

Empirical Research in Software Engineering

P. van Keeken, 0264539
pkeeken@cs.uu.nl

STC Paper

Center for Software Technology,
Institute of Information and Computing Sciences,
Utrecht University,
P.O. Box 80089, 3508 TB,
Utrecht, The Netherlands.

29th August 2005

Abstract

To cope with the continuous increasing complexity and effort spend on development and maintenance of software, new tools and techniques are developed by the software engineering community. Often these tools and techniques are claimed to be an improvement. But these claims are usually not accompanied with a thorough explanation of the underlying theory of how and when to apply the tool or technique. Even basic evidence validating the claim, is often absent. This article addresses the need to validate one's claim. Two empirical research methods, especially applicable for evaluating new tools, techniques or practices in the context of software engineering are discussed. Both methods are relatively easy to apply once the obvious preconditions are met.

Keywords: *empirical research, experimentation, software engineering, experiment, case study, survey*

1 Introduction - Why do empirical research?

With the growing complexity of software products and services, the development process for creating software becomes harder as well. Typical questions that become more important to address are: "What language or tool can be used best, for this particular job?"; "What is the best practice to evaluate the quality of code, through inspection, unit testing or another method?"; "How scalable is my project?". Adequate answers and thorough understanding of used tools is essential to deal with the increasing software complexity and to stay ahead of competitors. Empirical research can be used to answer these questions, especially in situations where experience is not sufficiently available and common sense reasoning is misleading or not possible.

There are many tools and techniques available today which claim to be an improvement in certain phases of the development process. But often the understanding of why these methods work is not clear. When selecting a tool, it would obviously be favorable to have this understanding or at least some supporting evidence for the claimed effectiveness of the tool or technique. This understanding

and/or evidence can only be optimally derived by real life experimentation, as part of an empirical study.

Unfortunately we often see that research claims regarding the use of tools or techniques are not properly validated. This holds for computer science in general as well as software engineering in specific. Tichy [14] shows out of an analysis of over 400 research articles¹ that 40% lack sufficient empirical validation. This is over 50% for articles from the software engineering field². A later study by Zelkowitz and Wallace [18] shows that 20% of the analyzed papers lack necessary empirical evidence and another third which argues in terms of assertions. The evaluation through assertions is an (often weak) example-based research approach, favoring a proposed technology over alternatives.

The lack of empirical validation can also be the reason for the often heard complain that new technology is not used by practitioners in the software engineering field [19]. Ideally a community progresses through discovery and consensus. Researchers discover and implement (concepts of) new tools and techniques. They provide the (basic) understanding about how and why to use the tool or technique, supported by strong argumentation or empirical evidence. Practitioners from the industry use the proposed tools and techniques in real world cases. Resulting experience improves the understanding of the tool or technique, and increases the level of empirical evidence. New problems and research questions from the practitioners' experience, form new input for research, resulting in a cycle of improvement of the subject of study.

So the main reasons for doing empirical research is to *build an understanding* on how, why and when to apply certain development tools, techniques or methods. Besides this, empirical research can be used to *compare practices* and look for the best approach. Finally empirical research can *build a generally accepted model or theories* for the field of study.

The purpose of this article is to describe how

¹Analyzed publications are from *ACM Transactions on Computer Systems (TOCS)*, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *IEEE Transactions on Software Engineering (TSE)* and the *SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* in a period from 1991 till 1994.

²Based on publications from *IEEE TSE*

to do empirical research and to increase awareness among the tool developers about their possible contribution to empirical research within software engineering. When developing a tool, one could already consider ways to support future empirical evaluation of the tool. This would for certain increase the maturity of the software engineering field.

Section 2 discusses how to do empirical research in the software engineering field. We look at the process for doing empirical research as well as commonly used research methods (formal experiment, case study and survey). Following in Section 3 we will focus on how software engineering can benefit from two specific empirical research methods, benchmarking and monitoring. These methods are relatively easy to apply and offer ways to evaluate tools or techniques for effectiveness. In the final section we conclude this article with a short summary.

2 Doing empirical research

Empirical research is the process of collecting data regarding events happening in an experimental or real life setting with the purpose to conclude something about an earlier formulated hypothesis. Such a hypothesis can describe a claim regarding a model, theory or belief. Empirical research is a broad field which is practiced in many scientific areas. This section describes an introduction to this subject focusing on software engineering research. Other more in depth literature is available as well [8, 16].

In the first section we will discuss the objects which can be empirically studied. Section 2.2 discusses three research methods, *formal experiments*, *case studies* and *surveys*, which together form an often used empirical method classification. Section 2.3 addresses the basics behind formulating a claim in the form of a hypothesis, which is an elementary part of any study. In Section 2.4 we discuss an overall process for doing empirical research. Section 2.5 and 2.6 both discuss additional subjects regarding respectively research design principles and additional design techniques.

2.1 Objects of empirical research

Software engineering studies the collection of methods and techniques that apply on an engineering approach to the construction and support of software products. Software development includes typically activities like managing, budgeting, planning, modeling, analyzing, specifying, designing, implementing, testing and maintaining software products [4]. From this development process typically three categories can be an object for an empirical study:

- the separate development processes;
- the products from a process;
- the process resources.

The *processes* from a software development are arranged according to a development model, for instance the waterfall model³ or the extreme programming approach. Empirical research can be used to answer questions regarding the best practice to complete a process. A question could be for instance: "What is the influence of pair programming⁴ on the time to deliver a product?"

Every process produces one or more *products*, for instance a requirement specification, documentation or a software component. These products should adhere to the original intentions of the development project. This adherence can be evaluated by empirical research as well. Typical questions that empirical research can answer are: "How well are the initial requirements met in the resulting technical design or software code?" and "What are the characteristics of a product? How many variables, constants or other language constructs are used in the programming code?". A convenient characteristic of the products during the development process is that they often become more exact and structured. This makes it possible to analyze the product in an automated and quantitative way, making it easier to collect data for empirical research.

Processes are carried out by people, like developers and testers. Those people use tools, like compilers and database systems. These *resources* are vital

³In the waterfall model development flows steadily through each phase.

⁴Pair programming is a practice from the extreme programming development approach.

for the success of a process. Empirical research can be used to evaluate their performance. It is important to realize that this *human factor* is the reason that software engineering is not a typical hard science, like mathematics or physics. When the human factor is involved, evaluation through logic is not an option anymore, making empirical research an obvious choice. Typical questions that empirical research can answer in this category is: "Does tool *X* improve the productivity of a programmer or quality of a product?"

2.2 Typical research methods

There are several methods for doing empirically research. For these research methods several different classifications are available [5, 18]. In this section we discuss the classification as described by Freimut et al. [5], which is used often. Freimut et al. [5] divides research methods in *formal experiments*, *case studies* and *surveys*.

A *formal experiment* is a rigorously controlled investigation of an activity to prove or disprove a hypothesis using statistical techniques. A formal experiment typically involves the measuring of an effect of a treatment, in order to obtain a statistically significant result regarding a hypothesis about the treatment. A treatment in this context is the use of a tool or technique. When a sufficient amount of data from (a sufficient amount of) subject(s) is collected, one must either accept or refute the hypothesis.

A formal experiment offers a high level of control over the variables, making it a convenient method for analyzing causal relationship and testing proposed models. Downsides to this method is that when too much control is put on the variables, the study might not be completely representative for actual use of the tool or technique in a real world situation.

Example: A researcher is interested in how effective a new technique is regarding another commonly used testing technique. The treatment is the use of a testing technique. To evaluate and compare the testing techniques, he can design an experiment in which one group of testers uses the commonly used testing technique and the other group the new technique. Both groups use the same code. By applying the treatments and measuring effect on the variables of interest he can conclude some-

thing about the relative performance of each of the treatments.

A *case study* is a detailed investigation of a single case or a number of related cases. Such an investigation is performed in typical conditions, e.g. using representative (real life) projects. Often the data is qualitative of nature and is being collected with specific goals in mind. An advantage of this method is that costs are relatively low, since existing projects are used. Downside to this method is that the level of control over variables in the study is low compared to a formal experiment. This makes it harder to analyze underlining mechanisms and generalize results to other environments.

A *survey* is a broad investigation where information is collected from a group of people in a standardized form, usually a questionnaire or an interview. The answers to a (fixed) collection of questions are analyzed using statistical techniques. This method is retrospective since information is recorded after an event or situation has taken place. Because of this, the level of control over the variables is rather low, limiting the possibility to analyze the reasons behind an outcome of the questions. A survey is merely observant, not so much explanatory. Nonetheless a survey is a relative easy tool to analyze the response from large groups of subjects, unlike experiments and case studies, which are usually applied to smaller groups. Because of this, results are also easier to generalize.

2.3 Formulating a hypothesis

Common to all empirical studies is that one or more (formal) hypotheses are studied in context of observations from the real world. A hypothesis typically describes a claim regarding the effect of a treatment. A *treatment* in this context is the actual use of a tool or technique. Usually a study includes two hypotheses: a null hypothesis and an alternative hypothesis. A *null hypothesis* describes the situation where there is no significant advantage in using a treatment (optionally compared to other treatment(s)). The *alternative hypothesis* states that there is a significant difference. When the first can not hold under observations made, the alternative hypothesis has to be accepted.⁵

⁵This approach is often used for formal experiments, while case studies and surveys follow this pattern less strictly.

To be able to use a hypothesis as basis for measuring an effect, it must clearly describe the relation between the dependent and independent variables at stake. *Dependent variables* or *response variables* are factors that are expected to change as a result of applying a treatment. Examples of dependent variables are quality of code, number of defects per thousand lines of code or the productivity of programmer.

Independent variables are those factors that may influence the application of a treatment and thus indirectly the result of the research. These variable themselves are not likely to change because of the application of the treatment. Examples of independent variables are: the experience or background of a programmer or the number of hours of instruction in using a tool. Figure 1 shows the relation between the different variables when defining a hypothesis.

Unfortunately there can be more variables that influence the outcome of a treatment. These are called the *state variables* or *environment variables*. These variables often confound with the variables being measured, and are harder to manipulate and measure as a single factor. It is important to keep a level of control over this confounding effect as described in Section 2.5. An example of a state variable is provided in the example at the end of this section.

Variables are also called *factors*. This is mostly the case when the variable or combination of variables, express a special attribute of interest.

Example: When considering the example from Section 2.2, the null hypothesis could be: "There is no significant increase in the amount of discovered defects in the code when using the new treatment, compared to current commonly used testing technique, when the same amount of effort is spent on testing.". The alternative hypothesis would state: "There is a clear significant increase in the amount of discovered defects in the code when using the new technique, compared to current commonly used testing technique, when the same amount of effort is spent on testing.". The performance comparison of the two techniques can be based on the difference in the average number of detected defects. The dependent variable involved is the number of discovered defects. The independent variable(s) is the effort spent, which is the combination of the number of evaluated lines of code and the time spent testing code. Possible

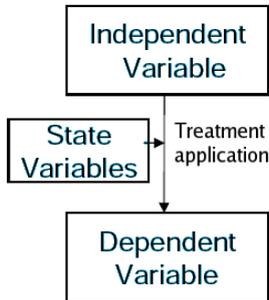


Figure 1: Hypothesis as relation of variables

confounding or state variables are the experience and background of the testers.

2.4 High-level model for empirical studies

Almost all empirical research methods, like the ones discussed in Section 2.2 follow a similar approach. From this a high-level model can be derived to function as a template for doing empirical research. Such a high-level process description can help designing and implementing empirical research as well as stimulate the acceptance of empirical research. This section presents a high-level model as described by Fenton [4] and Freimut et al. [5].

An empirical research typically consists out of the next six phases:

1. Definition
2. Design
3. Preparation
4. Execution
5. Analysis
6. Reporting

The *definition* phase is for setting the research goal(s) and stating the hypothesis. A research method is selected, e.g. formal experiment, case study or survey, and argumentation is provided, explaining why the selected research method is appropriate.

In the *design* phase the study goal and hypothesis are operationalized into a clear study plan.

The study plan describes in detail what needs to be measured and how it relates to the variables at stake. This phase is not always straightforward, especially if no generally accepted theory about the subject of study is available. The Goal Question Metric (GQM) approach is a technique that can assist in this process. The GQM approach will be discussed in Section 2.6.

In the *preparation* phase all necessary material is developed. This phase is also for making prototypes and doing test runs. This can prevent problems in the execution phase. In the *execution* phase the actual research is done by carefully following the research plan. In this phase, data is collected regarding the object of study.

The *analysis* phase is for processing the collected data, selecting and grouping according to the study plan. By applying (predefined) statistical tests, one can see whether the hypotheses hold.

In the final *reporting* phase the results and context of the study is summarized in such a way that others are able to understand and use the results.

Although the phases are listed sequentially, in practice the first three phases are often performed in an iterative fashion. When test runs or prototypes in the preparation phase fail or show an inefficiency, the study plan needs to be changed.

2.5 Design principles

When designing empirical research or selecting a research method, it is important to consider the issues of replication, local control and validity. These subjects concern the validity of the research results and help to improve the strength of the research conclusions.

Replication

Replication is the degree to which a research can be repeated under identical conditions (not necessary with the same subjects). This is desirable because others need to be able to verify your conclusions. In turn this will strengthen the believability of your research conclusion(s).

Another advantage of replication is that it can control the experimental error in the research. Every research can have an experimental error which is due to the fact that any experiment can have a

small bias in the actual observation or interpretation of a measurement. With replication this error can be estimated and controlled. In the same way replication can also assess the mean influence of the variables concerned in the study.

To ensure replication it is essential to limit confounding effects in the research process as much as possible. Confounding means that (unanticipated) variables are effecting the outcome of a measurement. For example, when measuring the effect of a tool, it is important to not only measure the hours of instruction a subject got, but also the experience of the subject. Both variables can influence the outcome and are therefore confounding. If it is not possible to limit the effect of confounding variables, then the effect can be counteracted by randomizing those variables in the studied group(s) of subjects.

Local Control

Local control describes the level of influence the researcher has over the variables of study. For instance when doing a case study or survey, the local control is relatively low, unlike during most formal experiments. High levels of local control provides ways to design research more effectively, but this comes with a price in effort.

Examples of possible design improvements are *blocking* and *factorization*. A *blocking experiment* divides the subjects in groups in such a way that all groups are relatively homogeneous. With this design one can anticipate the (expected) variation among the subjects. Conclusions can then be drawn for the complete group as well as the separate groups.

Example: When measuring the clarity of error messages from a compiler, one could measure how long it takes for subjects to solve particular errors in provided programming code. One could for instance block this experiment by looking at the groups with five or more years of experience and the group with less then five years of experience.

A *factorial design* takes this concept a step further. With this approach a treatment is applied to each value of a factor of interest. When for example factors like experience (assumed to be a boolean variable) and gender is considered, the study design will include four trails for each combination of the mentioned factors.

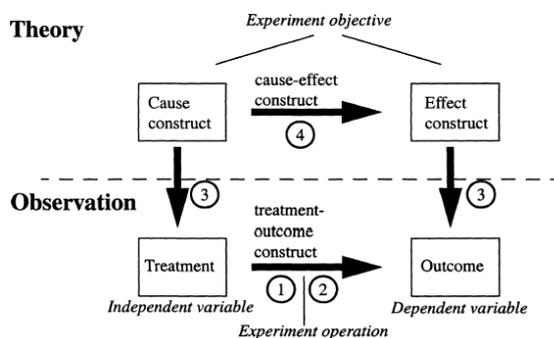


Figure 2: Aspects to validity (Freimut et al. [5])

Other more sophisticated techniques can be found in the literature [5, 8, 16].

Validity

Often the results from empirical studies are criticized for how conclusions are drawn from observed phenomena. Operationalization of variables and interpretation of data are typical cases for which multiple solutions are possible. This holds especially when abstract notions, like experience and quality are concerned. To make conclusions drawn from research more credible, one has to address the next four aspects to validity:

1. Conclusion validity
2. Internal validity
3. Construct validity
4. External validity

Typically what you do when applying empirical research, is check whether observations made in the real world match with a model or idea in the theoretical world as depicted in Figure 2. Above the dashed line represents the world of theory in which we model how a cause construct is related to an effect construct. Under the line represents the observable world in which the actual effect of a treatment takes places, influencing the variables at stake. To validate a model or idea, we have to operationalize the constructs of this model in the real world and try to observe if the consequences of the model are manifested there as well. Each of

the four validity aspects focus on a specific part of this validation and translation process.

Conclusion validity addresses the need to have a relationship between the treatment and the outcome. A study is not very credible if this relationship is weak or absent.

Internal validity raises the question that assuming there is a relationship between the treatment and the outcome, is this relationship a causal one? It could be the case that other factors are (also) responsible for the outcome. Missing or neglecting certain (confounding) variables can form a threat to the acceptance of the study result.

Construct validity addresses the issue whether we can correctly claim that a treatment reflects our cause construct and that our outcome reflects our effect construct. This is all about correct operationalization of the variables in the study. Often different operationalizations are possible, especially for abstract notions like experience or quality of code. The study should provide solid argumentation about decisions made regarding the operationalization.

External validity addresses the issue whether study results are generalizable outside the actual context in which the experiment was run. Assuming that the other validity aspects are addressed adequately, it is interesting to see whether the study results also hold for other cases. This depends on issues regarding the representativity of subjects and the environment settings of the experiment. When there is too much difference between the context of the study and the generalized case(s), the generalization may be dubious.

These four aspects of validity are cumulative, meaning that addressing a certain aspect makes only sense if all preceding aspects have been addressed adequately. When all aspects are sufficiently dealt with, the study results are more credible, making it more likely that the results are accepted by an audience.

2.6 Additional research techniques

When designing an empirical study, the task to specify and operationalize study goals and hypothesis into representative metrics is often experienced as being hard. Additional research techniques, like the Goal Question Metric approach, are developed to assist researchers in this.

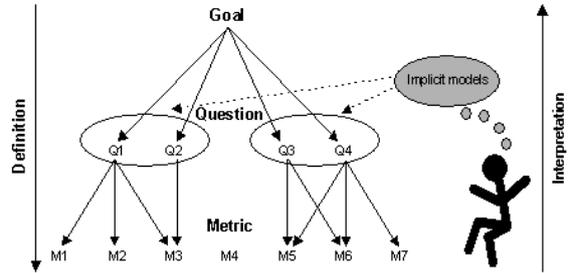


Figure 3: GQM approach (Basili et al. [2])

Goal: Explore the use of language X in terms of program characteristics
Question: What is the complexity of written programs?
Metric: lines of written code, language constructs used in total

Figure 4: GQM example

The typical problem with operationalizing hypotheses and study goals is that often no clear, generally accepted theory is available to provide a basis for undisputed metrics design. The *Goal Question Metric approach (GQM)* is a metric program which overcomes this problem by providing a framework for the operationalization process in a stepwise fashion. The GQM approach presents a top-down framework, involving three steps or levels in achieving the minimal level of detail necessary:

1. **Conceptual level (Goal):** List the major goals.
2. **Operational level (Question):** State for each goal the relevant questions.
3. **Quantitative level (Metric):** Describe the (combination of) metric(s) answering the questions.

The results from these three levels form a graph of (inter-)related goals, questions and metrics as shown in Figure 3. To fully specify the GQM graph additional information about the exact implementation of each metric and their mutual relation are required.

Example: To analyze the use of a language one can be interested in the characteristics of the

programs written in that language (a goal). A questions related to this could be: "What is the complexity of programs written in this language?". Metrics that can answer this question are: number of (non-blank) lines of written code, number of language constructs used in total. Combining the two metrics can yield a complexity factor, describing the number of language constructs per line. Figure 4 shows this example in the form of a table.

A convenient result of the GQM approach is that the questions derived form an implicit model. This model can be used as a starting point for a more formal theory. The GQM approach also provides assistance during the interpretation phase of a research. When data for the metrics are collected, they can be used to answer the stated questions. The answered questions can be used in turn to check whether the original study goals are achieved. So the interpretation phase can be guided by reversing the GQM steps.

Other research techniques assisting the research process are *Quality Improvement Paradigm* (QIP) [9] and *Experience Factory* (EF) [3]. QIP is a broad research paradigm focusing on improving the overall research process. EF takes this a step further with a focus on improving the reporting phase and the storing of study results for future use.

3 Empirical research methods for software engineering

An often heard complaint is that most empirical research methods take too much time and effort. Nonetheless many authors [14, 18, 12] argue that solid empirical validation of claims is vital for the progress and maturity in a field like software engineering. This section discusses benchmarking and monitoring, two empirical research methods which are relatively easy to apply within the context of software engineering.

3.1 Benchmarking

Benchmarking is an empirical research method in which one or more representative or standardized tests, so called benchmarks are run in trials to compare the characteristics of an object of study. In computer science benchmarks are for instance used

to compare the performance of CPU's, database management systems, as well as compilers. In this section we limit ourself to benchmarking with a primary focus on scientific research. Benchmarks designed for business or marketing purposes are optimized for different goals and are for this reason often criticized for being unrepresentative.

3.1.1 Advantages & disadvantages

Benchmarking is, under the precondition that standardized or generally accepted tests are available, a relatively easy way to evaluate a claim, tool or technique [14, 12]. Benchmarking can quickly eliminate unpromising approaches and exaggerated claims, stimulating the consensus needed by a scientific field to make progress.

The difficult, but essential part in benchmarking is achieving acceptance and standardization of benchmarks. Without this, benchmarking is often not creditable. Achieving acceptance and standardization requires researchers to examine the understanding of their field and focus on what the key issues are. Researchers then need to agree on one or more operationalizations or benchmarks for each issue.

Since the development of representative and generally accepted benchmarks can be an technical and social challenge, Sim et al. [12] recommend to make a community effort, sharing the burden over the different parties. This would also increase the communication and collaboration among the different research groups significantly, leading to a stronger consensus in the field, even before any benchmark is applied. Once benchmarks have been developed, they can be used multiple times at low costs, yielding generally accepted results.

3.1.2 Process of benchmarking

The general process for benchmarking does not differ much from the overall process of doing empirical research as presented in Section 2.4. Special care has to be considered when the benchmark tests are developed (by a community). Developing benchmarks puts more pressure on the definition and design phase of a study. Achieving consensus is a technical as well as social challenge.

In the case a community takes the effort in designing benchmarks, Sim et al. [12] recommend to

have small number of leaders in the field to take the lead. The resulting design decisions for the produced benchmark needs to be supported by laboratory work and good argumentation. Obviously there should also be sufficient possibility for the community to participate and give feedback, during the benchmark-specification phase. Once the benchmarks are established, they can be used multiple times, under the condition that the benchmarks are maintained and updated to prevent overfitting.

3.1.3 Application

Benchmarking is used throughout the computer science field for comparing hardware and software, like programming languages [1] and applications in general (e.g. web-servers, graphical applications) [13]. Sim et al. [12] advocate that benchmarking should also be used in the software engineering field to assess tools, techniques and development practices. Many unvalidated claims are made, regarding the performance of a tool, productivity of a technique, quality of a program resulting from a tool and ease of use, without a clear consensus on what these terms really express. Making these terms measurable and comparable would provide a base for easier and more effective research. Sim et al. [12] point out that specifically fields like requirement engineering, model checking and software evolution, are at a stage where benchmarking can be used well.

3.2 Monitoring

Monitoring is an empirical research method for studying how tools or techniques are used in a certain period. The subjects produce (partial) artifacts, like program code, documentation or design descriptions, which are structurally stored together with extra environmental information, for evaluation. This storing of artifacts can be done on pre-set times or when certain events occur. Although it is not really a requirement, logging is preferably done in an automated and centralized fashion. This makes the collection and evaluation of data a lot easier and more reliable.

Monitoring can for instance be used to evaluate a programming tool, like a compiler. In an experimental setting, subjects, with different backgrounds can be asked to write programs for pro-

vided exercises. The compiler is instrumented in such a way that with every time a subject compiles, the source together with compilation results and compiler parameters is sent to a central server. The collected artifacts, or loggings, show the evolution of the programming code of the subjects over time. Analysis of these data could show the difference in programming style among the different backgrounds of the subject as well as factual information about how the subjects program, e.g. which language constructs are used often.

Monitoring can be done *in vitro* (like in the previous example) as well as *in vivo*. *In vivo* means that subjects working on real life projects are being monitored.

3.2.1 Advantages & disadvantages

As a research method, monitoring is relatively easy to implement. Of course studied tools or techniques should provide a way to store produced artifacts, preferably to a central logging server. This is merely a technical issue and even a logical extension to tool. A developer interested in the empirical validation of his or her tool or technique should also provide ways for others or him or her-self to evaluate it.

The major advantage of this research approach is that produced artifacts are logged over time, providing the possibility to study the change of a wide variety of metrics over time. If the monitoring occurs with a high frequency, collected data from monitoring can be used to replay the birth and evolution of an artifact matching closely to the original recorded process.

A slight disadvantage of monitoring is that interpreting and analyzing the data can be tedious. More advanced techniques, like data-mining might be necessary.

3.2.2 Process of monitoring

Like any other empirical method, monitoring follows the general process of doing empirical research from Section 2.4. With monitoring it is quite tempting to start recording the use of a tool before you even decided which characteristic(s) to study. Problems might occur when certain characteristics are not sufficiently present or measurable. A good research design optionally used in combination with

techniques like GQM can prevent most of these problems. Another approach could be to decide what to study based on collected data, accepting the limitations and occurring problems. This places the burden more on the evaluation phase than on the design phase.

3.2.3 Application

Monitoring is applicable for evaluating the use of a tool, technique and practice. Monitoring is possible for any system that can store produced artifacts and optional meta information. Examples are versioning systems and tools, like compilers and IDE's.

In literature not much examples of the application of monitoring are available. Beside some old researches [10, 11, 17], Jadud [7] uses an automated form of monitoring for the evaluation of BlueJ, a pedagogic programming environment, for learning the programming language Java. BlueJ is instrumented to send the source code and relevant meta data of a novice programmer to a central server at compile time. The focus of the study is to analyze programming behavior of novice programmers and how this behavior can be influenced.

Similarly, monitoring is used to evaluate the Helium compiler [15, 6]. Helium is a compiler especially designed for learning Haskell with a special focus on providing informative error messages. The compiler is equipped with a logging facility, which collected over 60,000 loggings (source code and compiler result) from three introductory courses on functional programming, currently being analyzed by the author. The loggings provide a vast amount of information about the programming style of novice functional programmers, and the effectiveness of the compiler. Further research of the data is necessary and can be used to improve the compiler and increase the understanding on how student learn the functional programming paradigm.

4 Conclusion

Several researchers [14, 18] advocate having more empirical research in the field of software engineering and provide assistance in applying research methods [5, 16, 8]. Applying empirical research and validating claims is an ongoing process, nec-

essary for the maturity of the software engineering field. This article is written with the purpose to provide an overview of how to do empirical research in the software engineering field. Additionally two research methods, especially applicable in the context of software engineering have been discussed. Both are relatively easy to apply if certain preconditions are met. For benchmarking the precondition is the availability of generally accepted benchmarks. Monitoring requires tools to provide a way to store the deliverable of a tool, preferably in an automated and centralized fashion.

Software engineering, like science in general, advances through the conception of new ideas and consensus on these ideas. Empirical research is important since it helps to achieve consensus when argumentation through logic is not enough or not possible. The consensus helps the community to focus on the problems that really matter.

It is of importance for the software engineering community to realize this. If we want our community to mature, then validation is a necessity. When developing tools, techniques or practices, the developer has the responsibility to evaluate it or at least provide ways for others to evaluate the technology. With this, the use of a new tool, technique or practice becomes a more rational choice instead of a leap of faith. With the increasing dependence of an industry and society on software engineering products, and increasing need to cut costs and promote predictability, research is required to provide solid validating research more than before.

References

- [1] D. Bagley, A. Calpini, and B. Fulgham. The computer language shootout benchmarks. <http://shootout.alioth.debian.org>.
- [2] V. Basili, G. Caldiera, and D. Rombach. *Encyclopedia of software engineering*, chapter The Goal Question Metric Approach. Wiley-Interscience, New York, NY, USA, 1994.
- [3] V. Basili, G. Caldiera, and D. Rombach. *Encyclopedia of software engineering*, chapter The Experience Factory. Wiley-Interscience, New York, NY, USA, 1994.

- [4] N. E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., London, UK, UK, 1991.
- [5] B. Freimut, T. Punter, S. Biffi, and M. Ciolkowski. State-of-the-art in empirical studies. Technical report, 2002.
- [6] B. Heeren, D. Leijen, and A. van IJzendoorn. Helium, for learning Haskell. In *ACM Sigplan 2003 Haskell Workshop*, pages 62 – 71, New York, 2003. ACM Press.
- [7] M. C. Jadud. A first look at novice compilation behavior using BlueJ. 2004.
- [8] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic, 2001.
- [9] A. Kinnula. *Software process engineering systems: models and industry cases*, chapter Quality Improvement Paradigm. University of Oulu, 2001.
- [10] C. R. Litecky and G. B. Davis. A study of errors, error-proneness, and error diagnosis in Cobol. *Commun. ACM*, 19(1):33–38, 1976.
- [11] P. G. Moulton and M. E. Muller. Ditrans - a compiler emphasizing diagnostics. *Commun. ACM*, 10(1):45–52, 1967.
- [12] S. E. Sim, S. Easterbrook, and R. C. Holt. Using benchmarking to advance research: a challenge to software engineering. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 74–83, Washington, DC, USA, 2003. IEEE Computer Society.
- [13] Standard Performance Evaluation Corporation. <http://www.spec.org>.
- [14] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz. Experimental evaluation in computer science: a quantitative study. *J. Syst. Softw.*, 28(1):9–18, 1995.
- [15] A. van IJzendoorn, D. Leijen, and B. Heeren. The Helium compiler. <http://www.cs.uu.nl/helium>.
- [16] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic, 2000.
- [17] M. V. Zelkowitz. Automatic program analysis and evaluation. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 158–163. IEEE Computer Society Press, 1976.
- [18] M. V. Zelkowitz and D. R. Wallace. Experimental validation in software engineering. 1997.
- [19] M. V. Zelkowitz, D. R. Wallace, and D. W. Binkley. Experimental validation of new software technology. pages 229–263, 2003.