Are Use Cases Beneficial for Developers Using Agile Requirements?

Rosalva E. Gallardo-Valencia
University of California, Irvine
rgallard@ics.uci.edu

Vivian Olivera
University of California, Irvine
volivera@ics.uci.edu

Susan Elliott Sim
University of California, Irvine
ses@ics.uci.edu

Abstract

Agile teams commonly use User Stories, conversations with Customers, and Test Cases to gather requirements. Some agile teams like to add other artifacts, such as Use Cases to provide more detail to the agile requirements. This paper presents the results of a controlled experiment aimed to find evidence that Use Cases could help agile requirements, and, indirectly, to find if agile requirements techniques are sufficient. In the study, subjects were given requirements for three maintenance tasks as Use Cases, as agile requirements, or both. We found that subjects using Use Cases spent less time understanding requirements in contrast to subjects not using Use Cases. In addition the presence of the Use Cases helped subjects to ask better questions to the On-site Customer. However, we could not determine if subjects using Use Cases understood the requirements better. We conclude that the inclusion of Use Cases in agile requirements could benefit agile teams.

1. Introduction

Extreme Programming (XP) [1] is a popular Agile Software Development Processes [2]. In this process, requirements are not written in a formal requirement specification document. Instead, the requirements are elicited in User Stories [3], which is a description of a feature from a customer's standpoint that is 1-3 sentences long. A common misconception about agile requirements techniques is that they are not sufficient for gathering requirements, because they do not record enough details. This is not correct, because details about User Stories can be found in conversations with the On-site Customer [4] and also in the Test Cases. Although requirements are spread among these three entities and are not compiled into a formal document, these techniques seem to work in practice.

User Stories are flexible and can be complemented by adding Use Cases, Graphical User Interface (GUI) sketches, or other artifacts required by agile teams. In contrast, we have the Use Cases [5], which have been widely used by software engineers. They have many benefits and a great number of software engineers know how to use them. Most importantly, they provide a detailed record of the requirements. We want to investigate if the use of Use Cases can be beneficial for agile teams. We want to know if adding Use Cases is helpful. Furthermore, answering this question will provide insight into whether User Stories are enough to gather requirements.

With this goal, we conducted a controlled experiment involving a small number of subjects with three conditions. In the experiment, subjects had to make a modification to an existing software system with the requirements for the changes specified as Use Cases, as agile requirements (User Stories and with access to an On-site Customer) or both.

We recorded each subject's sessions in video. We also had the chat transcripts of those subjects who had access to the On-site Customer. We used this data to find some evidence that Use Cases could be helpful for teams using User Stories.

We found that agile subjects using Use Cases spent less time on understanding requirements and they asked better questions compared with the agile subjects not using Use Cases. However, we could not find conclusive results regarding the effect of the requirements formats on implementation activities.

Subjects in Group 2 who were using agile requirements alone spent more time reading the User Stories than subjects in Group 3 who were using both types of requirements formats. We observed that this occurred because subjects who where using both Use Cases and User Stories preferred to read the Use Cases, because they provided more detail.

Subjects in Group 2 who were using agile requirements spent more time asking relevant as well as irrelevant questions, and also asked a greater number of relevant and irrelevant questions, than subjects in Group 3. However, subjects in Group 3

asked a higher proportion of relevant questions. This group also asked better questions.

With respect to the effect of requirements format on implementation, we did not have any conclusive results. We found that subjects in the group using agile requirements and Use Cases had a poorer overall performance, but this result was not statistically significant.

The remainder of the paper is organized as follows. Section 2 introduces Use Cases, User Stories, and the On-site Customer and reviews previous work on the relationship between Use Cases and User Stories. Section 3 presents the method used in our empirical study. Our results are described in Section 4. The discussion of our results is given in Section 5. Section 6 discusses future work, followed by our conclusions in Section 7.

2. Background

In this section, we review the requirements formats used in this study. As well, we discuss existing studies that compare agile requirements with plan-based requirements techniques.

2.1. Agile Requirements

In recent years, Agile Software Development Processes have been gaining acceptance in the software engineering community. These processes are based on the Agile Manifesto [6]. The principles behind this manifesto suggest that changing requirements should be welcomed in every stage of the development, that working software should be delivered frequently, and that conveying information via face-to-face conversation is more efficient than through written documentation.

Agile Processes are characterized as informal and unwritten. In addition, these processes put more emphasis on the verbal and social communication in the development team. In contrast, traditional processes that are sequential and phased emphasize formal and written work and communication.

User Stories are composed of three aspects: a written description of the feature or to-do item used for planning; conversations about the story that serve to flesh out the details; and tests cases that convey the details [3]. Usually User Stories are written on 3-by-5 index cards.

User Stories are used as a unit of work and agile teams plan their releases by scheduling a set of User Stories for completion in each iteration. User Stories are also a promise of conversations between developers and customers. It is important to note that User Stories by themselves do not document requirements; they represent requirements whose details are found in conversations with the On-Site Customer and in Test Cases.

User Stories are used by Agile teams and especially in Extreme Programming (XP) [1], which is one of the more practiced agile methodologies. XP is based on 12 practices; the ones that are related with our research are the Planning Game, the On-site Customer, and the Test-Driven Development.

During the Planning Game, customers write what they want the system to accomplish in the form of User Stories. Then, developers estimate how much time it will take them to implement these User Stories. With the estimates and the velocity of the developers, the customers prioritize the User Stories and choose which ones will be completed in an upcoming iteration. User Stories will be assigned to a developer or a pair of developers to be implemented. During the implementation, developers are expected to have questions regarding the User Stories and to answer them by talking face-to-face with the On-site Customer [4]. XP suggests having a real customer working side by side with the development team. This customer will be responsible for answering questions raised by the development team.

Implementation in agile processes is done using Test-Driven Development, which mandates that automated test cases based on the User Stories be created before the source code is written. Starting with a system that fails all the new test cases, developer implement just enough code necessary to pass the tests.

User stories should be written without using any technical jargon. They should be understandable by the business people and their content should fit on an index card. They could be explained in 30 seconds. It should be possible to complete them in less than one week. They should be easy to translate into a test. It is common to use the pattern "As <role> I can <function> so that <business value>."

One example of a User Story is: "As a Buyer, I can modify the quantity of each item in the cart." We used this User Story in our experiment. As we can see in the example there are no details about validations, error messages, and exceptional paths. However, subjects who were given User Stories had access to an On-site customer via chat, to whom they could ask all the questions related to the feature.

Table 1. Use case example

USE CASE 5		Modify Quantity
Goal in Context		Modify quantity of the items already in the cart
Preconditions		Buyer has pressed the "Modify Quantity" button
Success End Condition		Buyer has successfully changed the quantity of the items in the cart
Failed End Condition		The user could not change the quantity of the items in the cart
Primary, secondary Actors		Buyer
Trigger		"Modify Quantity" button is clicked
DESCRIPTION	**Step**	**Action**
	1	The system shows a table with the following columns: Item Number, Description, Quantity, and Cost. The quantity should be editable
	2	The user enters the new quantity for the items and press the button "Update Cart"
	3	The system verifies that all quantities are integers greater than zero
	4	Use "Use Case 8. View Cart"
EXTENSIONS	**Step**	**Branching Action**
SUB-VARIATIONS		**Branching Action**
	3	If the quantity is lower than zero or is not an integer. The system should show an error message "The quantity should be an integer greater than zero."

2.2. Use Cases

Uses Cases are used extensively in plan-based software processes, such as the Rational Unified Process (RUP) [7]. This format has the goal of describing the set of interactions and events between the users or external systems (also known as actors). These descriptions include the functionality the system is required to meet. There are different guidelines to write Use Cases and the effectiveness of a Use Cases depends on the author's ability to write them.

An example of a Use Case is showed in Table 1. We used this Use Case in our experiment and it corresponds to the User Story presented in the previous subsection. As can be seen in Table 1, a Use Case contain the name, goal, preconditions, success end condition, failed end condition, primary and secondary actors, trigger, description of each step in the main scenario, description of each step in the extensions, and the sub-variations. In this case, the Use Case has specific information about the steps the user should follow to use successfully the new feature. There were also included details such as name of buttons, specific validations (for example, the new quantity should be greater than zero), and error messages.

2.3. Comparison

Use Cases are longer than User Stories; they can vary between two paragraphs and ten pages. They are good for showing the alternate paths of a specific feature. This could also be achieved by User Stories by writing the exceptional paths in different User Stories, but this approach is an adaptation of the technique, rather than a planned usage, which contrasts with Use Cases.

Table 2. Comparison of use cases/scenarios and user stories

Use Cases/Scenarios	User Stories
They are expressed using a constrained (semi-formal) syntax	They are expressed using natural language prose
They are specifications of object interactions	They are descriptive and expressive of human desires
They contain "how"	They contain "what" and "why"

Usually, User Stories will not be sufficient in an organization where formal documentation is mandatory. Their main difference with Use Cases or Scenarios is that User Stories have the goal of capturing the perspective that the user has about the system. Some of the differences between Use

Cases/Scenarios and User Stories presented by Beck and West [8] are shown in Table 2.

Another requirements format that is widely practiced and mentioned in the literature is Scenarios. Some times, the terms Use Cases and Scenarios are used interchangeably. However, we will stick with the definition given by Salinesi [8] that "A Use Case is always composed of several scenarios that describe alternative ways to try and achieve the goal." In this view, scenarios are part of Use Cases, in this study we evaluate the Use Cases, which consist of Scenarios.

As we have seen, each requirement format has some benefits and drawbacks. It is not clear if Use Cases complement or are redundant to User Stories in Agile Development. The goal of our study is to provide some evidence to guide Agile teams regarding whether or not there could be any benefit in using Use Cases in addition to the requirement formats they are currently using.

2.4. Related Work

A number of studies have been performed comparing traditional and agile requirements. In general, they found that the two approaches are complimentary. Paetsch et al. [9] presented a comparison of traditional requirements engineering approaches and agile software development. He concluded that agile methods and requirements engineering are pursuing similar goals in key areas and the main difference between them is the amount of documentation created in the project.

Eberlein and Sampaio do Prado Leite [10] presented a position paper that discussed the applicability of requirements engineering to agile processes. They argued that four practices (Customer Interaction, Analysis, Non-Functional Requirements and Managing Change) should be added to agile requirements in order to assure quality in the produced software.

Meszaros [11] wrote an experience report to propose four "storyotypes" (story stereotypes) based on Use Cases to be used as guidelines to split large User Stories. Meszaros based the proposed storyotypes on Use Cases because Uses Cases are the best understood and most used format of requirements expressed in prose. He is especially concerned that teams members who have had previous experience using Uses Cases will experience some difficulties creating User Stories. Such developers are used to working with Use Cases that can have many scenarios and are more likely to create a big User Story containing one Use Case.

Imaz and Benyon [12] studied how User Stories and Use Cases can be used together to better capture interactions during requirements gathering. They concluded that User Stories are effective for capturing interaction, but Use Cases are needed for implementation purposes where formal documentation is required.

However, there has not been a controlled experiment to compare the effectiveness of adding Use Cases to agile requirements, such as the one described in this paper.

3. Method

We conducted an initial controlled experiment on a small sample of software engineers to find evidence regarding whether or not the Uses Cases could be beneficial for teams using agile requirements. We had three conditions in the experiment. In the first condition subjects were using Use Cases to understand requirements, in the second condition subjects were using agile requirements (User Stories with access to an On-site Customer), and in the third condition subjects were using agile requirements and Use Cases.

We had a total of nine subjects assigned to the three conditions. Subjects were assigned to each condition to counterbalance the level of experience in each group. We asked our subjects to modify an existing feature and to add two new features to a web-based shopping cart application for purchasing boats. Subjects received the requirements in different formats and were asked to implement the tasks specified. In addition, subjects were asked to 'think aloud' while they worked to provide us with additional insight into their behavior.

3.1. Experiment Design

We had three conditions in our experiment. In the first condition, our subjects were using Use Cases as a requirements format. We will refer to the set of subjects in this condition as the UC Group. In the second condition, subjects were using agile requirements, more precisely User Stories and had access to an On-site Customer via chat. This condition will be referred as the US&OC Group. Finally, subjects in the third condition were using both of the above two requirement formats. This last condition is called UC+US&OC Group from here onwards. This design would allow us to perform side-by-side comparisons of Use Cases and agile requirements, as well as comparisons against usage of both formats together.

We decided not to include Test Cases with the material given to the groups using agile requirements for two reasons. One, 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 increased further the length of each experiment session, which was already two and a half hours. We provided access to an On-site Customer via chat (instant messaging). This part of the design is similar to the approach used by Shukla and Williams [13]. They presented a study where they integrated the Extreme Programming practices into their courses at North Carolina State University. They used the User Stories practice, but they also completed several Use Cases and discussed this requirement format as an alternative to User Stories. They also applied the On-site Customer practice, having the customer available through email and not literally 'On-site.' Our method is an improvement, because customer response was available in real time.

We expected that subjects using the agile requirements (User Stories and access to an On-site Customer) and the Use Cases (UC+US&OC Group) would perform the best among the three groups. We believed that because those subjects have more information and more details, this would result in better understanding of the requirements and thereby better performance. We also felt that subjects who spent more time trying to understand the requirements would perform better than the others. This would include reading the requirements from the Use Cases or User Stories or extracting details regarding the requirements from the On-site Customer.

3.2. Procedure

The experiment consisted of four activities: a background questionnaire, tutorials and familiarization tasks, the maintenance tasks and the design, and finally the debriefing interview. Only the maintenance tasks and the design were timed and the total duration of each run of the experiment was around two and a half hours. Table 3 shows the schedule of the experiment including expected times per activity.

Table 3. Schedule of experiment

- Background Questionnaire	~10 minutes
- Tutorials - Familiarization Task	~10 minutes
- Maintenance Tasks (3 tasks) - Design	~120 minutes
- Debriefing Interview	~10 minutes
Total	**~150 minutes**

Background Questionnaire We asked our subjects to fill out a background questionnaire regarding their education, software development experience, and familiarity and preferences of different requirement formats.

Tutorials and Familiarization Task The goal of these tasks was to familiarize our subjects with the requirements format that they would be given as well as the programming environment. Hence, we provided a Use Case tutorial to the UC Group, a User Story and an On-site Customer tutorial to the US&OC Group, and both the above tutorials to the UC+US&OC Group. The Use Case tutorial included an explanation of the template used including the purpose and meaning of each section in the template. The User Stories and On-site Customer tutorial included an explanation of what User Stories are, how they work, and what the format is. The explanation of the role and responsibilities of an On-site Customer and an example of a User Story was also provided.

After finishing the tutorials, we asked our subjects to implement a "List of Courses" JSP page using the Eclipse IDE. The task required subjects to make a modification and compile a Java file with the list of courses, and a JSP page. Step-by-step instructions were given for completing the familiarization task .

Maintenance Tasks and Design The main task in our experiment lasted for approximately two hours. The requirements were for a change to an existing feature and the addition of two new features to the system. The details of the tasks are discussed in section 3.4.

When we observed that a subject was struggling with the task due to lack of knowledge of the underlying technology, e.g. developing web applications in Java™ using JSP (Java Server Pages) and servlets, we re-directed them to produce a design for modifications. We asked the subjects to draw the design as screen captures to explain the functionality. The main goal of the design exercise was to evaluate how well the subjects understood the requirements when they were not able to complete the coding of the implementation.

Debriefing Interview After the allotted two hours for implementation had elapsed, we proceed with a debriefing interview where we asked the subjects open-ended questions regarding their performance and their experiences using the requirement formats during the experiment. We also asked them about their opinions and any feedback regarding the experiment.

3.3. Subjects

A total of nine subjects participated in our experiment. Seven of our subjects were graduate students, one was a research assistant, and one was an undergraduate student. Details about our subjects are summarized in Table 4.

Table 4. Characteristics of subjects

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

3.4. Subject System and Implementation

The application used in this study was a web-based system to purchase boats over the Internet called "An Online Boat Shop." The system is an example obtained from the book "More Servlets and JavaServer Pages" by Marty Hall [14]. We chose this system because it was of medium complexity, it used simple Java web technology, it was complete in itself and it did not require interaction with any additional external database. It uses the JSP and Servlets technology and runs on a Tomcat application server.

The Online Boat Shop consists of 12 Java™ files and 10 JSP files. In addition, it has an XML (extensible Mark-up Language) file with the configuration of the web application. There were 1,340 lines of source code. The application is well structured and the source code is well formatted.

We asked our subjects to modify a feature in the system and add two new features. Our requirements consisted of three subtasks:

Maintenance Task A   This task involved modifying an existing feature on the website. Subjects were asked to add a field whereby one could see the quantity of each item in the shopping cart.
In the existing implementation, new boats were added as new rows in the shopping cart display, even when the same kind of boat was already in the cart. Subjects were asked to add a "Quantity" column to the shopping cart, to maintain one row per unique item, and to increment the quantity column whenever an item that already existed in the cart was added. For the groups

using User Stories, the following User Story was provided: "As a buyer, I can see the quantity of each item after adding an item to the cart."

Maintenance Task B   This task required the addition of a new feature to the website. Subjects were asked to implement a modification of quantity feature. The Use Case for this feature was given in Table 1. The subjects had to include error handling, such as allowing users to modify quantities to only positive integers. For the groups using User Stories, the following User Story was provided: "As a buyer, I can modify the quantity of each item in the cart."

Maintenance Task C   This task also required the addition of a new feature to the website. Subjects were asked to implement a deletion of items feature. The requirements in the Use Case format asked the subjects to provide a feature whereby users could delete existing items from the shopping cart. The subjects were also told to incorporate a 'Confirm Deletion' message to prevent users from accidentally deleting items and to also provide a feature where an error message was thrown if the "Delete Items" button was clicked but no items were selected for deletion. For the groups using User Stories, the following User Story was provided: "As a buyer, I can delete items from the cart."

3.5. Threats to Validity

We are aware that this controlled experiment has some limitations. First, the number of subjects in our study was small, only nine. More subjects would improve the external validity of the study. However, this number is sufficient for a preliminary study.

Second, our subjects did not have enough experience with the technology used in the experiment. Only one out of our nine subjects was able to finish the implementation task. The rest of them were unable to complete the task due to a lack of knowledge of JSP and servlets. This threat is relatively serious, but we attempted to mitigate this problem by including a design task.

Third, a single researcher scored the source code from the implementation and the design. This may have introduced a bias into the data, but one benefit is it gave us a consistent scoring for all the subjects. This risk has not been mitigated in this study, but in future, multiple raters could be used and inter-rater reliability could be measured.

Four, we found that the analytical ability of our subjects is a key factor on how well they understood the requirements. However, we did not have a way to

measure or quantify the analytical ability of our subjects.

Although our study has some limitations, the results obtained represent the findings of an initial study and provide us with some useful empirical data to evaluate the benefit of Use Cases for agile requirements. As well, we controlled many factors in the study, so it was a fair comparison.

4. Results

The results of the experiment indicate that Use Cases complemented agile requirements by helping subjects to spend their time more efficiently when understanding requirements. However, our data do not provide enough evidence to state whether any group created better implementations than another.

We will present our results in two categories. First, we will report our data regarding the subject's performance on requirements. This result has the purpose of showing how our subjects spent their time on requirements. Then, we will report our data related to the overall task performance. This result has the goal of evaluating how well subjects understood the requirements. We tested our data using non-parametric statistics. This kind of statistical methods 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.

4.1. Performance on Requirements

We watched the video recordings of the experiment and measured the time subjects spent interacting with the requirement formats. We found that agile subjects using Use Cases (UC_US&OC Group) spent less time understanding requirements than subjects using only agile requirements.

We compared the time spent by the three groups understanding the requirements. In the case of the UC Group, we considered the total time understanding requirements as the time spent reading the Use Cases. For the US&OC Group, we included the time spent reading the User Stories and the time chatting with the On-site Customer. For the UC+US&OC Group, we included all the time reading and chatting with the On-site Customer.

The average time spent by each group in understanding requirements is shown in Table 5. To have a better idea about how subjects spent their time, Figure 1 shows graphically how much time subjects spent in each requirement format.
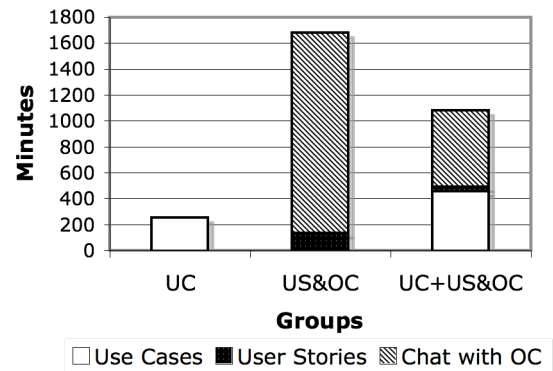
Figure 1. Time subjects spent in each requirement format

We found that on average subjects in the US&OC Group spent the most total time (28 minutes and 3 seconds) understanding the requirements, the US+UC&OC Group was second (18 minutes), and the UC Group the least (4 minutes and 13 seconds0. This difference was found to be statistically significant at $p<0.05$ using the Kruskal-Wallis one-way analysis of variance by ranks [15].

Overall, subjects spent little time reading User Stories. The second entry in Table 5 shows that on average subjects in the US&OC Group spent more time (2 minutes and 12 seconds on an average) reading the User Stories than subjects in the UC+US&OC Group (37 seconds on an average). This difference was found to be statistically significant at $p<0.05$ using the Kolmogorov-Smirnov test for two independent samples [15].

Having access to an On-site Customer in the UC+US&OC condition had the interesting effect of causing subjects to spend more time reading the Use Cases. This result can be found in the first row of Table 5. We observed that subjects on average in the UC+US&OC Group spent more time (7 minutes and 34 seconds) reading the Use Cases than subjects in the UC Group (4 minutes and 13 seconds). This difference was found to be statistically significant at $p<0.05$ using the Kolmogorov-Smirnov test.

While the US&OC spent the most time communicating with the On-Site Customer ($p<0.05$ by Kolmogorov-Smirnov) their questions were not as good as those in the UC+US&OC Group. The third and fourth entries in Table 5 show the average time that subjects spent asking relevant and irrelevant questions respectively. We observed that on average subjects in the US&OC Group spent more time overall asking questions.

Table 5. Time spent understanding requirements

Average/Group	UC (mm:ss)	US&OC (mm:ss)	UC+US&OC (mm:ss)	P
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

Table 6. Number of relevant and irrelevant questions asked to the on-site customer

Average/Group	US&OC	UC+US&OC	p
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 7. Partial and overall scores on tasks

Average/Group	UC	US&OC	UC+US&OC	p
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.

During that time, the US&OC Group on average had to ask more questions and a larger proportion of these were irrelevant (p<0.05 by Komolgorov-Smirnov). The average number of relevant and irrelevant questions asked to the On-site Customer is shown in Table 6. Subjects in the US&OC Group asked on average a greater number of relevant (6) and irrelevant questions (1.67) than subjects in the UC+US&OC Group (4 and 0.33, respectively).

4.2. Overall Task Performance

We also collected data from the coding and design by our subjects to provide an objective, performance-based measure of how well they 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 for either was 30 points.

Overall, the differences between the groups were not statistically significant. Although there are numerical differences between the average performance for each of the groups, 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.

5. Discussion

In this section, we will discuss and interpret the results presented in the previous section. In addition, we will also discuss other results collected from the Debriefing Interview.

5.1. Performance on Requirements

The subjects' performance on the requirements tasks were mixed and, at times, contrary to expectation. Subjects using agile requirements spent more time understanding the requirements than subjects using Use Cases. This time difference can be attributed to the need to elicit details through questions instead of simply reading them from a document. However, many software engineers have professed a preference for talking to people instead of reading. As well, conversations are more flexible and adaptable to change.

The most surprising result is the group using both Use Cases and agile requirements spent the most time reading the Use Cases. One would expect the UC group to spend more time reading the Use Cases because they were their sole source of information, but this was not the case. We believe that this difference in the time spent reading the Use Cases is because

subjects in the UC+US&OC Group used the information in the Use Cases to elaborate questions to the On-site Customer. They needed a deeper understanding of the Use Cases in order to frame their questions to the customer. On the other hand, subjects in the UC Group spent less time reading Use Cases because they only needed to understand the requirements and then implement them. They did not need to ask questions regarding them. This factor of being able to ask questions about requirements to the On-site Customer made subjects spend more time reading the Use Cases.

Probing deeper, we turn to data from the debriefing interviews. We also asked the subjects for feedback on the different requirement formats that were given to them. When asked if the Use Cases had enough information, all the subjects in the UC Group felt that they did, while all the subjects in the UC+US&OC Group felt that they did not. We believe that the subjects in the UC+US&OC Group disagreed, because they had the luxury of clarifying their doubts regarding the Use Cases with the On-site Customer. If they had only the Use Cases like the subjects in the UC Group, then perhaps they too would have felt that the Use Cases had enough information. However, it is interesting to note that all the subjects in the UC Group said that having an On-site Customer would have certainly helped them.

We observed that subjects in the US&OC Group spent more time reading the User Stories than subjects in the UC+US&OC Group. We believe that subjects in the US&OC Group spent more time reading the User Stories because it was the only written documentation that they had and they needed to ask questions based on the information in the User Stories. In contrast, subjects in the UC+US&OC Group did not spend too much time reading the User Stories because they preferred to read the Use Cases, which had more detail.

In the debriefing interview, we asked if the User Stories by themselves provide enough information. All the subjects in both the US&OC Group and the UC+US&OC Group said felt that they did not. They went on to elaborate that details of the implementation, such as special conditions and the flow of the application, were missing. However, they all felt that the User Stories in conjunction with the On-site Customer provided them with enough information.

We observed that subjects in the US&OC Group spent more time asking relevant questions as well as irrelevant questions than the subjects in the UC+US&OC Group. We believe that this took place because subjects in the US&OC Group did not have enough detail in the User Stories and they needed to ask the On-site Customer to elaborate about what was required. In contrast, subjects in the UC+US&OC Group had more details in the Use Cases and they did not need to ask so many questions.

Subjects in the US&OC Group asked a greater number of relevant and irrelevant questions than subjects in the UC+US&OC Group. We observed that our results on the number of questions asked to the On-site Customer are consistent with the time spent by each group communicating with the On-site Customer. Subjects in the US&OC Group did not only spend more time asking relevant and irrelevant questions, but also they asked a greater number of relevant and irrelevant questions to the On-site Customer.

When asked if the On-site Customer provided them with enough information, all the subjects in both the US&OC Group and the UC+US&OC Group answered affirmatively. When asked if they would prefer having the On-site Customer face to face, three out of six subjects said that they would prefer interacting with the On-site Customer face to face rather than through chat. Five out of six subjects said that they were comfortable with asking questions to the On-site Customer right from the very beginning. Only one subject said that initially he was a little uncomfortable asking questions to the On-site Customer, but as the experiment progressed the discomfort wore off.

5.2. Overall Task Performance

The results in the implementation/design task were also surprising. Our expectation was that subjects who spent more time understanding the requirements would be able to produce better implementations. We had also expected that subjects in the UC+US&OC Group would do better than subjects in the other two groups because they had more requirement documentation available and also had access to an On-site Customer. However, neither expectation was borne out. While there were statistically significant differences in the time spent, there was no statistically significant difference in the performance. In other words, the requirements format had no effect on how well the subjects completed the maintenance tasks.

This lack of a difference is very troubling because it brings into question the premise of software technology, that improvements in tools and methods can result in quality improvements for the software produced. Our subjective judgment, having observed the subjects as they worked, is that differences in individual skill, particularly their analytic ability, i.e. their ability to reason, had the greatest effect on their understanding of the requirements and overall task performance. Since we had only a small sample size and only one subject was able to finish the tasks, we leave this merely as a question.

Finally, our subjects showed a high level of self-awareness in terms of overall task performance. During the Debriefing Interviews, we asked the subjects to rate themselves on a scale of 1 (low) to 5 (high) on how well they understood the requirements. We found that subjects in all three groups were fairly accurate in estimating their scores. The largest difference between actual scores and self-rated scores was a 12.19% difference in the US&OC Group, as shown in Table 8.

Table 8. Actual scores and self-rated scores

	UC (%)	US&OC (%)	UC+US&OC (%)
Actual Score	78.33	68.33	64.44
Self-rated Score	80.00	60.00	66.66

In other words, our subjects had a good sense of how well they understood the requirements and the task, even if they couldn't articulate their errors precisely.

5.3. Use Cases vs. Agile Requirements

In comparing Use Cases and agile requirements, it appears that the two are complementary. Writing things down can save time—if people read the documents. Furthermore, being expected to formulate questions improves subject's willingness to read and attention when doing so.

Table 9. Strengths and weaknesses of the requirement formats

	Strengths	Weaknesses
Use Cases	• Gives context • Provides detail • Provides business logic	• Too much to read • Does not have UI
User Stories	• Provides a good overview • Gives the developer flexibility • Changes are easy to identify and implement	• Not enough detail • User Stories may overlap • No details on exception handling
On-site Customer (via chat)	• Can get quick answers to questions • Fills imple–mentation gaps • Customer can do other work as well	• Absence of physical presence may cause communication problems • Prefer talking to typing • Lack of writing skills may cause problems

We also asked the subjects as to what they felt were the strengths and weaknesses of the requirement formats that they were given. Their answers are summarized in Table 9. An examination of their answers also supports the contention that Use Cases and agile requirements are complementary, but more work is needed.

6. Future Work

It is often the case that research raises more questions than answers. If this is a measure of success, then we believe that we have made a contribution. We would like to continue this research and are considering some modifications.

We are planning to run the experiment with more subjects to improve the external validity of the study. In addition, we are also considering inclusion of a psychological test in the experiment, which will help us measure the analytical skills of the subjects to determine whether it is a confounding variable. If this is the case, this characteristic can also be used to counterbalance the assignment of subjects to conditions. As well, we plan to perform some evaluations of rater bias and reliability to improve internal validity.

Other improvements include the use of different colored sheets for different documents to help us in the video analysis, and the recruitment of subjects who are proficient in the technology used (JSP and servlets in this case). We also would like to be more consistent in the time we ask subjects to stop with the implementation and start with the design, if they are having problems.

7. Conclusions

Agile software teams occasionally use other requirements specification formats to complement their agile requirements techniques. We conducted a preliminary controlled experiment to determine if adding other information was helpful. This study would also indirectly answer the question of whether agile requirements techniques work effectively alone. Specifically, we wanted to know how the Use Cases, agile requirements (User Stories and On-site Customer), or both could affect the understanding of requirements for a maintenance task.

We found that Use Cases and agile requirements were complimentary. Subjects who had access to both spent less time understanding requirements than subjects who had access to only agile requirements. They also spent less time asking questions of the On-site customer, while asking a higher proportion of

relevant questions. As well, subjects who were planning to talk to an On-Site Customer spent more time reading Use Cases.

While we found statistically significant differences on how subjects used the different requirements formats, we could not find any statistically significant performance differences between the groups using different formats. In other words, the requirements format had no effect on how well they completed the maintenance tasks. The main factor affecting the understanding of the requirements appeared to be the analytical ability of each subject. We arrived at this conclusion because we did not find a clear relationship between the time subjects spent understanding the requirements and how well they performed implementing them. We found that subjects in the UC Group spent the least time understanding requirements, but they had the highest score in understanding requirements and implementing them. Similarly, subjects in the US&OC Group were the ones who spent the most time of all the groups in understanding the requirements but they did not have the highest score. Finally, subjects in the UC+US&OC Group had the least score even though they were not the ones who spent the least time understanding the requirements. We observed that it was the analytic ability of each individual subject that made the difference to the final score and the overall performance.

This study gives an affirmative answer to the question raised in the title of the paper. Based on the data from our preliminary study, use cases are beneficial for developers using agile requirements. But more research needs to be done to provide a more definitive answer and to answer the intriguing questions raised in this study about the effect of software technology.

8. Acknowledgements

9. References

[1] K. Beck, Extreme Programming Explained: Embrace Change, Mass: Addison-Wesley, 2000.

[2] A. Cockburn, Agile Software Development, Addison-Wesley, 2002.

[3] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley, 2004.

[4] K. Beck, "Embracing Change with Extreme Programming", Computer, 1999, vol. 32 pp. 70.

[5] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2001.

[6] M. Fowler and J. Highsmith, "The Agile Manifesto", Software Development, 2001, Vol. 9, pp. 28-32.

[7] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, 2003.

[8] I. Alexander and N. Maiden, Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle. John Wiley, 2004.

[9] F. Paetsch, A. Eberlein and F. Maurer, "Requirements Engineering and Agile Software Development", in Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), 2003.

[10] A. Eberlein and Sampaio do Prado Leite, J.C., "Agile Requirements Definition: A view from requirements engineering", in Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02), 2002.

[11] G. Meszaros, "Using Storyotypes to Split Bloated XP Stories", in Proceedings of the 2004 XP/Agile Universe Conference, 2004, pp. 73-80.

[12] M. Imaz and D. Beyon, "How Stories Capture Interactions", in Proceedings of Human-Computer Interaction - INTERACT'99, 1999, pp. 321-328.

[13] A. Shukla and L. Williams, "Adapting Extreme Programming for a Core Software Engineering Course", in Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02), 2002.

[14] M. Hall, More Servlets and JavaServer Page, First Edition. California: Sun Microsystems Press Publisher, 2001.

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