An Investigation of the Open Source Development Process

Elisa Yumi Nakagawa

Dept. of Administrative Science and Technology
Uniara - Araraquara University Center, Araraquara/SP, Brazil
elisa@icmc.usp.br**

Abstract. Recently, the open source process has sparked interest of researchers in the universities and in the software industry. So, our objective is to investigate if the open source process works effectively when developers do not have good knowledge about the functionalities of the software that will be developed. Thus, two studies have been performed; one to compare the "traditional" process and the open source process aiming to observe if analysis and design models are relevant to the software development; and other, to observe if a requirements specification gives benefits to the open source processes when developing a software with unknown or unclear functionalities. Both studies were formulated and documented according to experimentation process; and, the lessons learned can be considered as the first step to adjusting, establishing and transferring the best practices of the OS process to the software industry.

1 Introduction

The success of several OS (Open Source) projects has sparked interest of researchers over the past years. Naturally, in the face of this success, OSS (Open Source Software) and its development process have become interesting in the universities and commercial software organizations. Particularly, these organizations are interested in investigating if the practices behind OSS development process, or simply OS process, can be migrated into their practices.

The OS process has not been captured definitively in writings. However, Raymond has written the single best description called "The Cathedral and the Bazaar" [6]. Cathedral represents conventional commercial practices, where developers work using a relatively closed and centralized methodology to develop software. In contrast, Bazaar represents the openly cooperative effort of the OSS development, where the software is developed while requirements have been appearing. OS developers have usually good experience in coding and capability of cooperative distributed work. They frequently use version control systems and internet resources, e.g., Web sites and mailing lists. Furthermore, OSS developers are generally end-users of the software systems that they have developed; then, they have good knowledge about the functionalities required in the software.

** She is also professor and PhD student in Department of Computer Science and Statistics at University of São Paulo/USP.

Accordingly, these characteristics of OSS process have pointed out it as an agile process, together XP (eXtreme Programming) [1], Scrum [7], and others. Agile processes are generally lightweight and faster to develop software systems; thus, researchers have seen them like improvements in the way software is developed.

Recently, a significant amount of works related to successful OSS has been published, mainly those that have functionality similar to other softwares, e.g., web servers, browsers and text editors. Because of these successful OSS, interesting points can be investigated. For example, how OS development process works if the developers do not have enough knowledge about the software that they will develop. Another point is how OS process works if there is a limited number of developers or when the developers are "random", i.e., when they are employers of traditional commercial organizations or when they do not have characteristics of OS developers.

In this context, recent works have been investigating the OS development process, such as [5][10][13]. Besides, some researchers, e.g., [4][9], have motivated our work. In the same vein of Scacchi's work [9] that describes different ways to communicate software requirements in OS projects, the objective of this work is to investigate the role of a detailed document in text format describing requirements — the requirements specification — in the OSS development process. Thus, we have performed two studies that aim at investigating the followed questions: (i) How much are the analysis and design software models (as those build in "traditional" processes) relevant in the software development? (ii) How does the OS process work if developers do not have enough knowledge of requirements or if developers are not end-users? (iii) How does the OS process work if software requirements are not similar to other known systems? (iv) Is a requirements specification sufficient or important in the OS process? (v) Does OS process work with "random" developers? Are special characteristics required in developers?

The term "OS process" is usually used for referring to a set of processes with similar characteristics; development processes of the Mozilla and Apache are two good examples [4]; this is also valid to "traditional" processes. Thus, this paper aims at reporting our experience in simulating a process of these sets in a controlled environment, as well as describing knowledge acquired about the behavior of these processes.

According to [12], experimentation has commonly been applied with the purpose of evaluating methods, techniques, and tools in SE (Software Engineering). In this way, this work also aims at investigating the viability of using experimentation to evaluate software processes, specially the open source process. These studies were formulated and documented according to experimentation process proposed by [12] aiming to repeat these studies in next opportunities and achieve results with statistical significance.

This paper is organized as follows: Section 2 presents an overview of Experimentation in SE. Sections 3 and 4 describe the two studies (experiments) conducted. Section 5 provides learned lessons from the conducted experiments, and Section 6 gives conclusions remarks.

2 An Overview of Experimentation in SE

An empirical study is an act or operation for the purpose of discovering something unknown or of testing a hypothesis, involving an investigator gathering data and performing analysis to determine what the data mean. According to [12], there are three major different types of investigations that may be carried out: survey, case study, and experiment. A survey is often an investigation performed in retrospect. Case studies are used for monitoring projects, activities or assignments. The level of control is lower in a case study than in an experiment. A case study is an observational study while the experiment is a controlled study. In the experiment, the researcher has control over some of the conditions where the experiment takes place. It is normally conducted in a laboratory environment, which provides a high level of control. Thus, an experiment is a formal, rigorous and controlled investigation.

The experiment process aims at manipulating one or more variables and fix all the others. The effect of the manipulation is measured, and based on this a statistical analysis can be performed. There are two kinds of variables in an experimentation: independent and dependent variables. All variables in a process that are manipulated and controlled are called **independent variables**. Those variables that we want to study to see the effect of the changes in the independent variables are called **dependent variables**. Thus, an experiment studies the effect of changing one or more independent variables; these variables are called **factors**. The other independent variables are controlled at a fixed level during the experiment, or else we cannot say if the factor or another variable causes the effect. A **treatment** is one particular value of a factor. The treatment is applied to the combination of **objects** (elements to be manipulated in experiment, for example, programs or document that shall be reviewed) and **subjects** (people that participate of the experiment). Generally, the experiment process involves several activities [12]: (i) Definition: it determines the foundation for the experiment or why the experiment is conducted; (ii) Planning: the design of the experiment is determined, the instrumentation is considered and the threats to the experiment are evaluated; (iii) Operation: the treatments are applied to the subjects and data that will be analyzed are collected; (iv) Analysis and Interpretation: after collecting experimental data in the operation step, this one draws conclusions based on this data; (v) Presentation and Packaging: This includes primarily documentation of the results, which can be made either through the research papers for publication, lab packages for replication purposes or as part of a company's experience base.

The experiment process is iterative; it is not assumed that an activity is necessarily finished prior to that the next activity is started. In other words, it may be necessary to go back and refine a previous activity before continuing with the experiment. The exception is when the operation of the experiment has started, then it is not possible to go back to the definition and planning of the experiment [12]. According to [11], an important requirement for any experiment is repeatability, i.e. an experiment should be repeated several times to obtain statistical significance in the results.

3 Experiment I

This experiment compares the "traditional" process and the open source process to observe if analysis and design models are relevant to the software development process. In the next subsections, we will describe this experiment and document it according to steps proposed by [12].

3.1 Definition

Object of Study: The objects studied are the OS process and the "traditional" process to develop software.

Purpose: The purpose of the experiment is to observe if requirements specification and software models influence in the software development processes, i.e. if they are relevant.

Quality Focus: The quality focus is the productivity improvement in the software development. In the context of this paper, productivity is related to the quantity of functionalities of the software coded by developers.

Perspective: The perspective is from researchers, i.e. the researchers would like to know if there is a significant difference in the productivity using OS process or "traditional" process.

Context: The experiment was performed during the second semester of 2002 in the Department of Computer Science and Statistics, at University of São Paulo (USP/SCE). It involved 35 students divided in two teams.

3.2 Planning

Context Selection: The experiment was independently conducted by two teams: one based on OS process and other based on "traditional" process. Each team had a document to guide its activities. All subjects were students, then most of them had not worked in the software industry before. The experiment occurred outside of classroom and subjects annotated all tasks, respective dates and times consumed in the tasks. The problem used in this experiment was a real system, but simple, called Sapes. Then, we believe it would be a good system to our experiments.

Sapes is a Web application that manages references (bibliography) and its requirements specification is based on IEEE Standard[1]. This specification has been used for some years in other SE courses. SE researchers still verified this specification before initiating this experiment. This task had considerable relevance to identify contradiction and omission, as well as ambiguity in requirements. This specification totalizes 10 printed pages and 24 requirements. It is divided in: (i) preparation and management of references; (ii) general search and reports; and (iii) use of references in the preparation of scientific text.

[1] IEEE Std 830. IEEE recommended practice for software requirements specifications, June 1998.

Hypothesis: Analysis and design models are not necessary in the software development process.

Variables Selection: (i) **Independent Variables**: The independent variable is the process applied to develop software. Team 1 applied a "traditional" process; this team used an analysis and design object-oriented method to model the software; after that, the software was coded based on developed models. Team 2 applied the practices behind OS process to develop the software. (ii) **Dependent Variables**: The dependent variable is the time and effort required to build the software using OS process and "traditional" process. Another dependent variable is the relevance of the requirements specification and software models in the software development processes.

Selection of Subjects: The sampling technique to selection of subjects was the convenience sampling. The subjects were Computer Science students, during a SE course. From Background Experience Questionnaire that contained questions related to background experience of the subjects, we identified their experience level in software development (Table 1). This questionnaire asked about experience with requirements, analysis and design, object-orientation, UML [8], coding and knowledge of programming language.

It is important to observe that researchers writing papers would be users of the Sapes; on the other hand, undergraduate students that are not still used to write papers were developers (subject of the experiment). Therefore, requirements of this application were unknown or unclear for developers.

Table 1. Experience Level of the Subjects in Experiment I

Experience Level	Percentage
Never developed software	16.6
Software development on own.	22.2
Software development as part of a team, as part of a course.	55.6
Software development as part of a team, in industry.	5.6

Experiment Design: This experiment was designed applying general design principles: randomization and balancing. Subjects were randomly assigned to two teams; thus two similar teams in regard to experience were used for conducting the experiment. The experiment used an almost balanced design, which means that there is a similar number of subjects in each team.

In this experiment, the factor is the software development process, and the treatments are the "traditional" process and the OS process. Table 2 presents the quantity of subjects and the treatment applied to each team. As showed in Table 3, Team 1 was still divided in three groups that had a great functionality of the Sapes to model and to code.

Instrumentation: The materials supplied to subjects were: Task to Do Document (that describes the subjects behavior and tasks to do depending on which team they would participate) and Tasks and Times Form (where subjects would annotate tasks and time to every task). Furthermore, in the

Table 2. Assigning Team to the Treatments in Experiment I

Team	Quantity of Subjects	Traditional Process	OS Process
1	20	X	
2	15		X

Table 3. Division of Team 1

Group	Quantity of Subjects	Functionality
1	6	Preparation and management of references
2	7	General search and reports
3	7	Use of references in the scientific text preparation

beginning of the experiment, we provided the requirements specification of the Sapes and its E-R model for both teams. We believe that providing E-R model would not influence in our observations, since we have been interested in observing the modelling and coding of the Sapes' functionalities, and not specifically the database.

Validity Evaluation: (i) **Conclusion validity**: The experiment was elaborated in the way that results prove or do not prove the hypothesis. A threat to the conclusion validity is the quality of the data collected during the experiment. There is the risk that the data will be faked or incorrect. (ii) **Construct validity**: This experiment was conducted as part of a course and subjects were the students. Then, a threat regarding to the construct validity is that students may bias their data. Thus, in the beginning of the experiment, the students have been informed that results or data registered would not influence in their grade. (iii) **External validity**: It is probable that similar results could be obtained when running the experiment with other students as subjects. However, the results of this experiment cannot be directly generalized to the population, since most of the subjects were not representative of the software industry.

3.3 Operation

Preparation: Since OS developers and developers of the software industry have frequently good knowledge of the technologies applied to develop software, all subjects had an intense training in PHP[2], MySQL[3], and object-oriented concepts. The subjects would use these open source technologies to model and to code the software in this experiment. We believe that this training is very relevant and a way to prepare students to use these technologies in the future. Moreover, students had a constant support in PHP and MySQL during the experiment execution.

Members of the Team 1 had a training in an object-oriented software development method called Prodes [2]. This method employs the UML notation

[2] http://www.php.net
[3] http://www.mysql.com

[8] and presents diagrams and guidelines for the phase of the software life cycle. It combines concepts used in some existing object-oriented methods, e.g., Fusion [3] and RUP (Rational Unified Process)[4]. Activities and techniques of the Prodes are presented in Table 4.

Table 4. Activities and Techniques of the Prodes

Activity	**Techniques**
Requirement Engineering	Use Case Diagram Use Case Specification Domain Class Diagram
Analysis	Scenario Analysis Class Diagram Operation Model Life Cycle Model
Design	Collaboration Diagram Visibility Diagram Detailed Analysis Class Diagram Class Description Class State Diagram

In order to initiate the experiment execution, the Tasks to Do Document was distributed to subjects of both teams. For Team 1, this document contained guidelines about tasks to be done like in a traditional commercial organization producing software and using Prodes. For Team 2, this document described the behavior of OS developers; moreover, discussions about OSS, its characteristics and its development process were conducted in classroom.

Execution: Subjects of both teams played their tasks as "real" developers. During the experiment execution, they reported their tasks and the times dedicated to each task (in Tasks and Times Form).

All groups of the Team 1 worked simultaneously; each group had a coordinator that attributed tasks and annotated tasks developed and times. Team 2 had a coordinator (an unique person) that had a good knowledge about the software to be implemented. Then, he centralized the process, answered electronic mails, motivated the participation of the developers, selected the best new source codes, and managed information in the Web site. However, each subject of the Team 2 registered the tasks and times (in Tasks and Times Form).

Since Internet-based services support widely OSS projects, Team 2 used two main services used in most of the OSS projects: Web site and electronic mailing list. Web site was used for centralizing information and versions, and communicating about last versions. Communications among members were realized using only electronic mailing list. This list was mainly used for reporting bugs and suggestions, and communicating about new source

[4] http://www-136.ibm.com/developerworks/rational/products/rup/

codes. Although we did not used a control version system in this experiment; we controlled new versions and availed it through the Web site. Each new version corresponded to skeleton of a class, a method, a page, and others.
Members of the Team 1 executed several activities to develop the Sapes. To begin with, they studied the requirements specification. Secondly, they built software analysis models using techniques of the Prodes. Next, they validated the analysis models and built software design models using also techniques of the Prodes. Finally, they validated the design models and coded the software. On the other hand, members of the Team 2 studied the requirements specification and next, they coded the software using practices of the OS process: cooperative distributed work, communication using mailing list, use of Web site to centralize the source code and freedom to choose tasks.
Both teams had approximately five weeks to the experiment. Subjects dedicated to this project while they were involved in tasks of other courses.

Data Validation: In the end of the experiment execution, we applied the Feedback Questionnaire; questions referred to mainly suggestions, difficulties, time and relevance of the requirements specification. In this way, this questionnaire contained seven questions to Team 1 and 14 to Team 2. To avoid erroneous data, we guaranteed that answers to this questionnaire would not influence in the grade of the subjects. The Feedback Questionnaire contained questions whose answers were not multiple choice, i.e. we did not force subjects to choose answers into a set of predefined answers. Thus, they were free to register their opinion. We also conducted a final interview with all subjects. During the interview, the material elaborated — software models and software implementation — was presented and discussed.

3.4 Analysis and Interpretation

Results collected in Feedback Questionnaires pointed out that all students regarded this experiment very positive. Team 1 acquired a good experience in developing software using an object-oriented method; Team 2 experimented a new and interesting way to develop software.

All groups of the Team 1 pointed out that requirements specification was very relevant and sufficient to develop the software models. As a result, Team 1 produced a documentation with approximately 70 printed pages containing analysis and design models. Table 5 presents the required times for each group of the Team 1 to model and to code the software, as well as the quantity of prepared documentation. They pointed out that requirements specification and system models are also relevant for new developers that have not followed the project from beginning. The members said that discussions during the model development can minimize errors and help to understand the software requirements. Thus, software models of the Sapes have been a faithful representation of requirements. After finishing the modelling, software coding became a mechanical activity to translate models in source code. Besides this fact, learning Prodes was the main difficulty of all groups because they had had little experience in

object-oriented methods before. Furthermore, learning the Web technology to code the software was other difficulty.

Table 5. Time Required to Model and to Code

Group	Analysis and Design (hr)	Coding (hr)	Documents (pages)
1	42	39	46
2	19	16	14
3	16	32	13
Total	**77**	**87**	**73**

Related to Team 2, we observed that many members of this team had a good participation in the project; on the other hand, some members did not have a so good participation (Table 6). This fact has showed the self organization of the developers in real OS projects. Like in real OS projects, developers dedicated to this project while they were involved in other tasks.

Table 6. Participation Level

Level	Percentage
Good (more than 10 hours)	60.0
Medium (5 to 10 hours)	26.7
Low (less than 5 hours)	13.3

Despite the preview discussion about requirements with the subjects, the software implemented by Team 2 is not exactly a faithful translation of requirements. Some original requirements were changed and some new requirements have been implemented, but in agreement with the original requirements. This fact shows a very interesting characteristic: the tendency of changes in the requirements. Nevertheless, most of the members have confirmed that requirements specification had a very important role to guide their tasks, because they did not know the software requirements; 79% of members pointed out that requirements specification was sufficient to implement the software. Members of Team 2 pointed out several advantages of process applied, such as the rapid implementation (due to parallel tasks realized by developers) and the possibility to work anywhere in their free time. A lot of members emphasized the freedom of choosing tasks as the most relevant advantage. Regarding to difficulties, although training had been given to developers, the main difficulty of almost all members was to use the tools to code the software.

Concluding, the requirements specification is an important mechanism to communicate the software functionalities in software processes, specially when the developers do not have good knowledge about the software to be build. Even in OS process that usually does not use formal specification, when developers are not end-users and consequently, do not have knowledge of the functionalities, the specification has an important role. How this experiment was part of

a course and it had a date to end, both teams finished this experiment in the similar time. However, we can observe from Feedback Questionnaire and final interview that members of the OS process had less work to complete the experiment than members of the "traditional" process, i.e., in general, the effort to do the same work is lesser in the OS process. So, a detailed software models cannot be necessary; the exception is when software documentation is required.

4 Experiment II

Results achieved in the first experiment motivated us to continue our study. Subsequently that, we conducted Experiment II that aims at investigating the relevance of a requirements specification in OS processes. Next subsections describe this study and document it according to [12].

4.1 Definition

Object of Study: The objects studied are OS processes.
Purpose: The purpose is to investigate if a requirements specification is relevant in the OS processes.
Quality Focus: The quality focus is the productivity improvement of OS processes using developers that do not have special characteristics of OSS developers, mainly knowledge of the software requirements.
Perspective: The perspective is from researchers, i.e. they are interested in knowing if there is a significant difference in an OS process using a requirements specification and other OS process without a requirements specification.
Context: The experiment was performed at USP/SCE and involved 35 students divided in two teams. It was realized during the first semester of 2003 and involved students that had not participated in the experiment before.

4.2 Planning

Context Selection: Both teams performed independently the experiment. Each team had a document to guide its activities. The experiment occurred outside of classroom and subjects annotated all tasks, respective dates and times consumed in the tasks. The problem (object) used in this experiment, a Web application to references management, was the same of the Experiment I.
Hypothesis: Developing a software from a requirements specification using practices of the OS process is easier than using a OS process without a requirements specification, when developers do not have special characteristics of the OS developers.
Variables Selection: (i) **Independent Variables:** The independent variable is the process applied to develop software. This experiment observes the behavior of two OS processes: one developing software from a requirements specification and other, from a brief description of the software. (ii) **Dependent Variables:** The dependent variable is the difficulty to develop software using OS process.

Selection of Subjects: It was used convenience sampling as sampling technique to selection of the subjects, since they were students of a SE course. From Background Experience Questionnaire, we collected information from subjects about their experience in software development. Subjects of the Experiment I (Table 1) and Experiment II (Table 7) had similar experience level; most subjects did not have a good experience in software development.

Table 7. Experience Level of the Subjects in Experiment II

Experience Level	Percentage
Never developed software	13.6
Software development on own.	31.8
Software development as part of a team, as part of a course.	45.5
Software development as part of a team, in industry.	9.1

Experiment Design: Randomization and balancing were used as general design principles. Subjects were divided in two teams and the assignment to each team was random. In this experiment, the factor is the software process and the treatments are the OS process using a requirements specification and the OS process without the utilization of a requirements specification. The treatments applied and the quantity of subjects of each team are shown in Table 8.

Table 8. Assigning Team to the Treatments in Experiment II

Team	Quantity of Subjects	With Requirements Specification	Without Requirements Specification
1	18	X	
2	17		X

Instrumentation: The materials supplied to subjects were: a Task to Do Document and a Tasks and Times Form. For both teams, we provided a overview of the Sapes — description of the software in one printed page — and its E-R model. Furthermore, we provided also the requirements specification of the Sapes in 10 printed pages only to Team 1. This specification was the same used in the Experiment I. We instructed Team 1 to not make this specification available to members of the other team; Team 2's members would have gradually the requirements through the electronic mail.

Validity Evaluation: (i) **Conclusion validity**: The experiment was also elaborated in the way that results prove or not prove the hypothesis. Conclusion validity has as threat the quality of the data collected, then we asked subjects to be as precise as possible. (ii) **Construct validity**: How the experiment was conducted as part of a SE course, the students were subjects. The main threat to this validity is the students manipulate the data. (iii) **External validity**: The results reached in this experiment cannot be directly generalized to the population, because most of the subjects in this

experiment were students that did not probably have good experience in the software industry.

4.3 Operation

Preparation: All subjects had a training in PHP, MySQL and CVS (Concurrent Version System)[5]. Like in the Experiment I, these courses were conducted in the weekends for approximately two or three weeks each one. Besides, support in these technologies was provided to the subjects during the experiment execution. Task to Do Document was distributed to all subjects; furthermore, a discussion related to OS process and its characteristics was conducted with all subjects in classroom.

Execution: Each team worked separately, and each subjects executed independently their tasks. We asked them that all communication was achieved through the electronic mail. It was a way to register the progress of the experiment and the relationship among members.

Both teams used repositories of OS code and applications available on the Internet: Team 1 used the CodigoLivre[6] and Team 2, the SourceForge[7]. These repositories provide free hosting to OS software development projects, and provide free services for OS developers, e.g., electronic mail, mailing lists, discussion forums, project web server, and version control system. Certainly, the use of these repositories become the experiment more realist.

Like in a real OS project, each team had a coordinator. In this experiment, the coordinator had a good knowledge of the Sapes. The role of the coordinator were to manage the project repository, to read and answer electronic mail, and to manage tasks. The coordinator and one or two developers selected the best source codes implemented by developers to add in the stable source code. The main tasks of the developers were to read and to understand the requirements specification, to write code, to read code of other subjects, and to test the software. Beyond these tasks, they studied the technologies used in the coding (PHP and MySQL) and the tools to management of the project (CVS and repository).

During the experiment execution, we observed the behavior of both teams. We reviewed implemented source codes and participated in mailing list.

Data Validation: We applied the Feedback Questionnaire to all subjects. Both teams answered 14 questions related to mainly difficulties, advantages, and suggestions about the process used, dedication in the project, and relevance of the requirements specification. This questionnaire did not have multiple choice questions, so as the questionnaire in Experiment I.

It is important to observe that during the experimentation execution, we also annotated the behavior and opinion of the subjects through interviews.

[5] http://www.cvshome.org
[6] http://codigolivre.org.br
[7] http://sourceforge.net

4.4 Analysis and Interpretation

Most of subjects in both teams pointed out this experiment as a good experience in software development. The differences between the processes experimented and other "traditional" processes in SE courses showed them that there is alternative ways to develop software.

From Feedback Questionnaire applied to subjects of the Team 1 and Team 2, we identified the main difficulties, advantages and suggestions about the processes. In this way, the main difficulties pointed out by subjects were:

- Even though training in the technologies to code the software — PHP and MySQL — had been provided to subjects, they pointed out the use of these technologies as a difficulty. So much time was required to learn and use them in practice. Certainly, if the subjects had good knowledge in these technologies, they had not taken time in this activity and concentrated in the activities directly related on software development; and
- The use of services provided in OS projects repositories, mainly the version control system, was another difficulty, although a training in this system had been also provided before experiment execution.

Subjects of both teams identified advantages in the processes:

- The decentralized process did not make pressure over the developers. They could work anywhere and anytime. Besides they had freedom to choose the tasks they want to do.
- Although face-to-face discussions could be more effective than using mailing list, the last one was an important way of communication among developers. They read constantly electronic mail, so as many "real" developers used to do; then, they could follow the project course, send and answer questions, discuss about source code and understand the requirements and the team organization;
- The use of a version control system had an important role to concentrate the source code, since often there were more than one subject developing the same Sapes' functionality at the same time. Besides, this version control system was provided by repository of OS software over the Web. Thus, it was possible to access the source code from any computer connected to Internet; and
- Exchange of experience was other advantage of the processes applied, confirming that learning is one of the motivational forces for developers as proposed by [13]. Reading source code developed by others, subjects could learn about how to use the technologies to code the software beyond tutorials and manuals. Another point very interesting in OS processes is that each developer shared spontaneously his knowledge.

Both teams noticed some suggestions about the process applied:

- Although we provided the E-R model, subjects of both teams suggested availing some high level model of the software; for example, using use case

diagram (identify the software interface) or class diagram (identify class and relationship). These models present a software overview and it can help the modularization and the developers organization. These models can also contribute to minimize the time of software development, since discussion about modularization can be minimized or avoided;

– The OS process works only if the developers are interested and motivated. So, when developers do not understand the requirements or when the objective of the software is not clear, it is difficulty for developers to have motivation and enthusiasm. Mechanisms, such as requirements documents, software models, interaction through the electronic mail, and dynamism in the mailing list, can be adopted to motivate the developers; and
– The coordinator, one person or a group of person, is extremely important to the OS processes work adequately. Her main tasks are to keep the dynamism in mailing lists, to bring up to date the information in Web site, to collect and to select the best codes to be inserted in the stable version. Moreover, he also must participate as developer in the project. Even having the requirements specification, the coordinator is still essential to developers organization.

So as Team 2 in Experiment I, the organization of the subjects in both teams was very similar to real OS projects. There were subjects that had good participation and others with low dedication (Table 9). We can observe that the participation levels in both teams are very similar.

Table 9. Participation Level of Team 1 and Team 2

Level	**Team 1 (%)**	**Team 2 (%)**
Good (more than 10 hours)	55.6	58.8
Medium (5 to 10 hours)	27.7	29.4
Low (less than 5 hours)	16.7	11.8

Subjects of the Team 2 pointed out that a brief description of the software to be developed is not sufficient to understand the software requirements, even discussing requirements in mailing list.We can observe that, even having a detailed document describing the software requirements, it was necessary for Team 1 to discuss the understanding of the requirements with other subjects through mailing list during the experiment execution. So, only the use of a requirements specification to communicate the requirements in OS process is not sufficient. Concluding, it is possible to develop a software using OS process without a requirements specification, but we recommend the development of a detailed requirements specification and discussions about the requirements through the mailing list, when developers are not end-users of the software to be developed.

5 Lessons Learned

To obtain valid results using experimentation, the experiment should be repeated several times to achieve statistical significance. From our first study (Experiments I and II), we reported learned lessons that can contribute to the conduction of experiments involving software processes, and the use of OS processes with "random" developers:

- We can stand out that simulation of software processes is not a trivial activity, mainly simulation of OS processes. Thus, a good way to investigate the practices related to software processes is to formalize the study using experimentation process;
- It is possible to use OS process with "random" developers to build software; however, some observations must be considered: (i) A requirements specification is relevant mechanism to provide a software overview. We believe that this specification is also important for new developers that want to collaborate with the project; (ii) According to developers of these experiments, documents in text format — a requirements specification, in this case — are more difficult than graphical representations to interpret. Thus, it is interesting to establish another way to communicate the requirements in addition to text documents. Analysis and design models can be inserted to a better communication of the requirements in OS processes. Known techniques can be used for building these models, like that proposed by UML; and (iii) The domain of the technologies used for coding and managing the project is extremely important to OS processes work effectively. In view of this, if OS process will be applied with "random" developers, it is necessary a intense training in these technologies; and
- To OS process works effectively, in real OS process or in OS process with "random" developers, the coordinator has a very important role. He must conduct his activities aiming to centralize the process, to create capability of cooperative distributed work, and the more important, to motivate the developers.

6 Conclusions

For SE students, introduction of the OS process in their curriculum was a valuable experience. This experience helped students to understand better agile processes, since OS process presents many similarities with that, mainly regarding to people coordination and tasks distribution. Several characteristics and advantages of OS processes discussed by OS researchers were confirmed with these pilot studies.

The investigation of software processes is not a trivial activity. So, we believe that studies like this can contribute as the first steps to understand and to formalize software process, specially OS process. Our experiments showed also that the requirements elicitation — an important activity of the Requirement

Engineering to understand the problem to be automated — and consequently, the requirements specification is a important vehicle to communicate the software functionalities. So even in this recent software process (OS process, in this case), the problem related to requirements elicitation persists and continues to be extremely relevant.

For future works, these experiments must be performed several times to prove the results achieved in these initial experiments. Furthermore, conducting these experiments in industrial environment is certainly a good way to become the experiment more realist. Moreover, we aim at collecting more precise and objective measures from feedback questionnaires using a multiple choice questionnaire, and consequently, becoming easier the data collection and the calculus of results with statistical significance.

Acknowledgments: Our thanks go to all students of SCE-186 and SCE-531 that collected data during these experiments and to Thiago Bianchi that participated as coordinator in Experiment II. Sincere thanks go to Prof. Dr. José Carlos Maldonado for valuable suggestions.

References

1. K. Beck. *Extreme Programming Explained.* Addison-Wesley Publishing Company, 2000.
2. T. E. Colanzi. A UML-based integrated approach of software development and testing. Master's thesis, ICMC/USP, São Carlos, SP, Brazil, June 1999.
3. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilcheist, F. Hayes, and P. Jeremes. *Object-Oriented Development: The Fusion Method.* Prentice-Hall, 1994.
4. A. Mockus, R. T. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), July 2002.
5. J.W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4):246–256, April 2004.
6. E. S. Raymond. *The Cathedral and the Bazaar.* O'Reilly & Associates, February 2001.
7. L. Rising and N. S. Janoff. The scrum software development process for small teams. *IEEE Software*, 17(4):26–32, July/August 2000.
8. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley Publishing Company, Massachusetts, 1999.
9. W. Scacchi. Understanding the requirements for developing open source software systems. In *IEE Proceedings – Software Engineering*, volume 149, February 2002.
10. D. Spinellis and C. Szyperski. How is open source affecting software development? *IEEE Software*, 21(1):28–33, January 2004.
11. W. F. Tichy. Should computer scientists experiment more? *IEEE Computer*, 31(5):32–40, May 1998.
12. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering.* Kluwer Academic Publishers, 2000.
13. Y. Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *Proc. of the 25th International Conference on Software Engineering*, Portland, Oregon, May 2003.