H. Dieter Rombach Victor R. Basili
Richard W. Selby (Eds.)

Experimental Software Engineering Issues:

Critical Assessment and Future Directions

International Workshop
Dagstuhl Castle, Germany, September 14-18, 1992
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

H. Dieter Rombach
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, D-67653 Kaiserslautern, Germany

Victor R. Basili
Department of Computer Science, University of Maryland
College Park, Maryland 20742, USA

Richard W. Selby
Department of Information & Computer Science, University of California
Irvine, CA 92717, USA

CR Subject Classification (1991): D.2, K.6

ISBN 3-540-57092-6 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57092-6 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author
45/3140-543210 - Printed on acid-free paper

Preface

Experimental Software Engineering Issues: Critical Assessment and Future Directions

Context

Since its inception in 1968, software engineering has struggled to find its identity. Today, we can identify three different approaches to study of the discipline of software engineering in the research community: the mathematical or formal methods approach, the system building approach, and the empirical studies group. Within the mathematical or formal methods group, the emphasis is on finding better formal methods and languages and software development is viewed as a mathematical transformation process. Within the system building group, the emphasis is on finding better methods for structuring large systems and software development is viewed as a creative task which cannot be controlled other than through rigid constraints on the resulting product. Within the empirical studies group, the emphasis is on understanding the strengths and weaknesses of methods and tools in order to tailor them to the specific goals of a particular software project.

The purpose of this workshop was to gather those members of the software engineering community who support an engineering approach, based upon empirical studies, to provide an interchange of ideas and paradigms for research.

Software engineering based upon empirical studies is made difficult when one observes that in practical software organizations, project contexts (i.e., project goals and environmental characteristics) vary from project to project. Thus, no single technology or method can be expected to work well in all contexts, and observing software phenomena out of context seems to be doomed to fail. As part of the learning process, we need to characterize and understand the project context and understand the various phenomena relative to that context and learn in an incremental and evolutionary manner. We need to replicate experiments in different contexts to fully understand the nature of the various phenomena and be able to build models to facilitate learning.

Improvement oriented approaches that take into account the evolutionary and experimental nature of software have recently been suggested as a framework for studying the relationships between product and knowledge engineering. This framework bears the potential of integrating the efforts of the formal methods, system building, and empirical studies approaches in a promising way. These improvement approaches are based on the use of empirical technology for building models. Formal methods as well as system building technology can be elevated to the level of useful technology from an engineering perspective if augmented with knowledge of their effectiveness based on empirical evidence. Other frameworks with similar objectives have been suggested too.

After twenty-five years of software engineering it seemed appropriate to rethink its scientific and engineering basis. Based on the increasing demands imposed on our field by the ever-increasing complexity and criticality of software related applications, a

move towards an engineering view of our field is needed. Such a move must not be construed as a competition between the mathematical, system building, and empirical studies approaches. Instead, it suggests that all three are necessary, but that we cannot ignore the nature of our field, which requires more than devising new languages and techniques and more than just building systems which can be judged at the end. We need to do all of this in a framework which enables us to understand all existing and new technologies, and use them in a controlled fashion to develop the systems required by our customers.

Objectives

We have only begun to understand the experimental nature of software engineering, the role of empirical studies and measurement within software engineering, and the mechanisms needed to apply them successfully. Workshop discussion was focused on assessing past accomplishments within the experimental software engineering community and proposing necessary future steps. The topics of discussion included several of the most eminent challenges within experimental software engineering:

(1) Identifying the appropriate paradigm for software engineering:
Should we adapt the mathematical approach or the experimental approaches used in physical or social sciences? For what purposes do we need empirical studies in experimental software engineering? What are software-specific constraints or requirements for empirical studies?

(2) Understanding the range of different contexts for empirical studies in software engineering:
Why do we measure? What is it we want to know? How do the changing project contexts affect our ability to measure?

(3) Devising the appropriate procedures and mechanisms for empirical studies:
How should we perform empirical studies? How should we specify the objectives and context of studies? How should we determine the appropriate measures for a given objective? How should we design the appropriate experiments or case studies? How should we collect and validate product and process data?

(4) Guiding the use of empirical data to build or improve existing software models:
What are the appropriate analysis procedures for software engineering data? How can these procedures help us create models of software processes and products? What alternatives exist to model building based on empirical data?

(5) Identifying appropriate concepts and mechanisms for packaging existing models for reuse across projects:
What makes models reusable? How do we determine the needs for reuse? How should we organize and build up reusable model libraries? What mechanisms are needed to support reuse of models across projects?

(6) Proposing appropriate means of distributing experimental ideas to practitioners and students:
How do we make improvement happen in practice? What organizational structures are needed to support technology transfer, especially what roles can universities and industry play? How can we change our university curricula in order to instill ideas of empirical studies into students early on? How can we train practitioners in the experimental paradigm of software engineering?

Session Organization

In order to address all these challenges, an international workshop on the topic "Experimental Software Engineering Issues" was organized and held at the International Conference and Research Center for Computer Science (IBFI) at Dagstuhl Castle in the Federal Republic of Germany. The motivation for this workshop was to provide a forum for a relatively small but representative group of leading experts in experimental software engineering with an emphasis on empirical studies from both universities and industry to meet and reflect on past successes and failures, assess the current state-of-the practice and research, identify problems, and define future directions. An organizing committee identified key topics and key people to participate in the workshop. The six challenges above were chosen for discussion along with people to present keynote presentations and chair those sessions. A final session was aimed at devising an agenda for the future.

After the selection of discussion topics, keynoters and session chairs was made, approximately thirty more participants were invited to submit position papers on one of the selected topics. The participants came from Europe, the United States and Canada, Asia and Australia. During each session, a keynote presentation was followed by a number of position statements and extensive discussion. The materials contained in this volume include for each session the keynote address, position papers, and a discussion summary.

This workshop was scheduled to run from Monday, September 14, through noon on Friday, September 18. Six half-day sessions (Monday, Tuesday, Thursday) were devoted to the topics listed. Each session was organized by a session chair, and introduced by a keynote presentation intended to provide a critical assessment of the topic at hand and to make provocative statements to stimulate discussion. The keynote was complemented by a number of position statements. The major portion of each session was reserved for lively discussion. Wednesday was reserved for sightseeing in the city of Trier. The wrap-up session on Friday morning was intended to synthesize the results of a week-long discussion into a statement of where we stand as a field (i.e., what we agree on, what we don't agree on) and devise an agenda for future progress in our field.

Results

The results of the workshop can be summarized in terms of what has been achieved in the past in terms of measurable benefits from the software practitioner's perspective, what lessons have been learned regarding experimental software engineering in gen-

eral and empirical studies in specific, what are remaining key points of dissent, and what are the important topics for future work. The following summaries reflect the consensus achieved in the discussion sessions complemented by the written opinions collected from the workshop attendees via a questionnaire.

(A) **Past Achievements** (Practitioners' Perspective):

Each workshop attendee was in a position to report about empirical studies and/or measurement programs and subjective results ranging from increased understanding of certain software engineering phenomena or the improvements of real-world software processes. Most of these achievements are not well documented or documentation is company-confidential. Most of the achievements are based on empirical studies in very specific contexts and cannot be generalized due to the variability in contexts across different organizations.

The workshop attendees felt that in order to live up to the theme of the workshop, we needed to come up with documented results which could be analyzed by others and could be used to convey the potential value of empirical work in terms of measurable benefits. One outstanding example was reported by Frank McGarry from the Software Engineering Laboratory at NASA's Goddard Space Flight Center and will be mentioned here as an example. Frank McGarry reported the following practical SEL achievements which had been achieved through and demonstrated via empirical studies:

- People oriented technologies are most effective (e.g., inspections, Cleanroom) as opposed to automated tools
- Commonly accepted complexity measures are not very meaningful in our domain
- Ada software costs more to develop; less to deliver because of reuse (multiple experiments)
- Inspections by stepwise abstraction reading are the most effective and most cost effective testing method
- Models/relationships developed are incorporated into management process (e.g., manager's handbook) and supported (e.g., SME)
- The error rates in development projects were reduced significantly through the use of Cleanroom (e.g., 33% on a series of 3 projects)
- Environmental heritage/context/legacy is the dominant impact on processes and products (e.g., use of Ada over time)
- Productive reuse is driven by process reuse and packaging of design - *not* by code packaging (e.g., Ada/OOD experiments)
- Some commonly accepted processes assumed to be beneficial are inappropriate for the SEL (e.g., IV&V)

- Personnel variation in productivity are tremendous (factors of 3 or 4 for large systems; factors of up to 15 in small systems)
- Design structure (strength/coupling) is an excellent predictor of module defects

Examples from other local environments have been provided but cannot be reported in this brief summary.

(B) **Past Achievements** (Researchers' Perspective):

The most important lessons learned about empirical studies and measurement include a broad agreement regarding the experimental nature of software engineering. This experimental nature requires empirical studies as a driver for learning and improvement. Empirical studies need to be performed with a goal and hypothesis in mind and the context characteristics need to be taken into account when interpreting measurement data. A variety of approaches for improvement and goal orientation were surveyed in Vic Basili's keynote address.

The most frequently occurring themes during the discussion and on the questionnaires were:

- Software engineering research needs to be driven by empirical studies
- Metrics and data in isolation (i.e., without context) are useless
- No single set of metrics is universally best
- Sound empirical approaches are essential
- Empirical studies have produced results in local contexts; but we have not been able to generalize local results

As an example, Vic Basili and Frank McGarry from the Software Engineering Laboratory at NASA's Goddard Space Flight Center reported the following research lessons they had learned about the application of empirical studies and measurement over the past fifteen years:

- The purpose of experimentation is for self-improvement and self-understanding rather than inter-organization and inter-country comparison
- Expectation/provision of N to 1 improvement in productivity over finite time (5 to 10 years) is baseless and won't happen!!!
- We are most effective when using multiple processes based on context (e.g., we are using Fortran/Functional-Decomposition-based design/reuse-oriented waterfall, ADA/OOD/reuse-oriented waterfall, and Cleanroom)
- Each of the above technologies and processes had to be tailored to our environment
- Understanding (baselining) is absolutely mandatory as a first step (before planning/controlling, technology transfer)

- Process definition/clarification of the empirical process is mandatory for successful experimentation/measurement (i.e., Doctor heal thyself)
- The process improvement paradigm is equally important for the software development task and the experimentation/data collection task
- The process for empirical studies has to be well defined and improved
- There must be a goal/rationale for data collection
- Data by itself provides minimal, most likely erroneous or detrimental insights
- The measurement data has intrinsic imprecision, inconsistency, and incompletely represented context and it always will. This drives the need to study trends not absolute facts.
- Packaging of experience is key to success - but is rarely done effectively
- Packaging (i.e., development of local standards) needs to be experience driven (e.g, 2167A is an incomplete approach)
- Effective cookbooks can be developed for particular domains (e.g., SEL measurement handbook, SEL management handbook)
- More data does not necessarily mean better results (i.e., national databases for measurement data are a waste of time and resources)
- Experimentation requires two identifiable, separate (but cooperating) organizational infrastructure components, which both involve cost
  - overhead to project (noise -- < 2%)
  - analysis and synthesis of data (8-10%)
  - support (quality assurance, databases, ...)
- Developers treat data collection/experimentation as an annoyance only, not as significant impact
- Infusion of significant process change (e.g., Ada, Cleanroom, OOD) requires 5 to 10 years

(C) Key Points of Dissent:

Although, all workshop participants agreed on the need for employing empirical studies and measurement in order to introduce engineering discipline into the field of software engineering, several points of dissent remained.

Examples of dissent explicitly voiced by participants included:

- Are large-scale, real-world experiments feasible from a scientific point of view? The majority opinion was that large-scale experiments (better: case studies) are feasible. They serve the purposes of observing trends rather than absolute facts and are needed for scaling up statistically significant observations from controlled experiments. In any engineering discipline

such large-scale experiments are a useful and necessary. The minority opinion was that large-scale experiments can never be sound from a pure scientific perspective and, therefore, are not helpful.

- What are the right kinds of goals for measurement and empirical studies to begin with? The majority opinion was that there is no general answer to this question. The only rule of thumb is to start with goals oriented towards understanding, to continue with goals oriented towards better management and prediction, and ultimately address goals aimed at change/improvement. The specific goals depend on the needs and characteristics of the organization at hand. The minority opinion was that one should always start with micro-level goals (e.g., understand a testing process) before moving towards macro-level goals (e.g., understand the entire development process).

- Is the Hawthorne effect crucial? Some viewed the impact of the Hawthorne effect as so crucial that they concluded measurement of people could/should not be performed at all; others viewed it as non-critical. Again, there seemed to be a difference of opinion depending on the purpose of empirical studies and measurement. The majority opinion was that the Hawthorne effect can be tamed using appropriate statistical designs/analyses.

- How can empirical studies and measurement be introduced in teaching curricula? The majority viewed it as essential to train students from the beginning in evaluating the effects of methods and tools. A minority suggested postponing the topic to advanced software engineering classes.

(D) **Important Topics for Future Work**:

Future work needs to emphasize both the development and assessment of better infrastructure technology (i.e., principles, methods, tools) for experimentation and measurement, the application of that infrastructure for empirically investigating existing software evolution aspects in order to build better models of the basic building blocks of our discipline, and the infusion of empirical studies ideas into the educational system (i.e., teaching and training) and the real world.

As far as the development and assessment of infrastructure technology for experimentation and measurement is concerned, there was widespread agreement that the most significant need for research exists in technologies for modeling software engineering aspects, feeding back empirical data to improve those models, and organizing models for reuse, as well as in the availability of more laboratory environments for empirical studies. Specific suggestions included:

- Better approaches for scaling up empirical results from small, controlled environments to large, real-world environments

- Infrastructure for software measurement, spanning metrics specification, collection, analysis, visualization and predictive guidance

- Alternate approaches for validating software models based on small sets of data points
- A taxonomy of study types depending on the purpose of study
- Formal approaches for hypothesis formulation
- Processes for creating baselines for different domains and environments
- An expanded notion of study context to include different processes and organizational structures in different business domains
- Investigation of the use of better graphical, animated methods for presenting and analyzing experience models
- Demonstration of practical benefits of measurement and risk (based on data)
- More real laboratories for conducting empirical research (Existing examples include the Software Engineering Laboratory at NASA's Goddard Space Flight Center and the Software Technology Transfer Initiative Kaiserslautern (STTI-KL) at the University of Kaiserslautern.)
- Communication networks enabling individual research groups and companies to cooperate in the sense that empirical studies are being replicated across environment boundaries to improve the believability of local findings or understand the impact of different contexts (An existing example is the International Software Engineering Research Network (ISERN) founded by Prof. Basili, USA, Prof. Cantone, Italy, Dr. Oivo, Finland, Prof. Rombach, Germany, Prof. Selby, USA, and Prof. Torii, Japan.)

As far as the empirical investigation of existing software evolution aspects are concerned, major efforts are needed in the following areas:

- Characterization of naturally occurring software artifacts
- Analysis of process-product relationships
- Measurement of (Derivation of measures for) the evolution of software
- Measurement of (Derivation of measures for) integration aspects of software
- Documentation and publishing of results and achievements in objective terms (see the SEL example under (A) and (B))
- Promotion of existing knowledge via software engineering handbooks (!! Don't be afraid of incompleteness at this stage !!)
- Development of social and economic models for software evolution

As far as the infusion of empirical studies ideas into education and practice are concerned, major efforts are needed in the following areas:

- Development of undergraduate software engineering courses stressing engineering aspects (e.g., problem solving using heuristics) supported by appropriate textbooks

- Development of graduate software engineering courses stressing the basic principles, methods and tools for measurement and empirical studies supported by appropriate textbooks

- Development of technology transfer programs based on measurement (i.e., first, quantitative baselines of the state of affairs need to be developed; second, changes for the purpose of improvement can be introduced)

In summary, the workshop served as an important event in continuing to strengthen empirical software engineering as a major subdiscipline of software engineering. The deep interactions and important accomplishments from the meeting documented in this proceedings have helped identify key issues in moving software engineering as a whole towards a true engineering discipline. By the end of the workshop, most of the attendees acknowledged that they feel part of a true community of empirically oriented software engineers. In order to foster that sense of community, the empirical software engineering community intends to hold a continuing series of conferences and meetings that build on this workshop. Furthermore, an e-mail list for communication and exchange of information among people interested in empirical software engineering research was suggested.
NOTE: Such an e-mail list "empirical-se@informatik.uni-kl.de" now exists! Requests to join should be sent to "empirical-se-request@informatik.uni-kl.de".

Acknowledgements

It would have been difficult to organize a workshop like this without help and financial support from a variety of organizations. First, we would like to thank all the workshop participants for their effort in submitting position papers and participating in the discussions. Next, the session chairs and keynoters deserve a special thanks for their contributions to the success of the workshop.

Clearly, an effort such as this could not have been successful without financial support. We would like to acknowledge and sincerely thank the IBFI for supporting lodging and providing excellent meeting facilities, and the University of Kaiserslautern for additional general purpose funds. Our special thanks go to Alfred Bröckers, Chris Lott, and Martin Verlage from the University of Kaiserslautern for having taken notes of all discussion sessions and to Mrs. Kilgore from the University of Kaiserslautern who did a splendid job in providing secretarial support and arranging the social events. Last but not least, we acknowledge the tremendous amount of work by Mr. Martin Verlage in editing these proceedings.

Kaiserslautern
May 1993

H. Dieter Rombach
Victor R. Basili
Richard W. Selby

Contents

Session 1: The Experimental Paradigm in Software Engineering

Session 2: Objectives and Context of Measurement/Experimentation

Session 3: Procedures and Mechanisms for Measurement/Experimentation

Session 4: [Measurement-Based] Modeling

Session 5: Packaging for Reuse / Reuse of Models

Session 6: Technology Transfer, Teaching and Training

Session 1:

The Experimental Paradigm in Software Engineering

Session Chair:	William W. Agresti
Keynote:	Victor R. Basili
Position Papers:	William W. Agresti Les Belady Norbert Fuchs Jochen Ludewig Walt Scacchi Walter F. Tichy

The Experimental Paradigm in Software Engineering

Victor R. Basili

Institute for Advanced Computer Studies
and
Department of Computer Science
University of Maryland

What is software and software engineering?

Software can be viewed as a part of a system solution that can be encoded to execute on a computer as a set of instructions; it includes all the associated documentation necessary to understand, transform and use that solution. Software engineering can be defined as the disciplined development and evolution of software systems based upon a set of principles, technologies, and processes.

We will concentrate on three primary characteristics of software and software engineering; its inherent complexity, the lack of well defined primitives or components of the discipline, and the fact that software is developed, not produced. This combination makes software something quite different than anything we have dealt with before.

One important characteristic about software is that it can be complex; complex to build and complex to understand. There are a variety of reasons for this. For example, we often choose software for a part of the solution, rather than hardware, because it is the part of the solution we least understand, or it is something new, or there is a requirement for change and evolution of the function or structure. In all of these cases complexity is introduced, the development becomes error prone, estimation is difficult, and there is a lack of understanding of implications of change.

However, the primary reason software is complex is probably the lack of models, especially tractable models of the product, process and any other forms of knowledge required to build or understand software solutions as well as the the interaction of these models. Software is not very visible, i.e., we do not have satisfactory models of the various aspects of the software, e.g., the functionality, the quality, the structure. In fact we do not even have intuitive models in many cases. This leaves us with a poor understanding of processes, requirements, and products.

Lastly, software is created via a development process, not a manufacturing process. This really means software is engineered.We have learned a great deal about quality manufacturing in the past few decades but we have not learned much about quality development/engineering.

So given the nature of this discipline, how does one begin to analyze the software product and process.