

# Designing your Next Empirical Study on Program Comprehension

Massimiliano Di Penta\*, R.E.K. Stirewalt\*\*, Eileen Kraemer\*\*\*  
dipenta@unisannio.it, stire@cse.msu.edu, eileen@cs.uga.edu

\* RCOST — Research Centre on Software Technology,  
University of Sannio, Via Traiano 82100 Benevento, Italy

\*\*Computer Science and Engineering, Michigan State University, East Lansing, MI

\*\*\*Computer Science Department, The University of Georgia, Athens, GA

## Abstract

*The field of program comprehension is characterized by both the continuing development of new tools and techniques and the adaptation of existing techniques to address program comprehension needs for new software development and maintenance scenarios. The adoption of these techniques and tools in industry requires proper experimentation to assess the advantages and disadvantages of each technique or tool and to let the practitioners choose the most suitable approach for a specific problem.*

*The objective of this working session is to encourage researchers and practitioners working in the area of program comprehension to join forces to design and carry out studies related to program comprehension, including observational studies, controlled experiments, case studies, surveys, and contests, and to develop standards for describing and carrying out such studies in a way that facilitates replication of data and aggregation of the results of related studies.*

**Keywords:** Empirical Software Engineering, Program Comprehension, Design of Experiments.

## 1 Introduction

Research in program comprehension has been characterized by a flourishing diffusion of novel approaches and tools. Inherent to each is a claim that the use of the approach or tool will provide some benefit to the user with respect to some comprehension task. Ultimately, such claims must be validated empirically so that a practitioner, when looking to adopt a particular technique or tool, can make the choice based on rigorous scientific data. There are several different types of empirical study—e.g., *surveys*, *case studies*, and *controlled experiments* [9]—each providing a different degree of strength and each appropriate to validate

different kinds of empirical questions and contexts.

When designing and conducting a study, one is confronted with a large space of decisions that must be made and data that must be collected. For example, one must choose the variables that are to be measured, record the contextual factors that affect the study, choose the subjects and objects of study, etc. If the wrong choices are made, the data may fail to support even a true hypothesis. Insufficient documentation of contextual factors may stymie the efforts of others to replicate results. If an insufficient number of data points are collected, a result may fail to be statistically significant. In summary, the effort required to design and conduct an empirical study is large and tedious, and many opportunities for error exist during the design of a study.

We believe these problems may be addressed by brokering collaborations within our community to:

1. bring many hands to bear in the empirical investigation of specific research questions,
2. collect and share “best practices” information regarding the design and conduct of empirical studies in program comprehension, and
3. develop community-wide resources—e.g., analysis/recording tools, rubrics and coding schemes, and repositories—that may be applied by many researchers to investigate many different research questions.

This working session is a forum that supports all of these activities. Specifically, we encourage researchers and practitioners working on program comprehension to combine their efforts in designing, preparing, and executing empirical studies and in disseminating the results. Researchers will submit to the session organizers ideas about new and interesting empirical studies and then present these ideas at the beginning of the working session. Session attendees will then work in teams to design studies, which they will carry

out after the conference, thus permitting each participant to contribute to the study in a different way—e.g., providing technology expertise, objects of study (e.g., software systems), populations of subjects, data analysis expertise, etc.

This paper is organized as follows. Section 2 discusses the relevant literature of the field, while Section 3 discusses relevant issues in designing and carrying on empirical studies on program comprehension. Sections 4 and 5 describe how the working session will be organized and the intended attendee population. Finally, Section 6 concludes by reiterating the purpose and utility of the working session.

## 2 Related Work

Some of the earliest attempts at empirical assessment in our field grew from the need to objectively compare the myriad tools and techniques that researchers have proposed to support reverse engineering and design recovery. Brown and Wallnau developed a general framework for technology evaluation [5], which Gannod and Cheng then adapted to classify and compare reverse-engineering and design-recovery tools [10]. More recently, Guéhéneuc *et al.* [11] proposed a framework to compare design recovery tools. The important role played by empirical evaluation in reverse engineering (and thus also in program comprehension) has been discussed by L. Briand in his keynote speech at WCRE 2006 [3]. He identified three dimensions to evaluate reverse engineering research: inputs, analysis process, and outputs. Outputs need to be evaluated in terms of correctness, completeness and usefulness to carry out a specific task. The analysis process needs to be evaluated in terms of performance and scalability.

The definition of proper benchmarks [8] can help to better compare approaches. As in other fields, such as information retrieval, software engineering researchers need to define benchmark sets of software artifacts, upon which to base comparisons of different approaches and tools. The existence of such benchmarks would provide advantages when performing experiments (direct comparison of results, ease of replication) or case studies (flexibility and robustness of the method). On the other hand, limited benchmarks might not permit the generalization of results.

In addition to technology-centric criteria, a tool that purports to aid in program comprehension should be assessed for its support (or hindrance) of an engineer's ability to form or manipulate the mental models required to solve a program-comprehension task. Storey and others performed user studies to uncover how and how well various program-understanding tools actually help programmers to understand programs [19, 20, 21]. An example of how the combination of quantitative data from a controlled experiment and of results from source code peer-review can be used to support the evaluation of a traceability support tool can be

found in a paper by De Lucia *et al.* [6].

A wide variety of experiments have been performed to assess to what extent different formalisms support program comprehension. The usefulness of graphical elements in software architecture diagrams, and in particular of UML stereotypes, has been assessed experimentally by Bratthall and Wohlin [2]. The same group [17] also replicated the study in the context of improving reading techniques for software artifacts. Recently, Ricca *et al.* [16] analyzed the effect of subjects' ability and experience on the understanding of UML stereotypes, finding that stereotypes are able to reduce the gap between highly experienced subjects and junior developers.

Experiments aiming at studying the impact of UML documentation in software maintenance [1] indicate that such documentation improves the functional correctness of changes and the quality of the design. This also happens when UML is complemented with complex formalisms, such as the Object Constraint Language (OCL) [4]: substantial training is required to make OCL useful, although for some tasks, such as defect detection, an interaction between ability and treatment was detected. OCL better helped low ability subjects, who were not able to guess system functionality from the textual description. Lawrance *et al.* found that graphical visualizations of code coverage information helps end-user programmers, while the same does not happen for professional developers [14].

In addition to formal experiments, Fenton and Pfleeger identify two other modes of empirical assessment—*surveys* and *case studies* [9]. Guidelines on how to plan and conduct case studies can be learned from a well-known social science book by Yin [25]. Which mode to use depends on several factors including sample size and whether the study intends to confirm or refute an explicit theory or to elicit observations from which to construct such a theory. General guidelines for designing studies in any of these modes can be found in [12, 13, 23].

Finally, Von Mayrhauser and Lang [22] argue that much of what we currently know about program comprehension is based on *observational studies*. These studies typically focus on understanding the user's current approach to problem-solving or task performance or seek to comprehend the user's mental model of a concept, procedure or software artifact. Approaches may include the use of a "think aloud" protocol while the user performs a comprehension task [22] or the use of eye-tracking equipment. The resulting data can be analyzed to construct or elaborate cognitive theories of perception or comprehension, which may then be subjected to formal experiments.

### 3 Issues in Designing and Documenting Program Comprehension Experiments

In addition to brokering collaborations among researchers, we hope that this working session and its successors will begin to address some of the issues that confront a researcher who intends to carry out an empirical study of program comprehension. As suggested previously, the design of such a study is tricky, and there are many obstacles to replication. At a similar workshop at ICPC 2006, Di Lucca and Di Penta began to codify some of the open issues that affect the design, execution, analysis, and replication of program-comprehension experiments [7]. These issues include:

1. Identification of key variables to measure during such an experiment. Example variables include *accuracy* of user actions with respect to a given task and *time to completion* of a comprehension task [15]. New technologies, such as eye-tracking devices, suggest other variables to measure. As yet, we lack a general taxonomy that precisely defines and relates these variables, documenting potential interactions and other confounding factors.
2. Identification of an appropriate subject population for a study. While students are often the most convenient to enlist, a subject group made up entirely of students might not adequately represent the intended user population. Open issues include how best to engage industrial practitioners in empirical studies and how to judge when a subject population made up of students is sufficiently representative.
3. Identification of appropriate benchmark systems or tasks. Benchmarks facilitate direct comparisons of alternative approaches and may permit the meaningful aggregation of data from multiple studies.
4. Standardization of the experimental design format, which would facilitate the replication of experiments and the aggregation of data from multiple experiments.
5. Development of standardized instrumentation and analysis methodologies for approaches that involve the tracking of user interactions, such as eye-tracking and other forms of interaction-logging.
6. Standard procedures and criteria for packaging the materials and the results of a study so as to facilitate replication.

This year, in addition to further articulating and elaborating these issues, we hope to develop collaborative efforts

whose execution will address one or more of them. For example, *meta-analysis* is a well-known technique for aggregating the results of multiple studies to improve the statistical significance of the (combined) result. Often, a single research group will conduct a user study whose result suggests a trend but lacks statistical significance because the group could not assemble a sufficiently large subject pool. If multiple research groups could run independent studies using the same materials, a meta-analysis may be able to combine these studies to yield a statistically significant result. By planning these independent studies and then conducting a meta-analysis, we could learn much about how to deal with this general problem.

Another open issue that is “ripe” for redress via collaboration concerns how to package materials and results for replication. To facilitate the eventual emergence of general guidelines, we will encourage collaborators to think about what kinds of information and materials one group member would need from another in order to replicate a specific study. For example, a controlled experiment using a pre-test/post-test design to assess the value of a treatment might require the following artifacts to be made available to facilitate replication and evaluation:

- subject selection criteria and justification,
- subject screening materials and results (with private information replaced by unique identifiers),
- pre-test questions, and results keyed to the unique subject identifiers, as well as an explanation of the knowledge that the questions are designed to evaluate,
- control and treatment groups (i.e., sets of subject identifiers),
- post-test design and control/treatment group materials, as well as an explanation of the knowledge the post-test questions are designed to evaluate, and
- if different instructions are given to control and treatment groups, some summary of the contents of these instructions.

In addition to these issues, we encourage the submission of studies that assess design methods which are specifically developed to yield artifacts that improve program comprehension. Example notations might include the *synchronization contracts* used in Szumo [18] or the extended sequence diagram notation of saUML [24]. To assess the comprehensibility of such a notation, the researcher might need to prepare two independent programs, one developed using the “status quo” notation(s) and one using the method that is supposed to yield more comprehensible artifacts. Here, the open issue is how to choose a problem that is not biased to favor the method being assessed.

## 4 Working session organization

As mentioned in the introduction, this working session aims to foster collaboration among the participants to develop and organize studies related to program comprehension. Interested participants are encouraged to submit to the workshop organizers a 1-2 page proposal that outlines an empirical study of the type described in Section 2. For example, a proposal could describe:

- an *observational study* of how programmers think about a concept or approach a problem-solving task.
- a *controlled experiment* involving human subjects that assesses the usefulness of some tool or technique;
- a *contest* that compares alternative techniques/tools for performing a given task, e.g., design pattern identification, reverse engineering of object-oriented code, etc.;
- a *case study* that investigates the application of some program-comprehension techniques or tool on realistic industrial or academic projects;
- a *survey* that investigates some topic, such as the adoption of comprehension techniques in industry.

During the first part of the working session, each proposer will briefly describe the study, with the objective of advertising the idea and recruiting other potentially interested researchers. Participants will then cluster into groups based on interest in a particular study. Each group will then be tasked with clearly defining and designing the study, e.g., identifying experimental objects and subjects, planning replications, etc. Each group member should be able to contribute to the study in one or more ways, e.g.:

- providing the techniques or tools to be used in the study;
- providing experimental subjects, e.g. students taking a software engineering course or groups of professionals in industry, with the aim of carrying out experimental replications;
- providing objects for the empirical study, such as software systems to be analyzed, industrial project data, etc.

Having completed their work, each group will briefly present the design of the study, including the specific execution procedures and collaboration plans, and will solicit feedback from the larger group.

## 5 Intended Participants

This working session will include participants of various backgrounds, including academics, students, and practitioners interested in empirical studies related to program comprehension. An attendee could be:

1. a proposer of some empirical study, looking for collaborators to help in the planning and execution of the study and in the dissemination of its results.
2. someone who wishes to get involved or otherwise collaborate in one of the proposed studies.
3. someone looking to gain insights into the issues related to the planning and execution of empirical studies of program comprehension, report personal experience, and/or provide feedback and stimulate the discussion.

## 6 Conclusions

To be adopted in practice, a program-comprehension technique or tool must be shown to produce benefits during maintenance tasks. This requirement must be fulfilled via empirical studies, of which there are many different kinds—e.g., comparing alternative techniques, surveying the industrial adoption of program-comprehension technologies, or evaluating the effectiveness of visualization techniques. The design and operation of such studies requires different kinds of expertise—e.g., expertise in some specific technique or tool, statistical analysis, or cognitive studies. In addition, the need exists for benchmark systems and pools of subjects to be involved in controlled experiments, as well as of industrial partners to be involved in survey studies. It is therefore highly desirable for researchers to share their resources and expertise to collaborate in the conduct of such studies.

This working session will be an opportunity for researchers and practitioners to cooperate in conceiving, preparing and executing empirical studies that would have not been possible relying on the efforts and resources of individual researchers. Further, it will be an opportunity for researchers in the area to plan and begin research collaborations.

## References

- [1] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche. The impact of UML documentation on software maintenance: An experimental evaluation. *IEEE Transactions on Software Engineering*, 32(6):365–381, 2006.
- [2] L. Bratthall and C. Wohlin. Is it possible to decorate graphical software design and architecture models with qualitative information? -An experiment. *IEEE Trans. Softw. Eng.*, 28(12):1181–1193, 2002.

- [3] L. C. Briand. The experimental paradigm in reverse engineering: Role, challenges, and limitations. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), October 2006, Benevento, Italy*, pages 3–8, Los Alamitos, CA, USA, 2006. IEEE Computer Society.
- [4] L. C. Briand, Y. Labiche, M. Di Penta, and H. D. Yan-Bondoc. An experimental investigation of formality in UML-based development. *IEEE Transactions on Software Engineering*, 31(10):833–849, 2005.
- [5] A. W. Brown and K. C. Wallnau. A framework for evaluating software technology. *IEEE Software*, 13(5):39–49, Sept. 1996.
- [6] A. De Lucia, M. Di Penta, R. Oliveto, and F. Zurolo. Improving comprehensibility of source code via traceability information: a controlled experiment. In *Proceedings of the 14th International Conference on Program Comprehension (ICPC 2006), June 2006, Athens, Greece*, pages 317–326, Los Alamitos, CA, USA, 2006. IEEE Computer Society.
- [7] G. A. Di Lucca and M. Di Penta. Experimental settings in program comprehension: Challenges and open issues. In *Proc. of the International Conference on Program Comprehension (ICPC'06)*, pages 229–234, 2006.
- [8] S. Elliott Sim, S. M. Easterbrook, and R. C. Holt. Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, pages 74–83, 2003.
- [9] N. E. Fenton and S. L. Pflieger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., 1998.
- [10] G. C. Gannod and B. H. C. Cheng. A framework for classifying and comparing software reverse engineering and design recovery techniques. In *Proceedings of IEEE Working Conference on Reverse Engineering*, pages 77–88, 1999.
- [11] Y.-G. Gueheneuc, K. Mens, and R. Wuyts. A comparative framework for design recovery tools. In *10th European Conference on Software Maintenance and Reengineering (CSMR 2006), 22-24 March 2006, Bari, Italy*, pages 123–134, 2006.
- [12] N. Juristo and A. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2003.
- [13] B. Kitchenham, S. Lawrence Pflieger, L. Pickard, P. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Software Eng.*, 28(8):721–734, 2002.
- [14] J. Lawrance, S. Clarke, M. M. Burnett, and G. Rothermel. How well do professional developers test with code coverage visualizations? an empirical study. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), 21-24 September 2005, Dallas, TX, USA*, pages 53–60, 2005.
- [15] V. Rajlich and G. S. Cowan. Towards standard for experiments in program comprehension. In *5th International Workshop on Program Comprehension (WPC '97), May 28-30, 1997 - Dearborn, MI, USA*, pages 160–161, 1997.
- [16] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, and M. Ceccato. The role of experience and ability in comprehension tasks supported by uml stereotypes. In *Proceedings of the International Conference on Software Engineering, May 2007 (to appear)*.
- [17] M. Staron, L. Kuzniarz, and C. Thurn. An empirical assessment of using stereotypes to improve reading techniques in software inspections. In *3-WoSQ: Proceedings of the third workshop on Software quality*, pages 1–7, New York, NY, USA, 2005. ACM Press.
- [18] R. E. K. Stirewalt, R. Behrends, and L. K. Dillon. Safe and reliable use of concurrency in multi-threaded shared memory systems. In *Proc. of the 29th Annual IEEE/NASA Software Engineering Workshop*, 2005.
- [19] M.-A. D. Storey. Theories, methods and tools in program comprehension: Past, present and future. In *13th International Workshop on Program Comprehension (IWPC 2005), 15-16 May 2005, St. Louis, MO, USA*, pages 181–191, 2005.
- [20] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software visualization. In *5th International Workshop on Program Comprehension (WPC '97), May 28-30, 1997 - Dearborn, MI, USA*, pages 17–28, 1997.
- [21] M.-A. D. Storey, K. Wong, and H. A. Müller. How do program understanding tools affect how programmers understand programs? In *Proceedings of IEEE Working Conference on Reverse Engineering*, pages 12–, 1997.
- [22] A. von Mayrhauser and S. Lang. A coding scheme to support systematic analysis of software comprehension. *IEEE Transactions on Software Engineering*, 25(4):526–540, July/August 1999.
- [23] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.
- [24] S. Xie, E. T. Kraemer, and R. E. K. Stirewalt. Empirical evaluation of a UML sequence diagram with adornments to support understanding of thread interactions. In *Proc. of the 15th IEEE International Conference on Program Comprehension (ICPC'07), June 2007*.
- [25] R. K. Yin. *Case Study Research: Design and Methods - Third Edition*. SAGE Publications, London, 2002.