ESEML — Empirical Software Engineering Modeling Language

Bruno Cartaxo, Ítalo Costa, Dhiego Abrantes, André Santos, Sérgio Soares, Vinicius Garcia
Informatics Center - Federal University of Pernambuco
Av. Professor Luís Freire, s/n, Cidade Universitária
CEP 50740-540, Recife-PE, Brazil
bfsc,imac,daom,alms,scbs,vcg@cin.ufpe.br

ABSTRACT

New processes, patterns, structures, tools, languages, and practices are being proposed for software development, but technology transfer is hard to achieve. One of the objectives of empirical studies is easing technology transfer from academy to industry. On the other hand, there are a number of issues that hinder the application of empirical studies, more specifically, controlled experiments. This paper defines a visual DSL for modeling controlled experiments supporting researchers that are not experts in such study. By using the language, the researcher is guided to define the elements of an experimental plan and connections, which is automatically generated, resulting a complete document of experimental plan. The proposed environment assists the definition of controlled experiments, increasing empirical evaluation of the proposed technologies. More specifically, the current version of our proposal generates the experimental plan from the experiment model defined using the DSL.

1. INTRODUCTION

Researches in Software Engineering normally proposes new processes, patterns, structures, tools, languages, or practices for software development, typically in order to increase productivity and quality of products and services. However, technology transfer is hard to achieve and empirical studies tries to help in moving technology from academy to industry [15, 12, 2].

A great part of these researches that propose new methods fail to present empirical evaluation. Only through these evaluations it is possible to establish whether and in which context the proposed technique is efficient, effective, and can be applied [6, 15, 11, 10], therefore easing technology transfer. According to Sjøberg et al [11], among 5,453 scientific articles published in 12 of the main journals of software engineering between 1993 and 2002, only 1.9% had a controlled experiment involved.

In order to make an empirical evaluation, it is necessary to point out empirical methods and techniques. Also, it is possible to adapt these in the context of software engineering. Empirical Software Engineering includes several types of studies, such as Surveys [6] Case studies [15], Secondary studies [7] and Controlled experiments [15, 10].

Each one of these studies has its own characteristics and should be used in specific contexts. In this paper we will focus on controlled experiments, which is the most controlled technique and is also commonly used within a very specific context [15, 10]. Therefore, such controlling and specificity use to compromise experiments generalization. On the other hand, experiments are a good technique to validate the effects of each small change in the observed environment, not requiring real world conditions, which are often difficult to obtain in academic research [6, 15]. Thus, a controlled experiment can be a good method to pinpoint if a technique, method or process in software engineering does what it claims it does. Additionally, it creates a setting for even more applied and less controlled empirical research conditions, such as surveys and case studies [6].

To conduct a controlled experiment it is necessary to bring together a wide range of skills that often create a barrier for adopting the technique. The conduction of an experiment requires skills in the matters we want to check (software engineering in this case), experience with technical terminology, statistics, as well as expertise in designing the experiment. Above all, the controlled experiment is a very particular knowledge domain.

In order to limit, define, and accelerate the development of solutions in a specific field, the concept of domain-specific languages (DSLs) has been created. These languages tend to be very expressive and natural, even for people without previous programming knowledge. DSLs are largely used to express problems within a specific domain in a natural and fluent way.

Since DSLs are good alternatives to model solutions in a specific domain, adding the fact that controlled experiments have their own domain vocabulary, the definition of a DSL is adequate to the problem of modeling and conduction of controlled experiments in software engineering. It mitigates social barriers between stakeholders in the field we want to validate, the team of statisticians, the experiments designers, and the domain expert.

In the face of all those previously explained reasons, this paper presents the definition and development of a visual DSL for modeling controlled experiments in software engineering. Our solution generates the experimental plan document from an instantiation of our domain model. By using the language, the researcher is guided to define the elements of an experimental plan and connections, which is automatically generated, resulting a complete document of experimental plan. This DSL is the kickoff for a major research initiative that is the development of a platform for conducting empirical studies in software engineering. Such platform and computer-aided systems are necessary tools to increase the volume and quality of empirical studies in software engineering [14].

This paper is organized as follows. Section 2 addresses the main concepts of empirical software engineering and controlled experiments. The approach we used to propose a DSL is discussed in Section 3. Section 4 presents the DSL and the workbench we defined to model controlled experiments. Section 5 presents the related work. Conclusions are discussed in Section 6 and future work in Section 7.

2. EMPIRICAL SOFTWARE ENGINEERING AND CONTROLLED EXPERIMENTS

The use of scientific method involves the comparison of theories and techniques with reality in order to verify if those are valid enough to be taken forward. A major problem in the current scenario is that software engineering has used time as a parameter for validity of its theories in detriment of confrontation with reality through experimentation. The value of an idea is judged by whether or not people use the idea. If many people use the idea, for a long time, it seems to be certain [6].

Since it is critical to use empirical methods to evaluate theories for software engineering, it is also necessary to master technologies for conducting the studies, as well as expertise to overcome the problems inherent to them. Some of the impediments to the systematization of empiricism in software engineering are: lack of familiarity with the scientific method, lack of experience in analyzing the data, and lack of comprehension to deal with the burden related to the human factors in software development.

In order to remove or at least reduce the presented obstacles, some actions are necessary. If software developers are not familiar with the scientific method, it is interesting to present them with positive results in other disciplines such as engineering and medicine. This would be a way of showing that the cycle of hypothesis confrontation with the reality is of great value to have a better understanding of the construction of a software. In the case of developers lacking the experience to analyze the data, just show them that the necessary statistical and mathematical foundation is part of their own education as engineers. If the experiment examples are more common in other areas such as agronomy and medicine, a good solution is to increase the volume of empirical studies in software engineering to create a body of knowledge and appropriate terminology. Finally, if human factors are strong confounders in the development of software, we can make use of knowledge from human sciences in conducting empirical studies in order to control and minimize such biases [6].

Once the obstacles are removed or controlled, it is possible to say that we are ready to conduct an empirical study. So the next step is to select which type of study is best suited to our research. For now, it is important to know the categories of empirical studies. Thus, we have two types of classifications for empirical studies: according to the nature of the data and regarding the process of conducting the study. Concerning the nature of data, there are qualitative and quantitative studies. On the strategy of conduction, there are surveys, case studies, secondary studies, and controlled experiments.

Quantitative studies investigate the relationship between the numerical variables being examined. Qualitative research, on the other hand, tries to understand the objects in their natural state without having to establish numerical relationships [6].

In addition to the types of studies by the nature of the data, there are studies on the strategy of conduction. Surveys, for example, are investigations made in retrospect, typically through interviews and questionnaires, when some method or tool is already being used, seeking to interpret the results to generate descriptive and explanatory conclusions. Case Studies, on the other hand, are widely used to monitor projects and activities during its execution, collecting data and doing statistical analysis without much control over the observed environment. Secondary and tertiary studies, such as systematic reviews and mapping studies are made by gathering up the empirical research done in a particular area of knowledge aiming to organize and take nontrivial conclusions about the subject in question [7]. Finally, controlled experiments are usually careful examinations made in a laboratory with a high degree of control. These studies are intended to manipulate variables and to observe their effects, to perform statistical analysis and draw conclusions about the impacts of variables in their contexts [15].

The experimentation process is done at different levels by various groups within the software engineering community, which means that each group must take a specific responsibility in verifying knowledge. The first link in the chain is the researcher, which proposes theories, methods, tools, and techniques to address open problems. This level of testing is most commonly performed in laboratories at highly controlled environments, in contrast to the studies in the real world. However, this first level of research is extremely necessary to answer preliminary questions such as: Does the methodology have some effect on the team productivity? Does the programming paradigm X have an impact on the readability and maintainability of the code? These are some of the recurring questions in software engineering [6].

In order to obtain reliable results when conducting a controlled experiment, it is important to follow a well defined systematic process. When we say process, we should understand it as guide to support some activity from the very beginning to its own end. Within this perspective we can characterize the process of conducting a controlled experiment with the following phases: definition, planning, operation, analysis, and presentation [15].

During the definition phase, the researcher is concerned with the experiment setup, in terms of problems and objectives. Then, in the planning phase, the researcher must design the experiment in order to specify variables, treatments and threats to validity. Later, during the operation, the experiment is actually executed according to a plan and the data originated by the execution collected. Once the data was collected, the next step is to perform an exploratory analysis of these data, which is to perform a statistical analysis and further hypothesis tests of these. Finally, the researcher must present the analysis and further conclusions for the experiment [15].

In addition, it is still necessary for the researcher to know the terminology inherent to the field of experimentation. With these terms in mind, the researcher will be able to map each concept to the experiment to be performed. Some key concepts are: experimental unit, experiment participants, response variables, parameters, factors, levels, block variables, validity, and others [15, 6].

The experimental unit or experimental object is the entity who "suffers" the execution of the experiment and can only be well defined in accordance with the objectives of the experiment. Patients are typically the units of a medical experiment. A software project or a phase of a development process can be the experimental unit of a software engineering experiment.

Participants are those individuals who apply, in the experimental unit, the techniques and methods that are being tested. In some branches of knowledge, the participant exercises little or no influence on the outcome of the experiment. However, it is known that in software engineering participants are key players influencing both positively and negatively the results of the experiment.

Response variables or dependent variables are typically the outputs (or results) of an experiment. In case of controlled experiments, these variables are typically quantitative. The execution time of an algorithm may be a response variable within the context of an experiment that evaluates performance on multiple implementations of an algorithm.

Parameters are characteristics fixed at a given value, so they do not vary throughout the experiment execution's process. In order to compare the quality of code to implement a solution using and not using design patterns, we could set as a parameter the programming language to ensure that the improvement in execution time would be inherent to the use of the pattern and the experiment would be valid within the scope of the language set as parameter.

Factors are features that intentionally vary during executions of an experiment. Thus, the understanding of their effects on response variables are the goals of an experiment. Levels (or alternatives) are the values that a factor can assume. If an experiment is conducted to assess which language is more efficient to solve a given problem, one factor could be the programming language, and the alternatives could be Java and C++.

Blocking variables are undesired variations that occur during the experiment and, thus, cannot be fixed as parameters. However, blocking variables influence the response variables and in many cases invalidate experiments. An experiment focused on assessing the quality of a code written in some particular language can be strongly influenced by the experience of the developer. Therefore, experience may be a blocking variable [6].

Another key concept in the scope of experiments is how valid are their results. A misguided planning, execution or data collection may completely invalidate the results of an experiment. For that reason, it is important to ensure that all aspects that guarantee the validity of an experiment are performed correctly. It is common to define the existence of four types of validity: conclusion, internal, construction and external [15].

When the basics are known, as well as the process of experimentation, we are already able to conduct an experiment. Recalling those after conducting a controlled experiment is very important to replicate it. Replication of an experiment consists in run it again, preferably by another group of researchers, to ensure the validity of the results and adjust possible mistakes in planning, analysis, and conclusions.

3. DOMAIN-SPECIFIC LANGUAGES

According to Fowler [4], a Domain-specific language is a computer programming language and, like any other language, is as a way of manipulating abstractions. Domain-specific languages are languages of limited expressiveness focused on a particular domain.

DSLs are composed by two key concepts: Like any other computer programming language, it should run on a computer attending to a purpose; Every language should have a sense of fluency where the expressiveness comes not just from individual expressions but also from the way they can be composed together.

A general-purpose programming language provides lots of capabilities: supporting varied data, control, and abstraction structures. All of this is useful, but makes it harder to learn and use. A DSL, on the other hand, provides a bare minimum of features needed to support its domain. You cannot build an entire software system in a DSL; rather, you use a DSL for one particular aspect of a system.

Fowler states that a limited language is only useful if it has a clear focus on a small domain. The domain focus is what makes a limited language worthwhile.

There are three different kinds of DSLs: Internal DSLs, External DSLs and DSL Workbenches. A DSL within a general purpose programming language is an Internal DSL (i.e.: Regular Expression in some languages). A DSL that runs separate from the language of the application it works with is an External DSL (i.e.: XML). External DSLs will usually be parsed by a code in the host application using text parsing techniques. When you have a specialized IDE for defining and building DSLs, used not just to determine the structure of a DSL but also as a custom editing environment for people to write DSL scripts, you have a DSL Workbench.

3.1 DSL Lifecycle

Fowler states that a common alternative is firstly define the DSL. He mentions that you should begin defining some scenarios and the way you would like the DSL to look. Additionally, he emphasizes the presence of a domain expert during the construction of the language: "This is a good first step to using the DSL as a communication medium".

When you sit down with some people who understand the customers' needs, you come up with a set of controlled behaviors, either based on what people wanted in the past, or on something you think they will desire. That is the input you need to create a way to write it in a DSL form.

For every interaction in this workflow, you will modify the DSL to support new capabilities. By the end of the exercise, you will have worked through a reasonable sample of cases and will have a pseudo-DSL description of each of them. Once you have a representative set of pseudo-DSLs, i.e. a representative set of features for a given domain, you can start implementing them.

3.2 DSL Workbench

In the present study, we constructed a DSL Workbench to improve productivity in the experiment execution, minimize flaws in the experiment modeling, and to mitigate the social barriers in the understanding and sharing of knowledge among stakeholders in controlled experiments. The first two goals can be achieved by systematizing the process of controlled experiments. The third goal, which is our main focus, can be achieved by a DSL Workbench.

A DSL workbench is an environment designed to help people create new DSLs, together with high-quality tooling required to use those DSLs effectively [4]. This visualization representation is similar to the DSL itself in that it allows a human to understand the model. The visualization differs from the source in that it is not editable, but on the other hand, it can do something an editable form cannot, such as a render diagrams. Communication with the customers and users is the most common source of project failure in software development. By providing a clear yet precise language to deal with domains, a DSL can help improve this communication. Additionally, as Fowler states: "I do think DSLs can improve communication. It is not that domain experts will write the DSLs themselves; but they can read them and thus understanding what the system thinks it is doing. By being able to read DSL code, domain experts can spot mistakes. The biggest gain from using a DSL in this way comes when domain experts start reading it. Involving domain experts in a DSL is very similar to involving domain experts in building a model. I have often found great benefit by building a model together with domain experts; constructing a Ubiquitous Language deepens the communication between software developers and domain experts" [4].

4. ESEML

In order to develop the DSL for the present study, our work started based on the lifecycle defined by Fowler to create Domain Specific Languages [4]. Instead of starting by the Domain Specific Language itself, a preliminary informal review of models, ontologies [5, 9] and other formal representations for controlled experiments has been done. Our objective

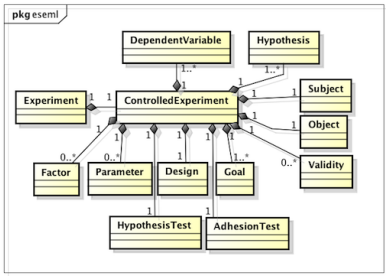

Figure 1: Part of ESEML domain model.

was to provide a rationale of concepts and their relations for controlled experiments. The idea was to define our Domain Model and then proceed with the DSL itself.

Based on concepts, entities, and relations raised by relevant studies, we came to a leverage for our model. Due to space constraints, part of the ESEML domain model is depicted in Figure 1[1]. The starting point for the implementation of our DSL was creating, through code generation, the experimental plan.

We chose the Microsoft DSL Tools to create the DSL. We first need to define a domain model, therefore we proceed with a preliminary definition of the model and implement the entire DSL later. After the conception of the model, the next step was defining the visual representation of the DSL. Just like the lifecycle mentioned by Fowler, a couple of interviews with experienced researchers and further validation with the proposed visual representation were done. Our objective was to follow the first step mentioned by Fowler, to start using the DSL as a communication medium among stakeholders.

These interviews led to the conception of new features and ideas for the DSL. Anyhow, we chose not to add new features, avoiding to exceed time and scope limitations we had due to the nature of this project, making us to stick with our preliminary domain model. Additionally, no semantic validators were implemented due to the same restrictions.

By using the ESEML Workbench, the user can instantiate his own experiment from a pre-defined model for controlled experiments. The whole idea of our domain model was to summarize, through a rationale of models in software experimentation, the entities involved in a controlled experiment. Part of the ESEML workbench is presented in Figure 2. The experiment elements that the researcher will use to define his experiment are on the objects palette in the left-hand side. Each element corresponds to one specific experiment element in the domain model (see Section 2). On the right-hand side you can see the diagram with the domain elements of the experiment being modeled.

At the elements palette the user can define both null and alternative hypothesis, factors, related treatments, parameters, dependent variables, subjects, experimental units, adherence tests, hypothesis test, define threats and valida-

[1]The complete domain model is available at http://bit.ly/Sr2mqu.

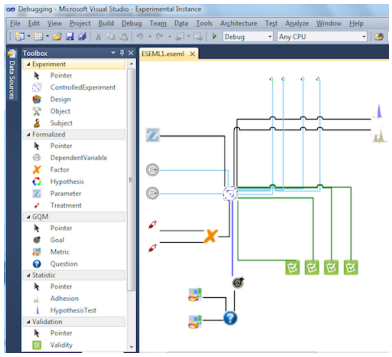

Figure 2: Part of ESEML Workbench.

tions (internal, external, conclusion and construction), and a Goal-Question-Metrics structure for the experiment. The complete list of elements is presented in Table 1. All those elements of an experiment were explained in Section 2.

Our objective was to define each element and its relationships in a model reflecting a controlled experiment. Thus, in a GQM structure, for example, a controlled experiment is linked to a goal that has an embedded relationship with questions, and each question have a set of metrics.

Elements within the ESEML model have properties that can be defined during the construction of the experiment model. The final graph that represents the experiment is going to be syntactically and semantically validated. Thus, in order to minimize errors during the configuration of the experiment, the ESEML will be capable of prompting the user for possible experiment threats of validity, also it can identify confounding factors or even show that questions are not based on any metrics.

The model represents the experiment. Thus, it might work as an effective communication channel between different stakeholders in the experimentation process mitigating doubts and conflicts in the experiment configuration.

Finally, from the defined model, there is information to generate the experimental plan, forms to collect data, scripts to perform statistical analysis of the defined metrics, and other artifacts related to your experiment. Thus, ESEML is intended to ease the burden in the process of experimenting, from planning through data collection and analysis, those can be achieved through transformations of the domain model instance. We believe that these transformations might improve productivity of experimentation in software engineering.

We focused our work on implementing a language that enables researcher to represent all the data needed to allow the automated generation of an experimental plan. Microsoft DSL Tools Framework [3] uses T4 transformation templates to generate code. A set of conditions and rules has been defined to iterate through our Domain model structure and transform the model into a PDF document. The document is intended to contain all demands to the experimental plan. After defining your controlled experiment through the visual representation of our DSL, you are just one click away from generating the experimental plan. Due to space contraints, an example of the experimental plan generated by our DSL using a real experiment definition is available at http://bit.ly/RusZYm.

5. RELATED WORK

Initially, an informal review of the literature looking for similar studies has been done. No studies involving the definition of a DSL and a Language Workbench for modeling empirical studies in software engineering were found. It was possible to find some tools focused on supporting the conduction of empirical studies in software engineering, which are presented in this section.

Torii et al [13] presented a Computer-Aided Empirical Software Engineering (CAESE) framework and Ginger2, a partial implementation of this framework, that aims to support all phases of an in vitro study (controlled experiment) in software engineering. However, the experiment design phase support was not implemented in the Ginger2 as reported in [13] and this phase is exactly what our work focus. Punter et al [8] show the advantages of online surveys making use of some web survey tools management systems like Globalpark iSurvey, or eSurvey. Bandara et al [1] propose an overall approach to conduct systematic literature review in the context of information systems making use of NVIVO, a qualitative data management tool, and ENDNOTE, a personal reference database.

6. CONCLUSION

In the present work we have reviewed the concepts of Experimental Software Engineering focused on controlled experiments. Additionally, we have addressed challenges found throughout different phases of an experiment, from an initial experimental plan to final study validations. Issues found throughout the process of experimenting were listed in order to present the complexity of the domain and confirm the need of better tools to realize experimentation.

A domain-specific language was defined and presented according to the needs aforementioned and based on experienced researchers in the empirical software engineering topic. The process described by Fowler was followed in the conception of the ESEML Workbench, its domain model, and the visual representation of the DSL. At last, we present a case of an automatic generation of the experimental plan, which was the starting point for the present study. See a generated plan at http://bit.ly/RusZYm.

Finally, the potential of the proposed DSL was emphasized through the exemplification of what can be done in code generation applied in model transformation, for the activities involved in a controlled experiment.

7. FUTURE WORK

For future work, our DSL Workbench is intended to generate any artifacts necessary to conduct a controlled experiment, including software to collect data effectively from the experimental units. Limited transformations and validators were implemented in our DSL due to scope and time limitations. Those are intended to be included in the next release.

Table 1: ESEML domain model elements.

Element	Short description
Controlled Experiment	The main element that starts the controlled experiment modeling.
Design	Specifies the experiment design (i.e. latin square, one or two sample comparison and so on).
Object	The same as experimental unit.
Subject	The participants of the experiment.
Dependent Variable	Variables that carries experiment response value.
Factor	The provoked variations or independent variables.
Hypothesis	Used to modeling the experiment hypotheses.
Parameter	Fixed independent variables.
Treatment	Values that factors can assume.
Goal	Experiment objectives.
Question	Experiment research questions related to specific goal.
Metric	Metrics associated with specific question.
Adhesion Test	Goodness-of-fit statistical test.
Hypothesis Test	Describes the statistical hypotheses tests and its level of significance.
Validity	Used to modeling internal, construction, external and conclusion validities.

We found out that a Systematic Review of Studies proposing a formal model for controlled experiments became necessary to proceed with our domain model. Thus, we are now intended to start a Systematic Review of Studies to provide the deeper rationale that is needed.

Furthermore, we want to obtain a deeper understanding on how to automate validation of formalized hypothesis and identify confounders within these. Thus, through systematization, our DSL will try to minimize bias in the controlled experiments using our tool or, in other cases, provide visual cues so domain experts can fix the formalized hypothesis manually.

When the language and environment achieve a greater degree of maturity, we intend to use empirical methods in a systematic way to assess whether the proposed ESEML really facilitates the modeling process and the definition of an experimental plan.

8. REFERENCES

[1] W. Bandara, S. Miskon, and E. Fielt. A systematic, tool-supported method for conducting literature reviews in information systems. In V. Tuunainen, J. Nandhakumar, M. Rossi, and W. Soliman, editors, *19th European Conference on Information Systems : ICT and Sustainable Service Development (ECIS 2011)*, Helsinki, Finland, 2011.

[2] V. Basili. The role of experimentation in software engineering: past, current, and future. In *Software Engineering, 1996., Proceedings of the 18th International Conference on*, pages 442 –449, mar 1996.

[3] S. Cook and G. Jones. *Domain-specific development with visual studio dsl tools*. 2007.

[4] M. Fowler. Domain-specific languages. pages 0–321, 2010.

[5] R. E. Garcia, E. N. Höhn, E. F. Barbosa, and J. C. Maldonado. An ontology for controlled experiments on software engineering. In *SEKE*, pages 685–690, 2008.

[6] Juristo and Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[7] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[8] T. Punter, M. Ciolkowski, B. Freimut, and I. John. Conducting on-line surveys in software engineering. In *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, pages 80 – 88, sept.-1 oct. 2003.

[9] H. Siy and Y. Wu. An ontology to support empirical studies in software engineering. In *Computing, Engineering and Information, 2009. ICC '09. International Conference on*, pages 12 –15, april 2009.

[10] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen. The future of empirical methods in software engineering research. In *2007 Future of Software Engineering*, FOSE '07, pages 358–378, Washington, DC, USA, 2007. IEEE Computer Society.

[11] D. I. K. Sjoberg, J. E. Hannay, O. Hansen, V. By Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Trans. Softw. Eng.*, 31(9):733–753, Sept. 2005.

[12] W. Tichy. Should computer scientists experiment more? *Computer*, 31(5):32 –40, may 1998.

[13] K. Torii, K. Matsumoto, K. Nakakoji, Y. Takada, S. Takada, and K. Shima. Ginger2: an environment for computer-aided empirical software engineering. *Software Engineering, IEEE Transactions on*, 25(4):474 –492, jul/aug 1999.

[14] G. Travassos, P. dos Santos, P. Neto, and J. Biolchini. An environment to support large scale experimentation in software engineering. In *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, pages 193 –202, 31 2008-april 3 2008.

[15] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.