

3.3 Shift to Research Methodology

The lack of concensus on what constitutes proper proof and the trade-off between relevance and rigor makes it hard to choose an appropriate research strategy. In my own Ph.D. research on software process improvement I have been confronted with these difficulties many times. This has led me to shift my focus from performing SPI research to performing research on the research methodology for empirical software engineering. I will continue to investigate the effects of SPI, but now these research projects will serve as testcases to test new research strategies instead of investigating SPI as an final goal.

4 Problem Context and Research Goals

The problem context of my research has been explained in the previous paragraphs. There is a dire need for clarification and guidance on for the selection of research strategies. It will however not be possible to design value-free criteria to allow a rational choice of research strategies, as the selection of a research strategy not only depends on the problem at hand, but also on the epistemological and methodological assumptions of the researchers.

In my Ph.D. project I will therefore aim to:

1. Define a framework to describe the relevant epistemological assumptions, methodological assumptions and properties of the research problem at hand that guide research strategy selection.
2. Identify which research strategies are used for which kinds of research. In essence this is establishing the links between epistemological assumptions, methodological assumptions, relevant properties of the research problem and the chosen research strategy.

3. Provide criteria that can help in selecting an appropriate research strategy, based on own research experiences and reflection on other research.

5 Further work

In the remaining two years of my Ph.D. project I plan to:

1. Study literature on epistemological philosophy to identify the different positions on epistemology and their links to research strategy.
2. Perform a thematic content analysis on current articles that have appeared in software engineering research, to investigate the state-of-practice in software engineering research and identify links between researcher assumptions, problem characteristics and chosen research methodology.
3. Critically compare articles that have the same research problem but have used different research methods, to see advantages and disadvantages of the various approaches.
4. Continue with empirical software engineering research, to have a body of personally performed research to reflect upon.

Acknowledgements

The author wishes to thank the two anonymous reviewers of this paper for their stimulating critique and insightful remarks.

References

1. McConnel, S.: After the Gold Rush. Microsoft Press, Redmond, WA, USA (1999)
2. The Standish Group West Yarmouth, MA, USA: Extreme Report. (2001) Available from: http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf.
3. Redwine, Jr., S.T., Riddle, W.E.: Software technology maturation. In: Proceedings of the 8th International Conference on Software Engineering (ICSE-85), Washington, DC, USA, IEEE Computer Society Press (1985) 189–200
4. Shaw, M.: Writing good software engineering research papers. In: Proceedings of the 25th International Conference on Software Engineering (ICSE-03), Washington, DC, USA, IEEE Computer Society Press (2003) 726–736
5. Miller, J.: Statistical significance testing–a panacea for software technology experiments? Journal of Systems and Software **73** (2004) 183–192
6. Jørgenson, M., Sjøberg, D.: Generalization and theory-building in software engineering research. In: Proceedings of the 8th Conference on Evaluation & Assessment in Software Engineering (EASE 2004), Stevenage, UK, IEE Press (2004) 29–45
7. Kitchenham, B.A., Dyba, T., Jørgensen, M.: Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering (ICSE-04), Washington, DC, USA, IEEE Computer Society (2004) 273–281
8. Yin, R.K.: Case Study Research: Design and Methods. 3nd edn. Volume 5 of Applied Social Research Methods Series. Sage Publications, Inc., Thousand Oaks, CA, USA (2003)

9. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Observations. Weidenfeld and Nicolson (1967)
10. Chua, W.F.: Radical developments in accounting thought. The Accounting Review **61** (1986) 601–632
11. Adrion, W.R.: Research methodology in software engineering: Summary of the dagstuhl workshop on future directions in software engineering. SIGSoft Software Engineering Notes **18** (1993) 36–37
12. Kitchenham, B.A.: Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods. SIGSOFT Software Engineering Notes **21** (1996) 11–14
13. Zelkowitz, M.V., Wallace, D.R.: Experimental models for validating technology. Computer **31** (1998) 23–31
14. Tichy, W.F., Lukowicz, P., Prechelt, L., Heinz, E.A.: Experimental evaluation in computer science: a quantitative study. Journal of Systems and Software **28** (1995) 9–18
15. Stapleton, J.: Framework for Business Centred Development: DSDM Manual version 4.1. DSDM Consortium, Ltd., Kent, United Kingdom. (2002)
16. Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V.: Capability maturity model for software, version 1.1. Technical Report CMU/SEI-93-TR-24, DTIC ADA263403, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA (1993) Available from: http://www.sei.cmu.edu/.
17. ABN AMRO Bank N.V. Amsterdam, NL: Process Benefits: Bridging the Process gap. (2002)
18. Schalken, J.J.P.: Trendanalyse productiviteitsontwikkeling: Effecten invoering CMM binnen Payments domein op projectproductiviteit ICT Services. ABN AMRO Bank N.V., Amsterdam, NL. (2003)
19. Scholz, R.W., Olaf, T.: Embedded Case Study Methods: Integrating Quantitative and Qualitative Knowledge. Sage Publication, Inc., Thousand Oaks, CA, USA (2002)
20. Schalken, J.J.P., Brinkkemper, S., van Vliet, J.C.: Assessing the effects of facilitated: Workshops in requirements engineering. In: Proceedings of the 8th Conference on Evaluation & Assessment in Software Engineering (EASE 2004), Stevenage, UK, IEE Press (2004) 135–144
21. Schalken, J.J.P., Brinkkemper, S., van Vliet, J.C.: Discovering the relation between project-factors and project success in post-mortem evaluations. In: Proceedings of the 11th European Software Process Improvement Conference (EuroSPI 2004). Volume 3281 of Lecture Notes in Computer Science., Berlin, D, Springer-Verlag (2004)
22. Schalken, J.J.P., Brinkkemper, S., van Vliet, J.C.: Measuring IT infrastructure project size: Infrastructure Effort Points. In: Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05). Lecture Notes in Computer Science, Berlin, D, Springer-Verlag (2005) Accepted for CAiSE'05.
23. Davison, R.M., Martinsons, M.G., Kock, N.: Principles of canonical action research. Information Systems Journal **14** (2004) 65–86
24. Klein, H.K., Myers, M.D.: A set of principles for conducting and evaluating interpretive field studies in information systems. MIS Quarterly **23** (1999) 67–93
25. Karahasanovif, A., Anda, B., Arisholm, E., Hoove, S.E., Jørgensen, M., Søxberg, D.I.K., Welland, R.: Collecting feedback during software engineering experiments. Empirical Software Engineering **10** (2005) 113–147

26. Cook, T.D., Campbell, D.T.: Quasi-Experimentation: Design & Analysis Issues for Field Settings. Rand McNally College Publishing Company, Chicago, IL, USA (1979)
27. Wentem, M.N., Seiler, C.M., Uhl, W., Buchler, M.: Perspectives of evidence-based surgery. Digestive Surgery **20** (2003) 263–269
28. Williams, L., Krebs, W., Layman, L., Antsn, A.I.: Toward a framework for evaluating extreme programming. In: Proceedings of the 8th Conference on Evaluation & Assessment in Software Engineering (EASE 2004), Stevenage, UK, IEE Press (2004) 135–144