# Practical Experiments are Informative, but Never Perfect

Rosalva E. Gallardo-Valencia
University of California, Irvine
5051 Donald Bren Hall
Irvine, CA 92697-3440
1-949-824-4047
rgallard@ics.uci.edu

Vivian Olivera
University of California, Irvine
5051 Donald Bren Hall
Irvine, CA 92697-3440
1-949-824-4047
volivera@ics.uci.edu

Susan Elliott Sim
University of California, Irvine
5226 Donald Bren Hall
Irvine, CA 92697-3440
1-949-824-2373
ses@ics.uci.edu

## ABSTRACT

The design of empirical experiments involves making design decisions to trade off what is ideal against what is achievable. Researchers must weigh limitations on resources, metrics, and the current state of knowledge, against the validity of the results. In this paper, we report on the design decisions we made in a small controlled experiment and their effects on the conclusions of the study. The goal of the study was to measure the impact of requirements formats on maintenance tasks. We encountered problems with the subjects' lack of expertise in the technology used, the equivalence of subjects in our experiment conditions, and the number of subjects. These issues meant that we were able to draw conclusions about how subjects worked with the requirements formats, but not about the effect of the formats on the completeness of the implementation. We had a practical and doable experiment, but our results were not conclusive, only informative.

## Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *Elicitation methods.*

## General Terms

Design, Experimentation, Human Factors.

## Keywords

Use Cases, User Stories, Agile Requirements, On-Site Customer, Controlled Experiment, Empirical Study, Experience Report.

## 1. INTRODUCTION

Designing an experiment has much in common with designing software. It is often necessary to select one option or a combination of options based on the availability of resources. Software developers are aware that the decisions made will have an effect on the system, but these decisions will help to make the system more feasible. In the same manner, researchers also have to make design decisions. Each design decision will have trade-offs. On one hand the decisions will make the experiment practical, but on the other hand the decisions could impact the validity of the results. To trade off these constraints, researchers must keep in mind the larger goal: to perform experiments that will provide valuable information and insight about the phenomenon being studied.

To illustrate our analogy, imagine that we have to develop a small web application that allows 50 students in an elementary school to upload their homework and keep some information about the status of the uploaded files. We want to create the best design but we also have to consider the budget (small) and time constraints (short). We can consider two options to store the information: a Database Management System (DBMS) or XML files. On one hand, if we use the DBMS we can have access to the information via SQL commands and have all the power of this specialized system. On the other hand, if we store information in an XML file, we can easily create the file on disk, and read it directly without installing any other software. We could choose to implement the system using XML files knowing that it is not the best design but it will meet the requirements and constraints.

We were interested in investigating the effects of different requirements formats on the performance of maintenance tasks. Should we do a case study in an industrial setting, or a controlled experiment in a laboratory? Should we do an exploratory qualitative study or test a hypothesis quantitatively? Should we do a detailed study with a small number of subjects or a more constrained study with a large number of subjects?

We decided to conduct a laboratory experiment using a small number of subjects and to collect both qualitative and quantitative data. A laboratory experiment would allow us to perform head-to-head comparisons of the formats and to draw conclusions about causality. We would collect both quantitative and qualitative data, so that we could both explore and test hypotheses.

Having selected the basic structure of the study, there were still many other choices to be made. In this paper, we will discuss the design decisions that affected the conclusions and validity of the study. Some of these decisions were made to optimize on scarce resources, in particular, the availability of subjects and the length of the experiment. Other decisions reflected the novelty of the research problem and the limitations of our methods.

It is always a challenge to find willing subjects for experiments in software engineering. We only screened for their knowledge of Java. However, the experimental task involved making three changes to an existing web application. As a result, only one out of nine subjects was able to complete the task, which led to inconclusive results regarding the effect of the formats on performance of the maintenance tasks.

When conducting a controlled experiment the length of the session is limited by how long someone can concentrate and how much time someone can commit in a single block. In our study, the experiment tasks combined with the tutorials, familiarization task, and debriefing interview, each session was very long (2.5 hours). Consequently, we administered only a short questionnaire about their background, and nothing on their personal or cognitive traits. Without this data, we were not able to effectively counterbalance the amount of subject experience in each of the conditions. As well, we were not able to control for background experience when analyzing subject performance.

The remainder of the paper is organized as follows. Section 2 presents related work done in empirical experiment design in software engineering, trade offs in experiment design, and research design. Section 3 describes the experiment we conducted which we use as an example for our hypothesis. Section 4 discusses the trading off between practicality and perfection. Our conclusions are presented in Section 5.

## 2. BACKGROUND

There is a great deal of literature on the design of empirical studies. There are many books available from the social sciences, and a number of papers, tutorials, and books for software engineering specifically. For example, Kitchenham et al. [3] suggested taking into consideration eleven design guidelines for empirical research in software engineering. Some of these guidelines are related to the identification of the population, the process for allocating the treatments, among others. Following these guidelines will help researchers to have an ideal experiment design, but resource limitations could make it difficult to follow them.

At a macroscopic level, the trade-offs between field studies and laboratory studies, long-term studies and single session studies, qualitative and quantitative studies are well known.

Perry et al. [4] proposed that the design of better studies and the effective collection of data could help create better empirical studies in software engineering and draw more conclusive interpretations from the results. They concluded that no study is perfect and the challenge is to conduct credible studies. Perry et al. also studied the management of trade-offs in the experiment design. They suggested that design decisions should try to maximize accuracy of interpretations, relevance, and impact. However, these decisions should be subject to resource constraints and risk.

Not only empirical studies but formal experiments in software engineering as well depend on careful experiment design to have useful results. Pfleeger [5] presented the needed activities to design and analyze an experiment in software engineering. She explained in detail the principles of experimental design, which aim to satisfy the need for simplicity and for maximizing information. The author emphasized the importance of having simple experiment designs that help making a practical experiment. Also, simple design reduces the use of time, money, people, and other experimental resources.

However, there is little in the literature on how to make design decisions at a detailed level. There is also little discussion of the consequences and lessons learned from particular design decisions. It is here that this experience report seeks to contribute to the software engineering literature.

## 3. DESCRIPTION OF THE EXPERIMENT

### 3.1 Experiment Design

We performed an initial controlled experiment to study which requirements format was most effective: Use Cases alone, Agile Requirements alone (User Stories with access to an On-site Customer), or Use Cases with Agile Requirements. A full description of the experiment has been published elsewhere [1].

We had a small sample of nine subjects, each assigned to one of the three conditions. We attempted to counterbalance the level of experience of the subjects in each condition. In the study, subjects were asked to modify a shopping cart in a web application, by changing an existing feature and adding two new features.

#### *3.1.1 Subjects*

Nine subjects participated in our experiment. We recruited them by word of mouth. Most of them were graduate students but we also had an undergraduate student and a research assistant. Most of them had a major in Computer Science. Five of them stated to have had between 1 to 2 years of experience in Java Web Development. More details of our subjects are shown in Table 1.

**Table 1. Characteristics of subjects**

| | |
|---|---|
| Average Age | 25.55 |
| Gender | 3 Females<br>6 Males |
| Occupation | 7 Graduate students<br>1 Undergraduate student<br>1 Research Assistant |
| Degree Major | 8 in Computer Science and<br>1 in Aerospace |
| Years of Experience in Software Development | Range: 0-15 years.<br>Average: 4.72 years. |
| Years of Experience in Java Web Development | <1 year: 4<br>1 year: 1<br>2 years: 4 |

#### *3.1.2 Conditions*

The goal of our experiment was to measure the level of impact that different requirement formats could have on how people implement a system. To achieve our goal, the experiment had three conditions. First, subjects in the UC Group were given the requirements only in Use Cases. Second, subjects in the US&OC Group used agile requirements. They received the requirements in User Stories and they also had access to an On-site Customer via chat. Third, subjects in the UC+US&OC Group used all the requirement formats used by the previous groups. We will refer to each condition by the name of the group from here onwards.

#### *3.1.3 Procedure*

Subjects first filled out a Background Questionnaire to provide information about their background and experience. Then we provided tutorials in the requirements format subjects would use. A familiarization task was also included to familiarize subjects with the Eclipse Workbench, Tomcat Application Server, and implementing JSPs (Java Serve Pages) and Servlets.

Subjects were given descriptions of three features in one of three requirements formats, according to their assigned experimental condition. They had to understand the requirements, perform three maintenance tasks, and to think aloud as they worked. The maintenance tasks involved modifying a feature and adding two new features to a shopping cart for a web-based application. This subject system, called "An Online Boat Shop," was taken from the book "More Servlets and Java Server Pages" by Marty Hall [2]. The boat shop application was developed using JSP and Servlets, it uses a Tomcat application server, and it does not need access to a database. The source code of the application includes 10 JSP files, 12 Java™ files, and a XML configuration file. In total, there were 1,340 lines of source code.

The first maintenance task asked the subject to change how items were added to the shopping cart. Initially, each time the user added an item, the system did not verify whether one was already in the cart. We asked our subjects to add a "Quantity" attribute to the shopping cart and to increment it when an existing item was added to the cart. The second maintenance task required our subjects to add a new feature that allowed users to update the quantity of an item in the shopping cart by entering the new quantity in an input field. The third implementation task asked subjects to add the functionality to delete items from the shopping cart.

If we observed that a subject would not be able to complete the tasks in the time available due to unfamiliarity with JSP and Servlets, we asked them to document their design. We suggested that they draw sketches of screens, but they could draw or write whatever they needed to show that they understood the requirements. The design sketch allowed us to collect data about how well they understood the requirements when they were not able to complete the implementation.

Finally, the subjects participated in a Debriefing Interview to provide feedback and insight about their experience using the requirements formats, about their preferences among the formats, and about their performance in the implementation and design tasks.

#### *3.1.4 Analysis*

We analyzed the data by reviewing the screen, video, and audio recordings of the experiment. We tallied the amount of time that they spent reading the requirements, chatting with the Customer (where applicable), and implementing the features. We also analyzed the chat transcripts, counted the number of questions asked, and judged the relevance of the questions.

We also collected data from the coding and design to provide an objective, performance-based measure of how well subjects understood the requirements. For subjects who completed the implementation, we scored the program code. Otherwise, we scored the design drawings and the explanation that they provided. The maximum possible score was 30 points. Finally, we examined subject responses from the Debriefing Interview.

We tested our data using non-parametric statistics. This kind of statistical method is appropriate for our study because we have a small sample size. In addition, we converted our ratio data into ordinal data by rank ordering the times and performance scores for the subjects.

## 3.2 Results

We found that subjects using Agile Requirements spent the most time understanding the requirements (average = 28:03 minutes), followed by subjects who used both Agile Requirements and Use Cases (average = 18:00), followed by subjects using only Use Cases (average = 4:13). This difference was found to be statistically significant at $p<0.05$ using the Kruskal-Wallis one-way analysis of variance by ranks [3]. This result is surprising for a number of reasons and to understand this difference we will examine how subjects went about understanding the requirements. Figure 1 shows how much time each group spent using each requirement format.

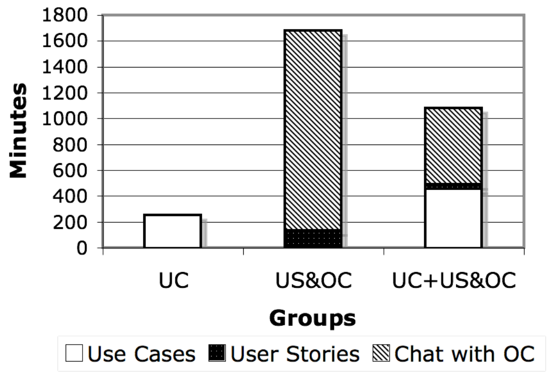

**Figure 1. Time subjects spent in each requirement format**

**Table 2. Time spent understanding requirements**

| Average/Group | UC (mm:ss) | US&OC (mm:ss) | UC+US&OC (mm:ss) | p value |
|---|---|---|---|---|
| Time reading Use Cases | 04:13 | - | 07:34 | p<0.05 |
| Time reading User Stories | - | 02:12 | 00:37 | p<0.05 |
| Time asking relevant questions to the OC | - | 22:46 | 09:12 | p<0.05 |
| Time asking irrelevant questions to the OC | - | 03:05 | 00:37 | p<0.05 |
| Total time understanding requirements | 04:13 | 28:03 | 18:00 | p<0.05 |

The time spent understanding the requirements can be divided into two parts, reading and chatting with the Customer. A summary of the time spent can be found in Table 2. Subjects who were in the UC condition did not have the opportunity to talk to a Customer, and this was the main reason that they spent the least time understanding the requirements. However, all three groups spent time reading the documentation that they were given. The Agile Requirements group spent more time reading User Stories than the UC+US&OC group who were given all the formats. This is understandable, because the US&OC only had access to these short descriptions. On the other hand, it is surprising that the

UC+US&OC group spent the most time reading the requirements of all the three groups. They even spent more time reading the Use Cases than the UC group (7:34 minutes vs. 4:13 minutes). This difference was found to be statistically significant at $p<0.05$ using the Kolmogorov-Smirnov test for two independent samples [1]. This difference can be attributable to the availability of the On-Site Customer and not the User Stories, because subjects in the third group spent a scant 37 seconds reading the latter. We now examine the chatting portion of the time spent understanding the requirements.

We found that subjects in US&OC condition spent more time chatting with the On-Site Customer than those in the UC+US&OC condition (25:51 minutes vs. 9:49 minutes). This result is statistically significant at $p<0.05$. While it appears that Agile Requirements are less efficient, in reality this time included requirements elicitation activities that were not needed in the other two conditions. In other words, subjects in the US&OC condition had to talk to the Customer just to catch up with the other two groups in terms of knowledge. While the subjects in the UC+US&OC group spent less time chatting, they made better use of it by asking fewer relevant and irrelevant questions ($p<0.05$ by Komolgorov-Smirnov). The average counts of the questions asked by the two groups are presented in Table 3.

**Table 3. Number of relevant and irrelevant questions asked to the on-site customer**

| Average/Group | US&OC | UC+US& OC | p value |
|---|---|---|---|
| Number of relevant questions to the OC | 6.00 | 4.00 | p<0.05 |
| Number of irrelevant questions to the OC | 1.67 | 0.33 | p<0.05 |

Table 4 shows the partial and overall score on the implementation tasks. Overall, the differences between the groups were not statistically significant. Although there are numerical differences between the average performance of each group, the variation could be explained by chance alone.

We broke down the performance score into sub-parts to determine if one group did better than another in a particular part of the implementation. While the UC Group had the highest average score on validations and messages, none of the differences in the sub-parts were statistically significant.

**Table 4. Partial and overall scores on tasks**

| Average/Group | UC | US&OC | UC+ US& OC | p value |
|---|---|---|---|---|
| Functionality score | 18.17 | 19.17 | 17.67 | n.s. |
| Validations and messages score | 5.33 | 1.33 | 1.67 | n.s. |
| Overall score | 23.50 | 20.50 | 19.34 | n.s. |

## 3.3 Interpretation of Results

Our experiment produced two important findings. The first is that subjects in the third condition (UC+US&OC) spent more time reading Use Cases than subjects in the UC condition, but spent less time than subjects in the US&OC condition understanding the requirements. This difference can be attributed to the availability of the On-Site Customer, which meant that subjects had to study the Use Cases and understand them well enough to ask questions about them.

The second finding is that there is no clear link between the format in which the requirements were presented and how well subjects scored on the implementation task. Because we failed to reject the null hypothesis, it is unknown if this is an actual effect. If our conclusions are true, it means that efforts made to improve requirement formats will not benefit software engineers. However, we doubt about the veracity of this implication because of the decisions we made in the experiment design.

We expected that subjects who spent more time understanding requirements would perform better in the implementation tasks. In addition, we expected that subjects using more requirement formats at the same time would also perform better because they would have more information available. Thus, we believed that the requirement formats which subjects used would have an impact on their performance. Contrary to expectations, our results showed that there was no link between the time subjects spent understanding requirements and their performance. In addition, our results showed that subjects using the most number of requirement formats scored the worst, though the difference was not statistically significant.

Not surprisingly, it appears that design decisions we made regarding limited resources had an impact on the validity of our results. We had a practical and doable experiment but our findings were not conclusive, only informative. In the next section, we will discuss some of the design decisions and their consequences.

## 4. TRADING OFF PRACTICALITY AND PERFECTION

We designed our experiment to measure the impact of requirements formats in the implementation of a software system, but had to take into account resource limitations. In particular, these were the availablity of subjects, qualifications of subjects, and duration of experiment sessions. Based on these limitations, we had to make choices to deal with difficult problems in the experiment design. As a result, these decisions were likely to affect the conclusions we were able to draw. These decisions made the experiment more feasible, but at the same time, also made the experiment less perfect and less ideal.

## 4.1 Subjects

The first scarce resource that we had to consider was the availability of qualified subjects. As a result, we had to make compromises in our sample size and our screening procedures.

### *4.1.1 Number of Subjects*

Our experiment was designed with three conditions to evaluate. We thought that having three subjects per each group and nine subjects in total was appropriate for an initial study and was enough to draw some conclusions.

Our decision to have a small number of subjects had the advantage that we could finish with the experiment faster and we are able to report our initial experience sooner. However, it has the disadvantage that we are not sure about our results and conclusions. Definitely, the small number of our subjects affected the generalizability of our results.

Another factor in the decision to use nine subjects was the effort required to run the sessions and analyze the data. Each session required two experimenters to run and required about three hours of their time. Each subject produced about 2.5 hours of screen, video, and audio recordings and other artifacts, which typically took a pair of researchers 4 or more hours to analyze, because we were collecting qualitative and quantitative data. In total, it took approximately 15 person hours to run and analyze each subject, which is not an inconsiderable number.

This small number of subjects meant that we had to use non-parametric statistics to analyze the data. This type of statistic relaxes assumptions about the distribution of the data, but at the cost of making it more difficult to achieve statistical significance.

Clearly, more subjects are needed in order to produce stronger results. Published software engineering experiments typically use a sample size in the mid-teens, though this figure can range from a handful to three dozen. Power analysis suggests that to achieve $\beta=0.95$ a total of 96 subjects are required (32 per condition), a truly infeasible number. Once again, we will need to make design decisions that trade-off resource constraints.

### *4.1.2 Qualifications of Subjects*

When recruiting subjects, we found that many of the potential subjects had knowledge of Java, but not of JSP and Servlets. We decided to accept these subjects and included JSP and Servlets tutorials and a familiarization task. It was very difficult to find nine subjects willing to spend 2.5 hours on the experiment. It was not practical to add another filter in the selection of subjects. We expected that a small number of our subjects would not be able to complete the implementation task. For that reason, we had the option of redirecting subjects who were struggling with the implementation technology to a design task.

The advantages of this decision were that we were able to recruit nine subjects and conduct the experiment within two and a half hours.

On the other hand, there were also disadvantages. First, we had a low rate of task completion; only one of our nine subjects finished the implementation task, which likely affected our results. One possible reason could be the level of difficulty of the implementation task. Subjects had to add a new attribute to the shopping cart to count items, allow this new attribute to be modified, and allow deletion of items in the shopping cart. We felt that this task was relatively straightforward, and subjects agreed. They were asked to rate the level of difficulty of the task and they assigned on an average 2.83 out of 5, 0 being easy and 5 difficult. We believe that the low task completion rate was caused by the low level of expertise in JSP and Servlets that our subjects had.

Second, depending on the completeness of the implementation task, we scored different artifacts. We scored the source code in case the subject completed some parts of the implementation task and also scored the design for other parts when the implementation was not completed. If a specific feature was completed, we assigned the same score for it without caring if the feature was completed in the implementation or in the design. However, it is possible that we could have been mixing apples with oranges when we equally scored the implementation and the design. We think that probably this equal scoring could not be fair in some cases because not all the subjects spent the same time implementing and designing. We asked our subjects to switch to designing at different times for each subject, depending on the difficulty they were having with the implementation.

## 4.2 Duration of Experiment Sessions

The second scarce resource that we had to manage was the length of the experiment sessions. There are limits to how long a subject can focus and work intensively on a task. As well, there are fewer people who are available and willing to commit to longer experiments. Our experiment sessions were 2.5 hours long, a duration that pushed these limits. Consequently, we had to make design decisions about what we asked subjects to do in the time available. These decisions had effects on the equivalence of subjects in the three conditions, and on the software tools and information that we gave them.

### *4.2.1 Equivalence of Groups*

Our experiment had three conditions to which we needed to assign the same number of subjects. Furthermore, we wanted to ensure that each group was comparable in terms of subject characteristics, background, and experience. Counterbalancing the groups helps to ensure that the performance of the groups can be compared.

Ideally, we would have assigned our subjects to groups based on tests of their cognitive traits and familiarity of JSP and Servlets. For example, we could have postponed our decision of assignment of subjects after having the results of the background questionnaire including quantitative information about their experience and background. The tests would have to assess knowledge and skills, and not just ask about how much prior experience the subjects had. The number of years of experience of subjects is known not to be a good measure of expertise. We found that four of our subjects said that they worked with Java web technologies for two years, but only one of them was able to finish the implementation tasks. However, adding more tests was not feasible, because it would have made the sessions too long. A skills test would have added 30 minutes and a personality test would have added 30-60 minutes.

Instead, we decided to assign three subjects to each group based on our knowledge of the expertise and background of our subjects. We were able to do this because we recruited our subjects by word-of-mouth. Before conducting the experiment we already contacted the nine subjects and we knew about the background and the expertise of some of them. Other subjects were asked informally about their familiarity with JSP and Servlets and background before scheduling the appointment for the experiment. Having this information, we assigned our subjects to each group before running the experiment and tried to have a balance of expertise and background in each group.

The design decisions we took had some advantages, for example before starting the experiments we knew that we had three subjects in each group and that this number of subjects will be enough to have balanced groups. Another advantage is that we did not need to include any additional test that could have increased the length of our experiment.

The main disadvantage of our approach was the groups created were not ideal, in the sense they were not completely balanced, and this likely affected the results of the study. As well, we were not able to control for personal traits, such as analytic ability, in analyzing the data. However, it was a reasonable trade-off, given the alternatives.

### 4.2.2 *Stimuli Given to Subjects*

Since this was a software engineering experiment, the subjects had to work with many different technologies in order to complete the maintenance tasks. We knew that we could not assume that all the subjects had worked with them previously, so we had to include time in the schedule for subjects to become familiar with the various tools and languages. We tried to reduce the technologies that subjects were required to use in order to save time and this affected the generalizability of the study.

Subjects in all three conditions had to use software tools (the Eclipse workbench and the Tomcat Application Server), programming languages and frameworks (Java, JSP, and Servlets), a data format (XML), and many conventions and best practices. Depending on the condition, subjects also had to work with Use Cases, User Stories, and an On-Site Customer via chat.

We decided not to include Test Cases with the material given to the groups using agile requirements for two reasons. One, we felt that Test Cases would have provided too much information and the comparison between the three conditions would have been too imbalanced and unfair. Two, we did not want to require our subjects to use yet another tool. Including a testing tool would have further increased the length of each experiment session.

In retrospect, this was not a good decision and the reasons were not well founded. Excluding Test Cases made conditions using Agile Requirements less realistic, and in turn, less generalizable. The prevailing view is that the trio of User Stories, On-Site Customer, and Test Cases form the core of Agile Requirements. The omission of test cases affected the credibility of study among agilists. We had assumed that we needed to provide the Test Cases in an automated testing tool, another common practice in Agile. We felt that this would have done too much of the work for those subjects using Agile Requirements, but at the same time required them to learn another tool. Looking back, we could have provided the written descriptions of the Test Cases, e.g. input, output, preconditions, to the US&OC and UC+US&OC conditions. This would have made the three conditions more similar in terms of the information given to them.

## 5. CONCLUSIONS

Making design decisions to implement a software system is not an easy task. Software designers have to evaluate the tradeoffs of each decision before selecting an option. Similarly, researchers need to evaluate different ways to design the experiment taking into account the resource constraints.

In this paper, we discussed the design decisions that had an effect on the validity of a small controlled experiment aiming to measure the impact of different requirement formats on how people implement a system. The limited resources that we were attempting to manage were the availability of qualified subjects and the duration of the experiment sessions.

Because it is very difficult to find qualified subjects, we decided to perform the study with nine subjects who had previous experience developing software using the Java programming language. The small sample size affected the power of the experiment, and meant that we could only use non-parametric statistics. The subject system in our study was a web application using JSP and Servlets. We did not screen for prior experience with these technologies and only one of our nine subjects were able to complete the implementation tasks. The poor scores of our subjects on this task led to inclusive results on the effect of the requirements formats on how well subjects implemented the change tasks.

The other constraint discussed in this paper was the duration of the experiment sessions. In our study, the sessions lasted 2.5 hours and included a background questionnaire, tutorials, a familiarization task, experiment tasks, and a debriefing interview. There were other tests and stimuli that we considered including, but did not.

Adding tests of skill and knowledge level in web technologies would have allowed us to make the groups in each conditions more similar to each other. Adding tests of personality traits and cognitive ability would have allowed us to control for the effects of these factors when analyzing performance. However, there simply was no time available to add these to the schedule.

We do regret one design decision that we made with respect to time constraints; we did not provide Test Cases to the groups using Agile Requirements and in retrospect we should have. We originally felt that we could not burden these subjects with yet another tool or format, but this was a poor decision, because it decreased the credibility of our experiment especially among Agilists.

In summary, these design decisions ensured that we had a study that was feasible, but at the cost of some threats to validity. It would not have been possible to conduct an ideal experiment. Instead, we had an imperfect experiment that shed light on a phenomenon, the effect of requirements formats on maintenance tasks. The result was a practical experiment, but our results are not conclusive, merely informative, which still allows us to make incremental progress as a field.

## 6. REFERENCES

[1] Gallardo-Valencia, R., Olivera, V., and Sim, S. Are Use Cases Beneficial for Developers Using Agile Requirements? In *Proceedings of the Fifth International Workshop on Comparative Evaluation in Requirements Engineering (CERE'07)*, India, 2007. To appear.

[2] Hall, M., *More Servlets and JavaServer Page*. First Edition. Sun Microsystems Press Publisher, 2001.

[3] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., Emam, K.E., and Rosenberg, J. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*. Vol. 28, No. 8 (Aug. 2002), 721-734.

[4] Perry, D., Porter, A., and Votta, L. Empirical Studies of Software Engineering: A Roadmap. *Future of Software Engineering*. ACM. 2000. 345-355.

[5] Pfleeger, S.L. Experimental Design and Analysis in Software Engineering. *Annals of Software Engineering 1*. Baltzer Science Publishers. The Netherlands. 1995. 219-253.

[6] Sheskin, D. Handbook of Parametric and Nonparametric Statistical Procedures. Second Edition. Boca Raton: CRC Press, 2000.