

Why and How to Substantiate the Good of our Reverse Engineering Tools?*

David Röthlisberger
Software Composition Group, University of Bern, Switzerland
roethlis@iam.unibe.ch

Abstract

Researchers and practitioners are usually eager to develop, test and experiment with new ideas and techniques to analyze software systems and/or to present results of such analyzes, for instance new kind of visualizations or analysis tools. However, often these novel and certainly promising ideas are never properly and seriously empirically evaluated. Instead their inventors just resort to anecdotal evidence to substantiate their beliefs and claims that their ideas and the realizations thereof are actually useful in theory and practice. The chief reason why proper validation is often neglected is that serious evaluation of any newly realized technique, tool, or concept in reverse engineering is time-consuming, laborious, and often tedious. Furthermore, we assume that there is also a lack of knowledge or experience concerning empirical evaluation in our community. This paper hence sketches some ideas and discusses best practices of how we can still, with moderate expenses, come up with at least some empirical validation of our next project in the field of reverse engineering.

Keywords: reverse engineering, software analysis, empirical software engineering, evaluation, validation

1 Introduction

In recent years, the reverse engineering community invented and realized many interesting, promising and potentially practically useful techniques, concepts, processes, or tools such as Moose [4], Program Explorer [8], Graphtrace [7], or Mondrian [11]. Many useful visualizations of software systems, presenting on a high or low level both software structure and behavior have been invented (for instance, Polymetric Views [9], Class Blueprints [3], Real-time visualization [14], or even 3D visualizations [5]).

While all these innovations are promising to actually support software developers in various software engineering tasks such as gaining an understanding for an unfamiliar

application, analyzing collaborations between source artifacts, correcting software defects, or extending and enhancing key software features, nobody really knows whether any of those invented techniques contributes any value in practice, and if so, to which degree these techniques add value. Thus many questions are left open, for example how big is the impact of a given reengineering tool or technique on productivity or whether practitioners actually employ the concepts our community developed. Another interesting question is which techniques are best suited for which kind of software tasks, that is, which techniques or tools should a software developer use when faced with a task in areas such as defect correction, feature implementation, refactoring, or porting an application from one programming language to another. Certainly we are also interested in learning how a specific reverse engineering tool or technique is best employed in practice to be most helpful. As long as these questions remain unanswered and largely neglected, we cannot be sure whether our work has an impact on software developers working in industry on practical, real-world software development and maintenance problems and challenges.

For most of the fore-mentioned tools, techniques, or visualizations we completely lack quantitative knowledge about their practical usefulness. Thus we discuss in this paper several means how we can concretely acquire such crucial knowledge. The ideas presented here should serve as a thought-provoking impulse for how to evaluate and validate existing reverse engineering tools or at least newly invented and implemented tools. Moreover, we want to provoke a discussion in the community to sensitize researchers and practitioners in our field for the importance of empirical research and to motivate them to actually perform such validation, that is, to conduct empirical studies and experiments. The ideas for empirical evaluations in our field presented in the following should give the basic knowledge about how to get started with empirical research.

This paper is structured as follows: In Section 2 we stress the importance of empirical validation with some concrete examples of the benefits of such validation. Section 3 presents and discusses different kinds of empirical validation, and how we can apply them to the context of reverse

*In: 3rd Workshop on FAMIX and Moose in Reengineering, 2009

engineering research. Finally, Section 4 wraps up the paper with some concluding remarks.

2 Why Empirical Validation is Important

In this section we briefly list the three most important reasons why our community should really invest in empirical research:

- Validating our work is crucial, particularly in research. Empirical evidence is hereby a bold argument for the quality of our research projects.
- Quantitative empirical evidence allows us to compare different reengineering techniques, also with respect to types of software engineering tasks for which they are useful, and to thus develop guidelines in which kind of software tasks we should use which technique or combinations thereof.
- Our primary goal as reverse engineering researchers should eventually be to have the largest possible impact on software development and maintenance in practice. Only validating our tools empirically in practice can give us information about how to actually achieve this goal and to learn what to improve to maximize our impact on practice.

3 How to Conduct Empirical Evaluation

In this section we discuss different means to empirically validate tools and techniques developed to reverse engineer software systems. We start with means focusing on acquiring qualitative feedback assessing the usability of evaluated techniques while later approaches are also able to quantitatively report on the impact of these techniques on, for instance, productivity of software developers. Quantitative measures are considered as a more powerful validation as their informative value is higher [10], in particular as concrete numbers allow researches to compare the performance of different techniques. This is not directly possible with qualitative evaluations, yet they still provide important insights in a more explorative way than quantitative evaluations can do; that is why we consider both, qualitative and quantitative evaluation procedures, as valuable.

3.1 Surveys

A survey gathers data by asking a group of people their thoughts, reactions or opinions to fixed questions. This data is then collected and analyzed by the experimenters to obtain knowledge about how the interrogated persons consider, for instance, the usefulness of a visualization to understand run-time collaborations between different modules

of a software system. Often surveys ask subjects to rate specific aspects or impacts of a technique or a tool in a Likert scale (typically from 1, “not useful”, to 5, “very useful”). Analyzing surveys with such questions yields quantitative data, however, this data is based on personal and subjective judgement of the subjects. Thus the outcome of surveys cannot be considered as highly reliable as it is too much dependent on the specific subjects interrogated.

Surveys have several benefits as well as disadvantages: They are cost effective and efficient as a large group of people can be surveyed in a short period of time. The disadvantages of using surveys to conduct research include the validity based upon honest answers. Answer choices could not reflect true opinions and one particular response may be understood differently by the subjects of the study, thus providing less than accurate results.

3.2 Case Studies

Unlike a survey, a case study closely studies an individual person or an individual application to be analyzed. Surveys always ask several people and might consider several systems being analyzed or covered by the reverse engineering tools under study. Case studies were developed from the idea of a single case (*e.g.*, a single application to be analyzed by a new technique) being tried in a court of law. Comparing one case study to another is often difficult, thus it is usually not possible to draw a significant conclusion from one or from a low number of similar case studies. However, a case study can often be seen as a starting point for further analysis, for instance by testing how people use and interact with a new visualization tool or a new analysis procedure. A case study can thus serve as a pre-study to gather preliminary, qualitative feedback and insights to learn how to design another, possibly controlled study which eventually quantitatively evaluates the impact of the reverse engineering technique on the variable of interest (*e.g.*, programmer productivity).

The benefits of performing a case study thus include getting an in-depth view into subjects behavior and can also help to determine research questions to study for a larger group. A disadvantage of case studies is that the researcher may start off only looking for certain data such as specific usage patterns of the studied tool. Such a narrowed focus might lead to overlooking other interesting and important aspects, such as shortcomings or hidden benefits of the tool or technique under study. A way to mitigate this threat is to not conduct a case study with a specific goal, *e.g.*, testing a particular hypothesis. Rather researchers should employ case studies purely as an explorative means to unprejudicedly observe how a reverse engineering tool is perceived and used by study subjects to actually benefit from the advantages of case studies, for instance their potential to re-

Table 1. Comparison of different kinds of empirical studies

<i>Study</i>	<i>Costs</i>	<i>Quantitative</i>	<i>Qualitative</i>	<i>Generalizability</i>	<i>Reproducibility</i>
Survey	+	Partially	Yes	+	-
Case Study	++	No	Yes	--	--
Observational Study	+	No	Yes	-	-
Contest	-	Partially	Partially	+	+
Controlled Experiment	---	Yes	No	++	++

veal unknown and unanticipated behavior of people concerning how they use a tool.

3.3 Observational Studies

An observational study is similar to a case study. Typically, a observational study includes several single and possibly independent case studies with, for instance, several subjects, software systems to be analyzed, or analysis tools and techniques being used. Thus observational studies help experimenters to generalize the findings and results of one case study to a broader context.

Conducting observational studies has similar advantages and disadvantages as case studies. Of course they require more time and effort to conduct than just a single case study, the amount of insights and reliability they yield is larger and broader though.

3.4 Contests

Contests are another, rather specialized means to evaluate reverse engineering tools. The basic idea is to let different subjects or different analysis tools compete against each other, for instance by imposing some tasks or questions on the subjects that have to solve these problems under time pressure and while competing against other subjects. The eventual goal is to win the contest, for instance by solving the problem in the shortest amount of time or in the highest quality. The performance of each subject is analyzed, the experimenters observe the contest and try to locate the cause why and how much a specific reverse engineering tool helped subjects to solve the problem in a particular time or accuracy.

The fundamental difference to an observational study is the fact that a contest puts significant time pressure on the subjects; thus they are more motivated to solicit the best out of the available tools they obtain to solve the given problem. Thus the results might be more reliable and comparable between different employed tools as the motivation of the subjects was high to do their best to learn, understand and employ a tool in the most efficient manner.

3.5 Controlled Experiments

(Controlled) experiments are crucial in finding the answers to questions such as “does tool X provide value in software maintenance activities”. The cause and effect of a particular problem can be studied through an experiment, providing it has “a set of rules and guidelines that minimize the possibility of error, bias and chance occurrences” [6]. Surveys and case studies require observation and asking questions, whereas experiments require controls and the creation of constrained situations to be able to actually record beneficial data.

The benefits of controlled experiments include a more scientific and thus more accepted approach of collecting data, as well as limiting potential bias that could occur in a survey, case study or an observational study. One disadvantage to many experiments is the cost factor involved; often laboratory procedures can be expensive. Another potential disadvantage is that a controlled experiment virtually always has a narrow focus, that is, it tries to study one or several very particular variables and thus neglecting any other, potentially interesting analysis of, for instance, developer reactions to specific characteristics of a reverse engineering technique.

Examples of well conducted controlled experiments in the area of software engineering and specifically in the field of program comprehension, software analysis, and reverse engineering are for instance: Cornelissen *et al.* [2] that evaluated EXTRAVIS, a trace visualizing tool, with 24 student subjects. Quante *et al.* [13] evaluated by means of a controlled experiment with 25 students the benefits of Dynamic Object Process Graphs (DOPGs) for program comprehension. Arisholm *et al.* [1] quantitatively analyzed in a large, long-lasting controlled experiment with 295 professional software engineers on three different levels (junior, intermediate, expert) whether pair programming has a positive impact on time required, correctness and effort for solving programming tasks with respect to system complexity and programmer expertise.

3.6 Summary

Table 1 summarizes the different kinds of empirical studies we presented in this section. The table shows for in-

stance which studies to use to gather quantitative or qualitative feedback (note that each study can certainly be extended to gather qualitative feedback as well even though its primary goal is to quantitatively analyze cause-effect relationships, for instance by observing the study subjects or by additionally handing out a questionnaire). With “generalizability” we refer to whether the study concept itself easily allows us to obtain results that can be generalized to practical, real-world situations. Generalizability is certainly very much dependent on how a concrete study is designed. With “reproducibility” we suggest how likely a type of study is to yield the same results when conducted several times.

Researchers sometimes only require the use of one of the discussed methods to successfully find the answer to a question, however it is often worthwhile to combine different methods when evaluating a specific research project as each evaluation method has its specific strengths. Thus a combination of different methods often yields more insights and results that are more reliable and generalizable to other contexts, software systems, or analysis techniques.

Kitchenham *et al.* [6] thoroughly report on preliminary guidelines for empirical research in software engineering; these guidelines are also applicable to the field of reverse engineering and reengineering. Di Penta *et al.* [12] encourage researchers and practitioners to design and carry out empirical studies related to program comprehension and to develop standards how to conduct such studies. They give some early hints how to establish a community motivated to perform empirical engineering in our field.

4 Conclusions

In this paper we first motivated the need for conducting empirical research in the reverse engineering field by raising some important questions that remain unanswered as long as only very little empirical validation on our techniques and tools is performed. In a second step, we introduced several types of empirical studies such as surveys, case studies, or controlled experiments that we can employ to actually conduct empirical studies. Moreover, we gave some hints how to concretely conduct which kind of study in which context and referred to important work on empirical software engineering done by other researchers.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects “Bringing Models Closer to Code” (SNF Project No. 200020-121594, Oct. 2008 – Sept. 2010).

References

[1] E. Arisholm, H. Gallis, T. Dyba, and D. I. Sjöberg. Evaluating pair programming with respect to system complexity

- and programmer expertise. *IEEE Transactions on Software Engineering*, 33(2):65–86, 2007.
- [2] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 35(5):684–702, 2009.
- [3] S. Ducasse and M. Lanza. The Class Blueprint: Visually supporting the understanding of classes. *Transactions on Software Engineering (TSE)*, 31(1):75–90, Jan. 2005.
- [4] S. Ducasse, M. Lanza, and S. Tichelaar. Moose: an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems. In *Proceedings of CoSET '00 (2nd International Symposium on Constructing Software Engineering Tools)*, June 2000.
- [5] D. Holten, R. Vliegen, and J. J. van Wijk. Visual realism for the visualization of software metrics. In *VISSOFT*, pages 27–32, 2005.
- [6] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.*, 22(8):721–734, 2002.
- [7] M. F. Kleyn and P. C. Gingrich. GraphTrace — understanding object-oriented systems using concurrently animated views. In *Proceedings of International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'88)*, volume 23, pages 191–205. ACM Press, Nov. 1988.
- [8] D. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. In *Proceedings ACM International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'95)*, pages 342–357, New York NY, 1995. ACM Press.
- [9] M. Lanza and S. Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. *Transactions on Software Engineering (TSE)*, 29(9):782–795, Sept. 2003.
- [10] T. C. Lethbridge, S. E. Sim, and J. Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering, Springer Science and Business Media, Inc., The Netherlands*, 10(3):311–341, July 2005.
- [11] B. A. Myers, D. A. Weitzman, A. J. Ko, and D. H. Chau. Answering why and why not questions in user interfaces. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 397–406, New York, NY, USA, 2006. ACM Press.
- [12] M. D. Penta, R. E. K. Stirewalt, and E. Kraemer. Designing your next empirical study on program comprehension. In *Proceedings of the 15th International Conference on Program Comprehension*, pages 281–285, Washington, DC, USA, 2007. IEEE Computer Society.
- [13] J. Quante. Do dynamic object process graphs support program understanding? - a controlled experiment. In *Proceedings of the 16th International Conference on Program Comprehension (ICPC'08)*, pages 73–82, Washington, DC, USA, 2008. IEEE Computer Society.
- [14] S. P. Reiss. Visualizing Java in action. In *Proceedings of SofiVis 2003 (ACM Symposium on Software Visualization)*, pages 57–66, 2003.