Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering

Guilherme Horta Travassos and Márcio de Oliveira Barros

COPPE / UFRJ – Computer Science Department
Caixa Postal: 68511 - CEP 21945-970 - Rio de Janeiro – RJ
Fax / Voice: 5521 2562-8675
{ght, marcio}@cos.ufrj.br

Abstract: Empirical software engineers usually use a two-staged taxonomy based on the terms in vivo or in vitro, according the control level that can be attained upon the environment where these studies are executed. Despite its importance, this taxonomy does not capture relevant issues concerned with different study categories, mainly those exploring computer models. We faced this limitation when classifying experiments regarding software project management in which we were involved. By analyzing other sciences that also apply experiments to support their research; we observe that the in vivo/in vitro taxonomy has been extended to accommodate the use of computer models. Two new experiment classes (namely in virtuo and in silico) complement the original taxonomy. They regard the use of computer models to simulate the environment, the object under analysis and the subjects that take part in an experimental study. When applied to the software engineering experiments, this four-staged taxonomy helped us to classify our studies, revealing additional experiment features not observed for the conventional studies yet. In this paper we propose the use of this four-staged taxonomy for categorizing software engineering experiments and evaluate some implications of using in virtuo and in silico experiments in software engineering. Besides, based on the possible contributions such studies can bring, we propose some topics to compose a research agenda for the future of empirical software engineering.

1. Introduction

Empirical software engineering literature usually classifies experimental studies as in vitro or in vivo [1]. This taxonomy helps experiment practitioners by bringing up major characteristics of the experiment being analyzed, such as the level of control that can be attained, external factors influences, and the ability to generalize experimental results. However, this taxonomy is not enough and could not strictly classify some recent experimental studies. We have identified that when trying to categorize some experimental studies in which we were involved [2]. A major
characteristic – the use of computer models – strongly influenced the experimentation process and could not be properly emphasized by the \textit{in vivo/in vitro} taxonomy.

Analyzing empirical studies classification schemas that are used in other sciences (for instance, mathematical biology) we observed a four-staged taxonomy including the following terms\textsuperscript{1}:

- \textit{In vivo} experiments: such experiments involve people in their own environments. In software engineering, experiments executed in software development organizations throughout the software development process and under real circumstances can be characterized as \textit{in vivo};
- \textit{In vitro} experiments: such studies are executed in a controlled environment, such as a laboratory or a controlled community. In software engineering, most \textit{in vitro} experiments are executed in universities or among selected groups of a software development organization;
- \textit{In virtuo} experiments: such experiments involve the interaction among participants and a computerized model of reality. In these experiments, the behavior of the environment with which subjects interact is described as a model and represented by a computer program. In software engineering, these studies are usually executed in universities and research laboratories and are characterized by small groups of subjects manipulating simulators.
- \textit{In silico} experiments: these studies are characterized for both the subjects and real world being described as computer models. In this case, the environment is fully composed by numeric models to which no human interaction is allowed. Due to the need of a large amount of knowledge (see section 4), \textit{in silico} studies are still very rare in software engineering, being limited to areas where subject participation is not an experimental study issue or intelligent agents can replace human subjects. For instance, we can find \textit{in silico} studies applied to software usability experimentation, such as software performance characterization.

Major advantages of \textit{in virtuo} and \textit{in silico} experiments in software engineering are concerned with the cost and feasibility of replicating a real-world situation. In this paper, we explore the characteristics of \textit{in virtuo} and \textit{in silico} experimental studies. We show examples of \textit{in virtuo} experiments and highlight specific attributes that can make these studies advantageous or problematic for specific situations. We analyze our capability to build quality models and their impacts on experimental studies’ conclusion capabilities. We believe that enlarging the way we classify our studies can make us able to better plan experiments and reduce the risks for their execution. Besides, such experiments could reduce the time we normally need for understand software engineering technology [3], speeding up its transfer to the software industry.

Six sections compose this paper. The first one comprises this introduction. Section 2 discusses \textit{in virtuo} experimentation in software engineering. Next, section 3 presents examples of software engineering experiments that could be classified as \textit{in virtuo} studies. Section 4 discusses \textit{in silico} experimentation in software engineering.

\begin{footnotesize}
\textsuperscript{1} http://www.enib.fr/~tisseau/doc/textes/invirtuo.html
\end{footnotesize}
The possible contributions of in virtuo/in silico studies and topics for future research are discussed in section 5. Finally, section 6 summarizes and concludes the paper.

2. In Virtuo Experimentation

The in virtuo expression is a neologism that represents a state between in vivo and in vitro. An in virtuo experimental study is conducted in a virtual environment, composed by numeric models that are manipulated by human beings.

A numeric model is a computer-oriented representation of a real-world element or phenomenon. It describes the assumptions regarding the element or phenomenon, conveying an explicit representation of knowledge that the model developers possess about it. A computer model is useful to make such knowledge available to a larger interested group, to allow the validation of such knowledge, and to predict the modeled element’s behavior when subjected to distinct conditions. A model’s major qualities rest on its capacities to describe, explain, and predict the element or phenomenon that it represents as closely as possible to the real world system.

Model behavior is analyzed through simulation. The behavior of a model in a selected instant is usually described as a set of variable values or a set of predicates. Simulation is a mechanism that reproduces a system’s behavior through numerical calculations accomplished by a computer program [4]. It is usually applied when a model’s behavior cannot be evaluated in an analytic fashion, that is, by solving an equation [5]. There are several distinct kinds of simulations [6] [7] [8], each one useful according to distinct model characteristics.

In in virtuo experiments, a computer model represents the object under study and, in some cases, the environment where the experiment is executed. Subjects that interpret the results provided by model simulation manipulate the model through a computer program.

Major advantages of in virtuo experiments regard the cost and feasibility of replicating a real-world situation. If the object under study evolves over a long time period, such as a multi-month or multi-year software development project, its replication in the real world for several participants can be prohibitive. Also, even if the object can be manipulated in an adequate time frame, the cost of creating the real object can also be prohibitive.

By using a computer model, an analyst can manipulate, for instance, time while studying the element or phenomenon represented by the model. The passage of time can be fastened, so that several months are simulated in seconds, or delayed, so that detailed characteristics of the modeled objects can be precisely examined. Relevant events occurring within this period can be highlighted, focusing subject attention in the most important issues, instead of the specific details that can happen during an experiment run. The model can be analyzed multiple times, under distinct initial conditions, with low replication cost.

A second advantage of in virtuo experiments concerns with data collection, which can be partially or completely automated by the program that performs model simulation. Error-free data collection can be a difficult task and a threat to validity in several experiments. By implementing data collection procedures in the program that
represents the object under analysis and the experimental environment, this validity risk may be contained.

Unfortunately, threats to validity are also the major problem to be observed while executing or analyzing in virtuo experiments. Most of these threats are concerned with the quality of a model, that is, its capacity to correctly represent the real-world elements’ behavior. The quality of the experiment, such as its potential of repetition or generalization to other groups, depends on how closely the model represents the real-world elements.

Evaluating model’s quality represents a hard issue. A computer model is always a simplified version of the reality, limited in its ability to replicate every detail presented by the real system behavior, since it emphasizes some aspects of the real world system while nominates others as details to be hidden. The purpose of a model identifies which aspects are relevant to be considered, being a major factor to determine its quality. When using a computer model in an experimental study the researcher must verify whether its purpose is compatible with the objects under analysis and the environment in which the study is executed. The model and the study shall focus on the same aspects of the environment/object. Distinct focus may take to wrong conclusions and represents a confounding factor that must be avoided.

Several tests where designed to evaluate a computer model’s quality given its purpose. They can be classified into two major groups: historical fit tests and structural tests [9].

Historical fit tests are useful when historical data about the aspects considered by the computer model are available. By using these data, a behavior replication test evaluates whether the model can replicate the behavior presented by the system in the past (as documented by the historical data). If the system does not behave in the expected way, its structure, parameters, and boundaries are suspect and structural tests must be applied.

Beyond the behavior replication test, other historical fit tests can be applied. Sensitivity tests observe variations on model results due to small variations in model parameters. Behavior prediction tests analyze if a model behaves as expected when subjected to particular actions during its simulation. Such tests are useful evaluation mechanisms for analyzing the model behavior, helping to identify possible flaws in the model structure.

Structural tests are useful when there are no historical data available, data are difficult to collect and not precise, and when a model behavior does not converge on the real system behavior described by the historical data. Such tests are concerned with the relationships among model elements and their real-world counterparts. They tend to rely on discussions and model scrutiny by specialists instead of replicating historical data. Major structural tests regard parameters and results dimensional consistency, the validity of parameter values (specially for soft values), the inclusion of all relevant factors observed in the real-system within the model, and a model applicability to a particular context.

However, neither test can definitely indicate whether a model is wrong, since complex models may present unexpected behavior even when small changes are enacted upon its parameters.

Other problems related to in virtuo experiments refer to training, motivation, and presentation of the results. Training is a very important risk factor in in virtuo
experiments: since subjects manipulate a computer model, usually represented and manipulated by a computer program, these subjects must be trained in using this program. If subjects do not know exactly how to handle the program and how to manipulate the model, errors (due to its representation in the computer and in the interaction between subjects and model) can affect experimental results. Sometimes subjects must be eliminated as outliers in in virtuo experiments due to incorrect use of the simulation program.

The use of a toy example in the training process and the execution of the experiment after a qualified training section seem to be a good choice when accomplishing in virtuo experiments. The toy example can be a simpler version of the proposed model, which is manipulated by the subjects. The moment subjects manipulate the toy example they learn the interactions with the model and the program that represents it, thus reducing the chances for errors in the experimental results due to this source. Further, executing the experiment just after the training session aims to reduce the drawbacks and possible issues regarding the use of the computational model by the subjects. Doing so, it also contributes to make sure the whole environment and configurations are set up for the study.

Another interesting issue to deal with is the number of subjects for in virtuo experiments. Based on our experience, we suggest that in virtuo experiments should run with no more than eight subjects at a time because the researcher that planned the experiment or a person who knows the plan, its materials, and procedures, must be present to observe the subjects. This helps to detect subjects that should not be taken into account during analysis, due to incorrect use of the study materials. Larger groups of subjects should be split into smaller groups, so that observation can be effective.

The motivation and seriousness problem is not specific to in virtuo experiments: it is also shared by in vivo and in vitro studies in software engineering. Subjects must be motivated to take part in the experiment or they may not act as they would in a real-world situation. For instance, if a subject is expected to act as a designer selecting a design alternative among a set of potential solutions, the designer must consider aspects such as software maintenance, ease of coding, among others. If the subject is not motivated by the experiment, he may not invest the same time and brain effort to analyze the group of alternatives, selecting at random just to fulfill the study. Intuitive and ease to use simulator program interfaces may help to keep subjects motivated.

Presenting the dynamics that govern the computer models used in the experimental study during the training sessions can also help to motivate subjects. Subjects tend to dedicate more effort before making a decision when they know the laws that govern the elements that will be influenced by their decisions. A careful examination and explanation of the technical terms used in the study, along with a presentation of the assumptions that compose the simulated model should be considered at this time.

The last problem that we found in in virtuo experiments is concerned with the readability of project results. The program that contains the model must present its results and characteristics using an ease representation for the user. Difficulties in understanding model results may affect experimental results. Again, the presentation facet of the model must closely represent the real-world elements.

In the following section, we present in virtuo studies examples. In particular, the first one has shared some of the preceding problems. In this case, the representation
problem had a strong influence in the study results. These problems were filtered during the analysis.

3. In Virtuo Experiment Examples

Several examples of in virtuo experiments can be found in the software engineering literature. Barros et al [2] present an in virtuo empirical study that analyzes the feasibility of a set of model based project management techniques. These techniques support decision-making by allowing a manager to build a model conveying project activities, its development team, and the artifacts that will be generated and consumed during project development. This model is called a project model. Next, the manager analyses the behavior presented by the project model when subjected to the effects of uncertain events, management actions, and strategies. A separate model, namely a scenario model, describes each of these events, actions, and strategies. Scenario models integration to a project model and further simulation of the resulting model supports this evaluation.

The study aimed to observe whether managers who used scenario models to support their decisions would perform better than managers who relied only on their experience and other techniques. The time taken to conclude a proposed project was selected as the experimental study performance criterion.

Experiment subjects were 14 master degree students, 3 doctoral students, and one undergraduate student. Each subject was asked to manage the proposed project without the intervention of other subjects. They were randomly assigned into two groups: one to use scenario models while managing the project and one to serve as a control group, accomplishing the study without the aid provided by the models.

Instead of executing a real project with similar characteristics (team formation, team domain knowledge, activities to be performed, among others) for each subject, a project emulator was used. The emulator is a software tool that controls and presents to its user the proposed project’s behavior, adjusting it in accord with decisions taken by the users. Decision points where subjects could act included: determining which developer to assign for each project activity, deciding how much time to spend in quality control activities, and defining the number of hours each developer should work per day. The subjects from the first group could test the project sensibility for their decisions in a scenario simulation environment before applying them to the project emulator. After applying their decisions, the subjects could instruct the emulator to advance project time by several days, observing the effectiveness of their decisions upon the project behavior.

Questionnaires were used to collect qualitative data that characterized each distinct subject and directly addressed the questions of feasibility and usefulness. The project emulator gathered quantitative information about the time that each subject took to conclude the project. After eliminating outliers, the quantitative data was submitted to a 95% T-test [10] that showed that the average time to conclude the project for subjects applying scenario models was less than the average time taken by the ones who have not used the scenarios. A Mann-Whitney [1] rank-based statistical analysis asserted the T-test results. Also, the questionnaires returned by subjects from the first
group indicate that some subjects had difficulties interpreting the results produced by the simulator.

The use of the project emulator characterizes this experiment as *in virtuo*: the object that is manipulated in the study (the proposed project) is represented as a computer model. Also, the environment where the object resides (the project emulator) is a computer program. So, subjects interact with a computerized model of the reality, thus classifying the study as *in virtuo*.

Hayhurst [11] presents a framework that allows researchers to perform experimental studies to scientifically evaluate their hypotheses and assumptions about software engineering methods used in spacecraft guidance control systems development. The framework consists of a software simulator, which emulates real landing control hardware, and a configuration management system that conveys information about several independent executions of the guidance control software under development.

To evaluate the proposed framework, the author presents a case study involving two independent software development teams that were required to build a guidance control system from the same requirements specification document. Each team consisted of a programmer and a verification analyst. Both teams followed a waterfall-like software development process life cycle, comprising requirements identification, design, coding, and integration activities. After integration, the systems developed by both teams were run in the landing control hardware simulator.

More than 7700 complete landing trajectories were run with the two systems. No relevant differences were noted during those tests, neither for the final results provided by the systems to the landing control hardware nor for intermediate values computed by the programs. The simulated vehicle landed successfully about 97% of the time, which was within the successful landing rate of 95 percent that was established for the system [11].

The case study showed indications that the proposed framework provided the desired functionality to simulate landing control hardware and to support the operational evaluation of guidance control software. Particularly, the simulator was useful to reduce the cost of testing the software in a real hardware before a higher confidence level on its correctness could be achieved. The simulator could also support the automation of data collection, although this issue was not explored in the case study.

The authors also noted that a significant amount of resources might be required to develop such an experimentation framework. But, although the framework does not eliminate many of the costs incurred in conducting software related experiments, the framework did provide a platform for consistent comparison for a small application [11].
4. In Silico Experimentation

The *in silico* expression can be used to characterize those studies that are completely executed by using computer models. It means that the environment, object, and subject behavior are represented by models that simulate their relevant features. Figure 1 shows the different types of computer models used by software engineering experimental studies. The arrows indicate that an experiment type requires knowledge acquired from executing experiments from preceding types to build its computer models.

The *in silico* classification for experimental studies is not as precise as the first three categories. We have found that it depends on the scientific area where experiments are being accomplished. For instance, in the bio-informatics area, *in silico* experiments represent those studies that encompass multiple combinations of program and data resources, where management is complex when accomplished manually [12]. Subject behavior in these experiments usually is connected to the ability of providing the feasible combinations of data and programs calls necessary for the study. There is an enormous amount of information used by the study, which is distributed through the web to automatic scripts or programs that execute the experiment. The human interaction is restricted to the starting point of the whole process, where the scientist (called e-scientist) configures the workflows that will guide the experimentation process, and at the end of processing, with new data available for analysis.

The *in silico* expression can also be used to qualify an experimentation environment. In this case, the environment is fully composed by numeric models to which no human interaction is allowed. The results of a simulation become then assumptions that one seeks to check by conceiving experiments on a singular prototype of the real system. Variations of *in silico* environments can also be

---

2 [http://www.quinion.com/words/weirdwords/ww-ins1.htm](http://www.quinion.com/words/weirdwords/ww-ins1.htm)
found in the electrical and education engineering fields. Callaghan et al. [13] describes an interesting approach for remote training and experimentation about embedded systems putting available a virtual laboratory on the web. Geoffroy et al. [14] presents a virtual assistant for Web-based training. The training and the evaluation process are shared between real and virtual assistants in order to deliver a tutoring scheme adapted to Web-based experimentation.

These research areas have in common the possibility of describing most of the models by using equations or computer algorithms. Behaviors usually are represented by the repetition of tasks (clerical activities) that do not demand any elaborated reasoning process to their execution or even by just providing interface mechanisms to virtual devices. For instance, the combination of data and program calls for the in silico bio-informatics experiment must be repeated for several times following the previously defined workflow and specified formation constraints.

However, there is some specific that distinguishes software engineering experiments from these previous experiments: because the constructs of software are the result of human reasoning, software experiments cannot ignore the human factors that do exist. Experience, intelligence, job security, personality, and other factors can impact software development efforts. In general, the level of control that is required to ensure that the differences noted in a software experiment are attributable only to variations in treatments (i.e., variations in applying different software engineering methods) is nearly impossible to achieve. Besides, the number of software developers that would be needed so that randomization techniques could be used to eliminate bias introduced through human factors would be huge. Consequently, the cost of conducting a full-scale randomized experiment would be prohibitive. The nature of software engineering may be such that it is intuitively not conducive to true scientific experimentation. This is one of the features in silico experiments could support. By being able to model several distinct subject behaviors, a large “virtual” population of study can be organized and tested prior to organize the experimental study on the real world. It can reduce the major hypotheses to simpler ones, by supporting them whether some conditions can be verified. Such conditions would be postulated in the models used by the in silico experiments, being further investigated in future in vivo experiments.

In silico experiments inherit the threats of validity regarding in virtuo studies. Besides, by using models for the subjects, new threats of validity can be identified. One of them regards the difficulty on building the subject behavior model by itself. The different variables that can affect subject behavior (e.g. level of experience, motivation, seriousness, personality, productivity, job security, cultural issues and others) must be considered when developing these models. However, they need to be first identified and characterized. To make the modeling of the subject behavior harder, the combination of these characteristics introduces concerns about the different subject states and distinct behavior modes necessary to run the experimental study for a large population of software developers. Some of these variables are easier to characterize (for instance, productivity), others can be fuzzy (e.g., level of experience). Fortunately for these situations, motivation and seriousness variables can always be assigned to positive values. Unfortunately, this is not completely true in real world.
The linkage among the computer models and the operational environment represent other possible threat of validity for in silico experiments. Computer models usually work as components (programs), interacting among them and with the operational environment. Concerns about computing performance, interoperability, network type, bandwidth, and so on become a reality. Without the definition of minimal standards and integration requirements for such models and the computing platform, this will be hard to guarantee that a specific study can be replicated in other places.

Therefore, prior to constructing models to capture all experiment dimensions, we need to truly understand the issues concerned with the development of software to be computerize them. A solid body of knowledge on software engineering anchored on knowledge resulted from previous in vivo / in vitro / in virtuo experiments is then necessary.


The experimentation process posses a recursive nature. On the one hand, knowledge acquired from costly in vivo/in vitro experiments is required to promote the execution of cheaper in virtuo/in silico experiments. But on the other hand, control, risks, and cost threats faced by in vivo/in vitro experiments can be reduced when executing in virtuo/in silico studies. Such studies shall evaluate the system behavior according to some hypotheses, filtering out the relevant hypotheses that should be evaluated in in vivo/in vitro experimental studies. By using computer models to select relevant experiments to be executed, in silico/in virtuo studies may reduce the overall cost of software engineering experimentation. Figure 2 aims to capture these ideas, showing the paradox between in vivo and in silico experiments.

That is the essence of science. For instance, engineers only were able to build safe jet planes after acquiring knowledge on how aluminum behaves when exposed to the higher atmosphere layers (at the cost of some airplanes crashes). We need to invest on the planning and execution of in vivo/in vitro experiments to build a body of knowledge that will allow us to go further on the main purpose of having faster and cheaper experiments. We believe that the contributions of in virtuo/in silico experiments for software engineering are many. Experiment risks and costs reduction can be listed here, together with the productivity increasing that experimenters could obtain using such types of experiments should be seriously considered.

The need for faster and cheaper studies is motivating researchers to produce facilities to support software engineering experiments. For instance, some initiatives have been accomplished to build Computer-Aided Empirical Software Engineering (CAESE) framework as a substrate for supporting the empirical software engineering lifecycle. CAESE supports empirical software engineering in the same manner as a CASE environment serves as a substrate for supporting the software development lifecycle [15]. Others are looking for a standard way knowledge concerned with software engineering experiments can be organized and made available to researchers [16]. Although important, these initiatives cannot answer most of the questions regarding in vivo/in silico experiments yet. Software engineers must also be thinking
about the necessary infrastructure for building the experiment computer models and how to make them available to in virtuo/in silico experiments. There are a lot of threads that must be understood prior researchers tie the computational pieces together for running their virtual experiments [17].

The future of empirical studies in software engineering relies on the perspective used for the software engineering research [18]. To combine the lessons learned from different research topics such as software engineering environments, COTS development, and e-speak [19] can give researchers some clue about how the future will look like for the Empirical Software Engineering Environments. There is a lot of work to be done regarding, for instance, the experimentation process, data representation standards, experiment knowledge management, characterization of tools, services, computer model’s quality and the needed models for each type of experiment. This represents an ambitious long-term research agenda that will involve software engineers and different computer science researchers.

6. Conclusions

In this paper we have proposed and justified the use of a four-staged taxonomy for categorizing software engineering experiments. The taxonomy adds two new experiments kinds (namely in virtuo and in silico) to the traditional in vivo / in vitro classification of software related experiments. Human subjects manipulating virtual, computer model-based, objects characterize in virtuo experiments. Experiments are classified as in silico when the environment in which the experiment is executed, the objects under analysis, and subject behavior are all described as computer models.

In virtuo/in silico experiments explore simulation of the real world to support different experimentation activities. According [18], simulation has been used as a practical research exploratory method for experimentation. Despite the weaknesses regarding the reality captured by the data (reflected by the computer model) and
development method adherence, the strengths concerned with the possibility of automation, ease of replication, be applied to tools and allow evaluation in safe environments highlight the importance of increase the research regarding *in virtuo/in silico* experiments. Table 1 summarizes this discussion.

**Table 1 – Four staged taxonomy for Software Engineering Experiments**

<table>
<thead>
<tr>
<th>Experiment Type</th>
<th>Applied Model</th>
<th>SE Knowledge Level</th>
<th>Control, Risk and Cost threats</th>
</tr>
</thead>
<tbody>
<tr>
<td>In vivo</td>
<td>No model</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>In vitro</td>
<td>Environment (not computational, but controlled model)</td>
<td>Intermediate</td>
<td>Intermediate</td>
</tr>
<tr>
<td>In virtuo</td>
<td>Object and environment as computational models</td>
<td>Intermediate</td>
<td>Intermediate</td>
</tr>
<tr>
<td>In silico</td>
<td>Subject behavior, object, and environment as computational models</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

The ideas described in this paper have been used to motivate some of the research activities of the Experimental Software Engineering Team at COPPE/UFRJ. The results regarding these activities are usually made available to the scientific community at http://www.cos.ufrj.br/~ese.

**Acknowledgements**

The authors would like to thank CNPq, CAPES and FINEP for their financial investment in this work.

**References**


