Experimentation is a very important discipline for several research areas, which includes software engineering. Empirical studies are necessary to evaluate how effective or how promising processes, methodologies and techniques are in contrast to other approaches. Experimentation also allows determining the value of the claims about the subject of the study, supporting decision makers to choose which method, process or technique to use in a software development project.

Unfortunately, several software engineers do not give much attention to experimentation [104], which might hinder how reliable or promising is the methodology or the technique. In order to better understand the progressive approach impact on the implementation activities, we performed an experimental study using the implementation method and comparing the progressive approach versus the regular approach. In fact, this study does not cover the whole implementation method, since the concurrency control concern is not considered. Therefore, the study focuses on data management and distribution concerns.

There are several principles of experimentation and empirical studies that must be followed [5, 104, 69, 40], otherwise any conclusions derived of the so-called experiment might not be useful. Therefore, another contribution of this chapter is to provide a framework that allows performing other studies under the same conditions to confirm some results we derived, and others we could not achieve.

This chapter describe the study plan specifying the study goal, the hypotheses investigated, the treatment applied and analyzed by the study, the control object, which is the object to be compared with the application of the treatment, deriving the analyzes, the experimental object, the object to which the treatment is applied, and the subjects, which are the participants of the study. Moreover, the independent variables, which are variables that do not change their values during the study, and the dependent variables, which are variables to be affected by the study, and therefore, variables to be measured in order to evaluate the study are also defined. Finally, the design, preparation, treats of validity, execution, and analysis of the study are presented, following a format based on [100].

The next sections describe the designed study performed from January to March 2004 in order to characterize the impact of using a progressive approach (see Chapter 3) when executing the implementation activities of the implementation method (see Chapter 4). It is important to make clear that this experimental study is concerned to characterize the use of the progressive approach versus not using it, and does not try to make any discussion between using aspect-oriented programming or object-oriented programming.

5.1 Goal definition

The following sections describe the goal of this experimental study.

5.1.1 Global goal

Characterize the progressive implementation approach in the context of aspect-oriented software implemented using AspectJ.

5.1.2 Measurement goal

Considering the progressive implementation approach, we plan to characterize its difference from a regular approach, which we call the non-progressive approach, with respect to:

- Implementation time: time to implement selected use cases;
- Requirement changes time (during development): time to perform changes in selected use cases, after the customer or the software architect request the change;
- Test execution time: time to run test-cases of selected use cases (acceptance tests);
- Pre-validation prototype time: time to yield a first executable prototype of selected use cases, before validation;
- Post-validation prototype time: time to yield an executable prototype and validate it with the customer. This implies that requirements are validated and some of them might have changed. A mentor played the customer role.

5.1.3 Study goal

Analyze the progressive implementation approach for aspect-oriented software and AspectJ.

With the purpose of characterize the impact of adopting a progressive implementation approach.

With respect to the time to implement, change, testing, and yield pre and post-validation prototypes.

From the point of view of the software developer.

In the context of students of a software engineering graduate course, with industry expertise, implementing selected use cases of a real application.

5.1.4 Questions

Is the time to implement, change, testing, and yield pre and post-validation prototypes in a progressive way different from not using a progressive approach in the context of aspect-oriented software implementation?

5.1.5 Metrics

Data on how long it takes to implement, change, testing, and yield pre and post-validation prototypes of the selected use cases. The data unit is time in minutes.

5.2 Planning

This section describes the study plan showing how the study is designed. This allows running other studies using the same plan, which can confirm some of our results and derive new ones we could not derive. Actually, future studies can change some variables in order to measure their impact.

5.2.1 Hypotheses definition

Before presenting the hypotheses, it is necessary to introduce some symbols used through the rest of this chapter to denote the metrics to be collected and analyzed.

IE — Implementation time;
TE — Testing time;
CE — Change time;
PE — Time to yield a pre-validation functional prototype;
VE — Time to yield a post-validation functional prototype.

Each of these metrics has two variations, progressive and non-progressive. For example, there is implementation time using the progressive approach (IE_P) and the same metric using the non-progressive approach (IE_{NP}). The following hypotheses definition use these symbols.

The main hypothesis is the null hypothesis that states there is no difference between using or not the progressive approach. Therefore, the study tries to reject this hypothesis. There are five null hypotheses, one for each metric the study analyzes. The following is a composition of these five null hypotheses.

Null hypotheses ($H0_{1..5}$): The time to implement (1), testing (2), change (3), and yield pre (4) and post-validation (5) prototypes using a progressive approach for aspect-oriented development is not different from using a non-progressive approach.

$$H0_1 : IE_P \cong IE_{NP}$$
$$H0_2 : TE_P \cong TE_{NP}$$
$$H0_3 : CE_P \cong CE_{NP}$$
$$H0_4 : PE_P \cong PE_{NP}$$
$$H0_5 : VE_P \cong VE_{NP}$$

Additionally, alternative hypotheses are defined to be accepted when the corresponding null hypothesis is rejected. In fact, there are two different alternative hypotheses set.

Alternative hypothesis ($H1_{1..5}$): The time to implement (1), testing (2), change (3), and yield pre (4) and post-validation (5) prototypes using a progressive approach for aspect-oriented development is different from using a non-progressive approach.

$$H1_1 : IE_P \neq IE_{NP}$$
$$H1_2 : TE_P \neq TE_{NP}$$
$$H1_3 : CE_P \neq CE_{NP}$$
$$H1_4 : PE_P \neq PE_{NP}$$
$$H1_5 : VE_P \neq VE_{NP}$$

Alternative hypothesis ($H2_{1..5}$): The time to implement (1), testing (2), change (3), and yield pre (4) and post-validation (5) prototypes using a progressive approach for aspect-oriented development is smaller than using a non-progressive approach.

$$H2_1 : IE_P < IE_{NP}$$
$$H2_2 : TE_P < TE_{NP}$$
$$H2_3 : CE_P < CE_{NP}$$
$$H2_4 : PE_P < PE_{NP}$$
$$H2_5 : VE_P < VE_{NP}$$

5.2.2 Treatment

The treatment applied in this study is the progressive implementation approach, where persistence and distribution are not initially considered in the implementation activities, but are gradually introduced, preserving functional requirements. This is a one factor experimental study, since we have a single treatment. The absence of the treatment (the progressive approach) is called non-progressive approach, which is discussed in the following section.

5.2.3 Control object

In order to identify the impacts of using a progressive approach, the control object is the implementation of the experimental object without the progressive approach, which we call non-progressive approach. Therefore, in the non-progressive approach, persistence and distribution are implemented at the same time, in a single iteration, together with the functional requirements. In this way, the subjects are divided in two groups: the subjects of one group implement use cases using the progressive approach, whereas the subjects of the other do not use the progressive approach.

Since using the non-progressive approach means not using the progressive approach, we actually compare using to not using the progressive approach, and therefore, this is a one factor study.

5.2.4 Experimental object

The study used selected use cases of the Health Watcher software, a real software, that allows citizens to complain about diseases problems and retrieve information about the public health system, such as the location or the specialties of a health unit. The selected use cases cover several kinds of services, such as, recording, retrieving, and updating data. In addition to the use cases, we also consider requests for functional requirement changes, which are performed during the software implementation, and test-cases that guide the unit tests. We selected five of the nine use cases of the Health Watcher specification, and those five are representative since they cover the same kind of operation performed by the others, which are retrieving information, giving an employee different access grant, and cleaning the database. Moreover, the selected use cases are the most important use cases of the software, and therefore, they would be naturally selected as the first ones to be implemented. The use cases and their scenarios are

RF01 Retrieve Information — describes several reports the software should implement such as retrieve information about complaints (1), diseases (2), specialties of a health unit (3), and which health units have a specific specialty (4);

RF02 Record Complaint — describes how to register the different kinds of complaints: food complaint (5), animal complaint (6), and special complaint (7);

RF10 Login — describes how an employee should login into the system in order to perform restricted operations (8);

RF11 Record Data — describes how to insert and update employees (9), and how to update health units (10);

RF12 Update Complaint — describes how to update data about how a complaint should be handled by the public health system (11).

Note that in these five use cases we identified eleven use-case scenarios to be implemented. In this study, those scenarios are distributed in three iterations, as shown in Section 5.6.

5.2.5 Experimental subjects

The participants of this study are MSc and PhD students taking a graduate course on advanced topics on programming language. The course were specifically offered to perform the study, and therefore, the participants were aware that they were participating of an experimental study and that their data would be used in the study analysis. We expected that most of them had experience in industrial software development. In fact, they answered a questionnaire in order to characterize their academic and industry expertise before the study. More information about their expertise is presented in Section 5.6.

5.2.6 Independent variables

- Implementation of information systems;
- Requirements and design expressed using use cases and UML diagrams;
- Iterative and incremental implementation;
- Aspect-oriented programming with Java and AspectJ, using specific design patterns and frameworks [91, 85, 53];
- Implementation order of persistence and distribution concerns. Although the implementation method does not demand a specific implementation order, we fixed the order to implement persistence before distribution to avoid undesirable bias, which means we are comparing times of these non-functional requirements implemented in a same order. Additional studies should be performed to evaluate the impact of the concerns implementation order.
- Specific technologies to implement web information systems. These technologies are: Java [27], AspectJ [41], databases, JDBC [103], distribution systems, Java RMI [59], Java Servlets [32], and object-oriented analysis and design.

5.2.7 Dependent variables

The time to implement, change, testing, and yield pre and post-validation functional prototypes. We can provide views of these data per use case, per use-case scenario, or per iteration.

5.2.8 Trials design

The software should be implemented using a Use-Case Driven Development (UCDD) approach, which is adopted by the Rational Unified Process (RUP) [39] and several other modern processes. According to UCDD, a software is designed, implemented, and tested based on its use cases. Chapter 4 gives a brief explanation on use cases. To implement a use case or a use-case scenario, programmers should implement parts of the system that are necessary to realize that specific use case or scenario.

Without using the progressive approach, the functional and non-functional requirements associated to a use case or scenario are usually implemented in the same iteration (the non-progressive approach).

On the other hand, when planning the progressive approach, the implementation of some non-functional requirements (persistence and distribution) is schedule after implementing the functional part of the use cases and the user interface. Therefore, use cases are partially implemented in functional iterations, until a functional prototype is finished. At this moment, this prototype is validated by the system stakeholders and, if necessary, changes are made. After validating, the functional prototype evolves to a persistent and distributed system, in two different non-functional iterations.

To avoid undesirable bias we enforced the implementation of persistence before implementing distribution in both approaches. This assures that we are comparing times of these non-functional requirements implemented in a same order. Additional studies should be performed to evaluate the impact of the concerns implementation order.

Figure 5.1 shows the two different development dynamics. In the figure, times x and y have no relation with times x' and y'. Note that, as explained before, when implementing a use case using a progressive approach the functional requirements and nonpersistent (volatile) data management are implemented in a first iteration, called functional iteration. After validating this functional prototype, persistent data management and distribution are implemented in two special iterations (non-functional iterations). When implementing the next iteration, the concerns implemented in the non-functional iterations should be turned-off, in other to implement the new use cases and test the whole system using only nonpersistent data collections. By using this approach a functional prototype, without persistence and distribution, is delivered at the end of each functional iteration to be validated.

On the other hand, if the use case is being implemented without using a progressive approach, both non-functional requirements (persistence and distribution) are implemented in the same iteration of the functional requirements. When implementing the next iteration, non-functional requirements already implemented might affect the new requirements implementation and testing, since they are not turned-off like in the progressive approach.

It is important to mention that the resulting software in both approaches is the same, they have the same functionality and the same non-functional requirements implemented. The only difference between the implementation approaches is the order non-functional requirements are implemented and tested, and the need to implement nonpersistent data collections if using a progressive approach. Therefore, at the end of each iteration (x or x' and y or y') we should have the same software, independent of what implementation approach was used.

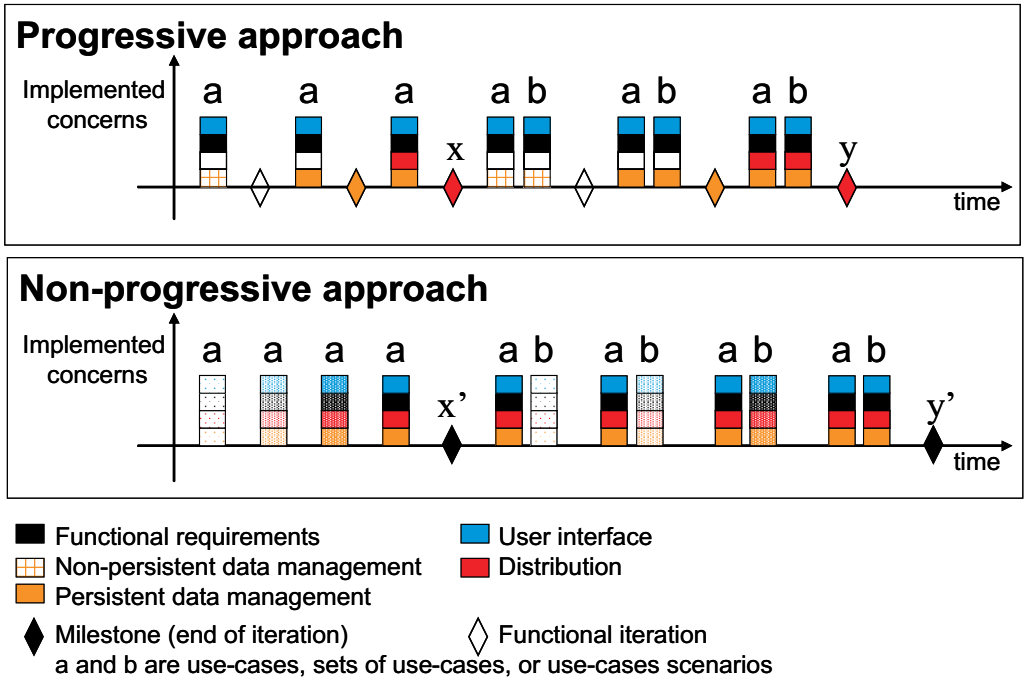

Figure 5.1: Iterations using progressive and non-progressive approaches.

5.3 Preparation

Before starting the study, the subjects answered the questionnaire (Appendix A) about their profiles and experience with software development. The subjects were aware that their data would be used by the experimental study. The subject's data is presented in Section 5.6.

5.4 Analysis

The study analysis compares the collected data from the experimental object trials and control object trials in order to see if the null hypotheses can be rejected.

The study analyzes the impact of the progressive approach on

- Implementation time;
- Requirement changes time (during development);
- Tests execution time;
- Pre-validation functional prototype time;
- Post-validation functional prototype time.

As we have a completely randomized one factor study [104, 40], we use a t-test [40, 104] to analyze the results. The t-test is a statistical test that compares the mean values of two sets of data, and analyzes if their differences are significant. In fact, we can perform two different analyses using the t-test.

5.4.1 Yes-No decision

The idea when analyzing the collected data is to try to reject the null hypotheses by showing that the expected mean values are not the same for a given significance. In this way, we perform the t-test with the samples of the progressive and non-progressive approach to implement the use cases. We consider this a one factor study since using non-progressive approach actually means implementing the use cases without using the progressive approach. The so-called non-progressive approach is our control object. In this way, this test is a yes-no decision, since we can only say if there is a difference between the sample means.

Consider $x_1, x_2, ..., x_n$ the samples (times) of applying the progressive approach, and $y_1, y_2, ..., y_m$ the samples of applying the non-progressive approach. We first define the mean values ($\bar{x}$ and $\bar{y}$) of the samples:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$\bar{y} = \frac{1}{m} \sum_{i=1}^{m} y_i$$

Now, following the t-test, we define the distribution t_0 in the following way:

$$t_0 = \frac{\bar{x} - \bar{y}}{s_p \sqrt{\frac{1}{n} + \frac{1}{m}}}, \text{ where } S_p = \sqrt{\frac{(n-1)S_x^2 + (m-1)S_y^2}{n+m-2}}$$

where S_x^2 and S_y^2 are the variances of the samples:

$$S_x^2 = \frac{(\sum_{i=1}^{n} x_i^2) - n\bar{x}^2}{n-1}$$

$$S_y^2 = \frac{(\sum_{i=1}^{m} y_i^2) - m\bar{y}^2}{m-1}$$

The null hypotheses expect the same mean for time values. Therefore, in order to reject the null hypothesis we want to find if the difference of the means is different from zero, which means they are different, comparable. The test consists in checking if $|t_0| > t_{\alpha,f}$, where $t_{\alpha,f}$ is the t distribution at a α percentage point, or significance level, using f degrees of freedom, where $f = n + m - 2$ [104]. The distribution is tabulated elsewhere [40, 104]. We perform the test at the 0.05 significance level, since we had few participants in the experiment. In other words, we use a confidence level of 95%, which means that we are working with the 95% probability of using values that are close to the mean values.

However, this test based in a yes-no decision might not be enough. There is a more effective way to analyze data than just saying if the samples mean are different or not.

5.4.2 Confidence interval

We can also analyze the collected data by determining the confidence interval [40] for the mean difference of progressive and non-progressive times ($\bar{x} - \bar{y}$). If the interval contains zero, which means that zero is a possible value for the difference, the samples are not significantly different. This is a more effective way to analyze data than just saying if the samples mean are different or not. A narrow confidence interval indicates a high degree of precision. On the other hand, a wide confidence interval indicates that the precision is not high.

We first used hypothesis testing to identify if the samples are significantly different. When the null hypothesis is false, the sample means are significantly different and it is not necessary any additional analysis. On the other hand, when the null hypothesis is true, instead of just saying that the sample means are not significantly different, we also determine the confidence interval to analyze the degree of precision of this analysis. The confidence interval is defined in the following way:

1. Compute the samples means $\bar{x}$ and $\bar{y}$, as defined in Section 5.4;
2. Compute the sample variance S_x^2 and S_y^2, also from Section 5.4;
3. Compute the mean difference: $\bar{x} - \bar{y}$;
4. Compute the standard deviation of the mean difference:

$$S = \sqrt{\frac{S_x^2}{n} + \frac{S_y^2}{m}}$$

5. Compute the effective number of degrees of freedom:

$$v = \frac{(S_x^2/n \ + \ S_y^2/m)^2}{\frac{1}{n+1}(S_x^2/n)^2 + \frac{1}{m+1}(S_y^2/m)^2} - 2$$

6. Compute the confidence interval for the mean difference:

$$(\bar{x} - \bar{y}) \mp t_{[1-\frac{\alpha}{2};v]} S$$

   where $t_{[1-\frac{\alpha}{2};v]}$ is the $(1 - \frac{\alpha}{2})$-quantile of a t-variate with v degrees of freedom;

7. If the confidence interval includes zero, the difference is not significant at $100(1 - \alpha)\%$ confidence level. Similar to the null hypothesis test, we used a significance level at 0.05 that correspond to a 95% confidence level, which means that we are working with a 95% probability that the population mean are in the interval.

This alternative analysis does not change the result of the previous hypothesis test. The idea is to provide additional data to decision-makers, in this case the degree of precision.

5.5 Threats to Validity

This section discusses how valid are the results and if we can generalize them to a broad population. There are four kinds of validity. Internal validity defines if the collected data in the study resulted from the dependent variables and not from an uncontrolled factor. Conclusion validity is related to the ability to reach a correct conclusion about the collected data, to the used statistical test, and how reliable are the measures and the collected data. Construct validity is concerned to assure that the treatment reflects the cause and the results reflect the effect, for example, without being affected by human factors. Finally, external validity is concerned with the ability to generalize the results to an industrial environment.

5.5.1 Internal Validity

The experimental subjects are MSc and PhD students of a graduate course. The students are from the Software Engineering area, so they have some experience in developing software. In fact, most of them have experience in the software industry, which contributes for being a representative set of software developers. However, despite most of the subject have experience in the software industry, their experience is no more than five years for most of them.

Despite being separated in two groups, one that uses progressive approach and the other that does not use, both subjects group used the same aspect-oriented technology, AspectJ. Therefore, we did not expect the subjects to be unhappy or discouraged in performing or not the treatment, since the resultant software is essentially the same for both situation. In addition, the study execution is necessary for the conclusion of the course, and therefore, we did not expect anybody to quit the study.

One confounding factor could be the subject's experience. In fact, the subjects filled a questionnaire about their experience and expertise in academia and industry. These data, presented in Section 5.6, are used in order to identify this possible confounding factor. Since there were few subjects to perform the study (6 in one group and 7 in the other), we randomly distributed the subjects to treatments (see Section 5.2) instead of defining blocks, which would decrease the number of samples to be compared.

5.5.2 Conclusion Validity

A mentor was always present during the study execution to guarantee the correct data collection and implementation of the treatment. The study uses a t-test to compare the data between applying the treatment (progressive implementation) and the control object (non-applying the treatment, also called non-progressive implementation). The t-test is more suitable to this study since it is concerned in comparing the mean values of the sample data of the two groups, instead of comparing a sample from one group with a sample from the other, with can provide undesirable bias. For example, data from two specific subjects cannot be compared to each other since they might have different expertise. On the other hand, when considering the mean value, we are comparing data from one group with the data from the other, and not specific subjects, which can decreases the subject's expertise impact.

5.5.3 Construct Validity

The subjects applied the treatment to selected use cases of a real information system using an execution plan, which explains how to apply the treatment. In addition, they performed a dry run to make clear how the treatments should be implemented and how data should be collected.

In fact, we performed a previous study, however, without the concern to provide exact guides to the subjects on how to implement the software, and how to collect the data. This actually resulted in incomparable data, since the subjects collected data in different way. This shows how useful are these threats of validity in order to guarantee the validity of the study results.

5.5.4 External Validity

One expected result of this study is to guide programmers on when to use the progressive approach. As we used randomization to separate the subjects in two groups, we expect to decrease the confounding factors, since the most important is the subjects' expertise. However, the limited number of subjects does not allow generalization outside the scope of the study. On the other hand, we expect that the results, including the subjects' feedback, can be used as guidelines to better implement aspect-oriented software.

The terminology, implementation process, and technology used in the study are currently used in the software industry, and therefore, are adequate to our objective. We did not have problems with temporary issues, since the study was performed in one of the university laboratories, specially reserved to the study execution.

Although the results are limited by the narrow scope we have, we believe that a considerable contribution is the study design. This framework can guide other studies in order to evaluate the progressive approach with more general and conclusive results and can also support other kind of studies, for example to identify the impact of other factors variation, evaluating alternative approaches to implement aspect-oriented software.

5.6 Execution

As previously mentioned, this study was performed during a graduate course. In the first half of the course we discussed several papers about aspect-oriented programming [42, 45, 46, 47, 43, 22, 54], AspectJ [41, 83, 30, 97], specific design patterns for the kind of system they implemented in the study [53, 2, 82, 86], and the use of these design patterns with AspectJ [91, 85, 48]. We provided exercises[1] about AspectJ, JDBC, and RMI, in order to give the subjects a minimum knowledge about these technologies, in particular to the ones that did not have enough contact with them. We also performed a dry run to give the subjects a chance to familiarize with data collection and plan execution.

The study was executed during the second half of the course, according to the schedule in Table 5.1. As previously mentioned in the experimental object definition (Section 5.2), there are 11 use-case scenarios to be implement.

[1]Those exercises can be found at `www.cin.ufpe.br/~scbs/talp1/nivelamento`.

Use case	Scenario to implement	Scenario id
Iteration 1 — 23/01/2004		
[RF02] Register Complaint	Food complaint	01
[RF01] Retrieve Information	Specialties of a health unit	02
Iteration 2 — 06/02/2004		
[RF02] Register Complaint	Animal complaint	03
[RF10] Login		04
[RF11] Register Data	Insert employee	05
[RF01] Retrieve Information	Complaint	06
Iteration 3 — 20/02/2004		
[RF12] Update Complaint		07
[RF01] Retrieve Information	Health units with a specialty	08
[RF01] Retrieve Information	Disease	09
[RF02] Register Complaint	Special	10
[RF11] Register Data	Update health unit	11
12/03/2004		

Table 5.1: Study schedule.

We provided several documents to the teams, including the selected Health Watcher's use-cases specification, class diagram, test-cases, and implementation plan stating the order in which the use-case scenarios should be implemented in each iteration (Table 5.1). We also provided the user interface code, which are HTML documents and Java Servlets, in order to speed up the study, since it should be performed during the second half of the course. Since the progressive approach demands implementing the user interface code at the same time the functional requirements are implemented, whether the progressive approach or the non-progressive approach is used, the user interface code is implemented at the same point. Therefore, this should not affect the study results. Moreover, since at the time of the study the tool support was not implemented, we also provided nonpersistent data collections (see Section 3.2) that would be automatically generated (see Chapter 6).

As previously mentioned, we also simulated requirement changes during development. The requirement changes are the following:

- In order to simulate a customer's request to change the requirement after using an implemented use-case scenario for the first time, we requested changes to add attributes to classes
  - After implementing the Food complaint scenario (01) of the use case Register Complaint ([RF02]);
  - After implementing the Animal complaint scenario (03) of the use case Register Complaint ([RF02]).
- In order to simulate a change in the system design after presenting the implemented use case for the first time, we requested a change to extract a class from three existing classes

– After implementing the Food complaint scenario (01) of the use case Register Complaint ([RF02]).

In fact, there are two change requests, since two of them are requested at the same time during Scenario 01 implementation. Those changes were chosen because they are pretty common changes, like adding new information or modifying the design model. It is important to mention that as the resulting software is the same independent of the used approach, we did not perform any maintenance change. In fact, the progressive approach tries to increase productivity by allowing the identification of requirement changes during the software development.

5.6.1 Questionnaire data

The first step before starting the study is to apply the questionnaire to the subjects in order to collect information about their experience. As previously mentioned, all the subjects were aware that the collected data would be used in an experimental study. In the following pages, Figure 5.2 depicts the academic expertise and Figure 5.3 depicts the industry expertise of the thirteen subjects according to the score in Table 5.2.

Score	**Expertise**	
1	Only in this course	(OITC)
2	Less than 6 months	(< 6m)
3	Between 6 months and 2 years	(6m-2y)
4	Between 2 and 4 years	(2y-4y)
5	Between 4 and 6 years	(4y-6y)
6	More than 6 years	(> 6y)

Table 5.2: Expertise scores.

It is interesting to notice that most of the subjects had their first contact with AspectJ in this course. On the other hand, two of the subjects used AspectJ in academic environment in the past six months, and other subject between six months and two years. A fourth subject actually used AspectJ in the past six months at industrial environment.

All the subjects have at least between six months and two years of experience in industrial software development. However, some of them had no contact with some technologies used during the study, such as RMI and JDBC. As previously mentioned, we provided exercises to help subjects that did not have enough experience with JDBC and RMI, reducing possible confounding factors.

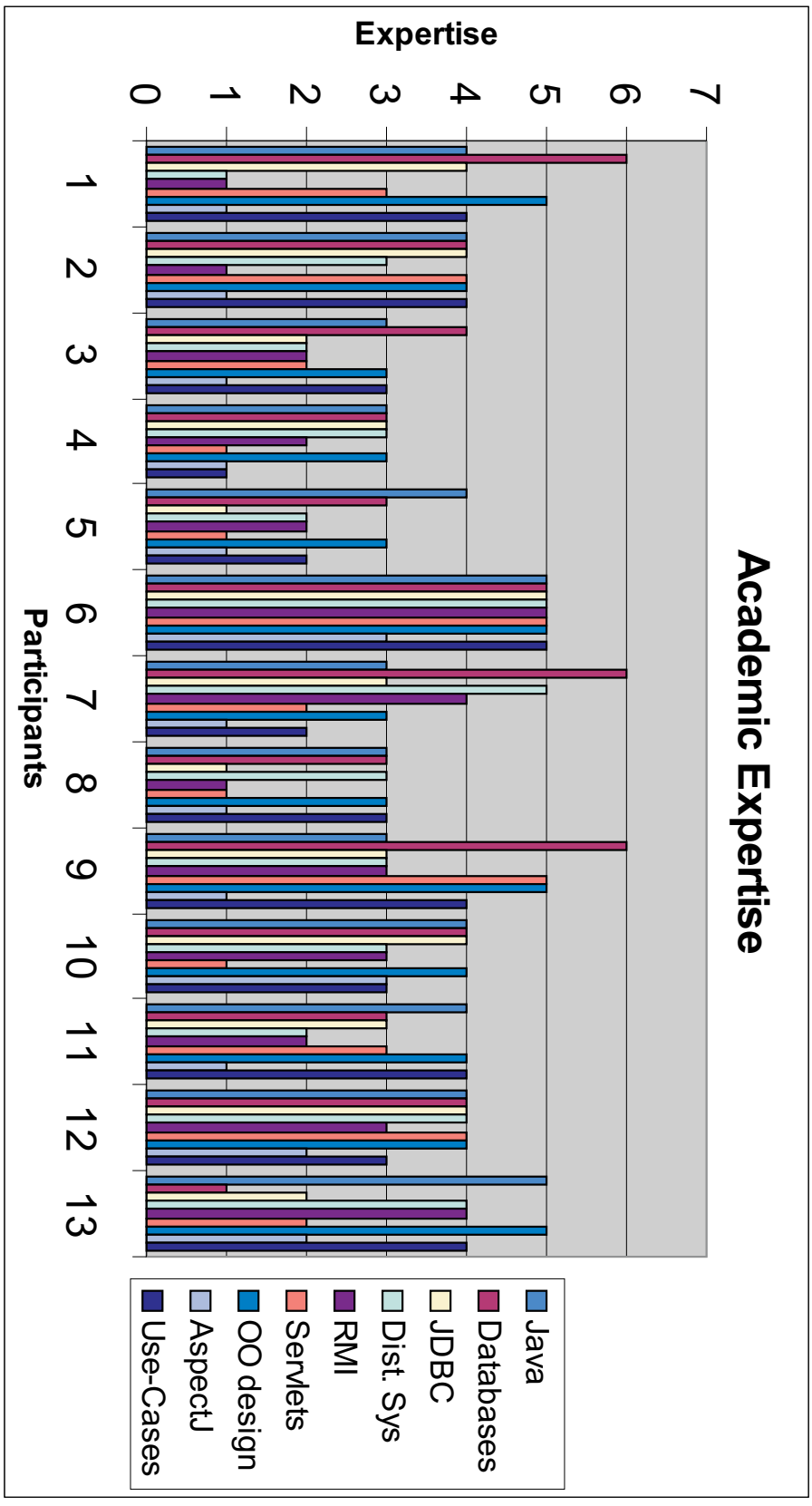

Figure 5.2: Subjects's academic expertise.

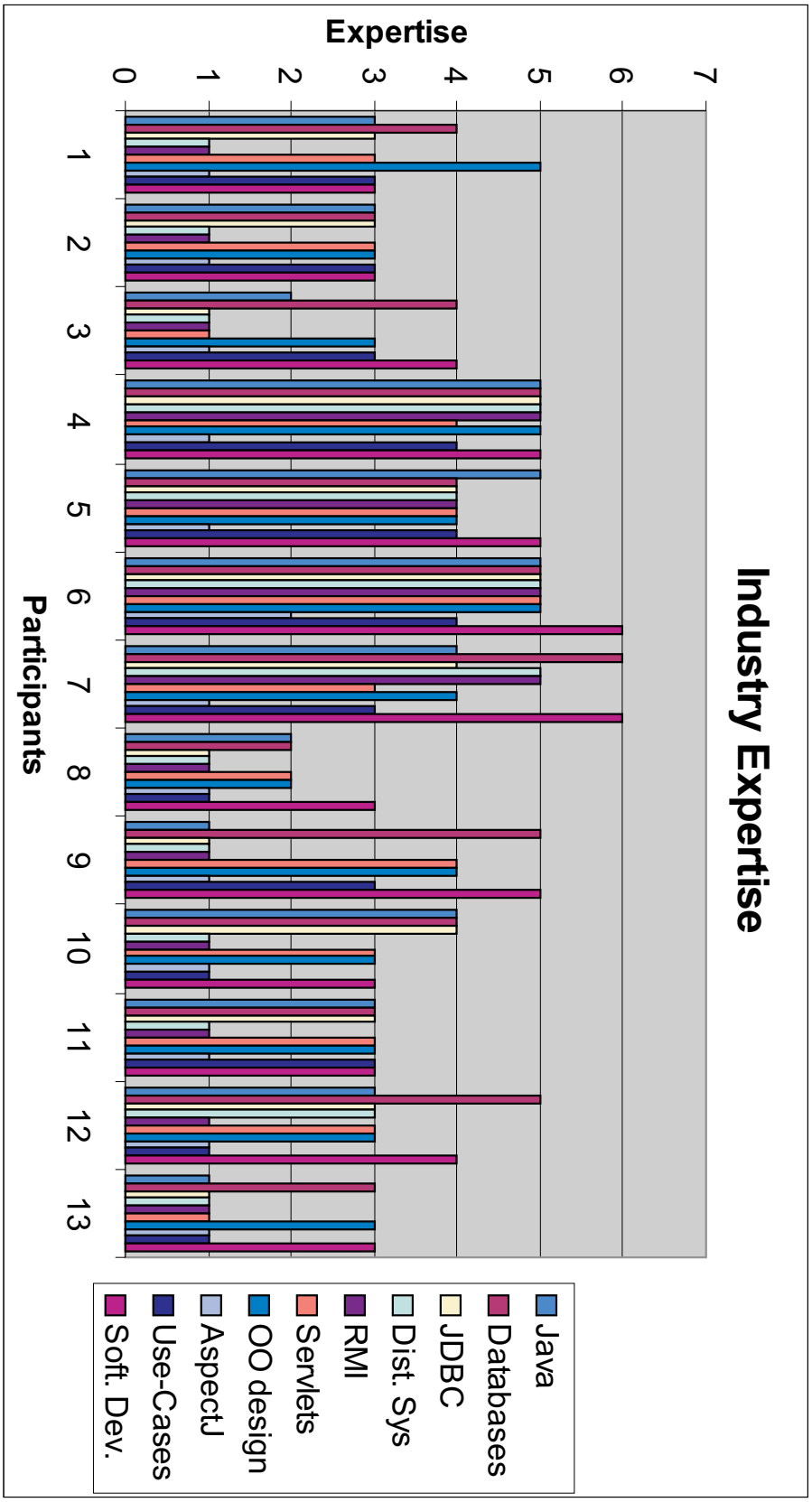

Figure 5.3: Subjects's industry expertise.

5.6.2 Study data

Tables 5.4, 5.5, and 5.6 present each subject's time, in minutes, to implement, test, and change requirements during the three iterations. The total time to complete each iteration is presented by Table 5.7. Tables 5.8 and 5.9 present the time, in minutes, to yield a functional prototype per use-case scenario for each subject. Table 5.10 presents the time in minutes to yield a functional prototype after requesting a requirement change in Scenarios 1 and 3. These tables use the legend presented in Table 5.3.

Code	Description
S. ID	Suject ID
Impl	Implementation
Test	Test
RC	Requirement change
Mean	Sample mean
Diff (%)	Difference in % from the other approach

Table 5.3: Tables legend.

NON-PROGRESSIVE			
S. ID	**Impl**	**Test**	**RC**
1	605	170	198
2	497	530	279
3	987	143	593
5	377	155	152
9	885	381	333
10	234	47	237
11	784	337	377
Mean	624	252	310
Diff (%)	+13.4	+41.4	+789.5

PROGRESSIVE			
S. ID	**Impl**	**Test**	**RC**
4	451	117	30
8	705	191	38
12	694	251	38
7	745	279	57
6	323	101	28
13	385	130	18
Mean	551	178	35
Diff (%)	-11.8	-29.3	-88.8

Table 5.4: Iteration 1 data.

The difference (Diff (%)) from the other approach can be read as an increasing or decreasing in time from the other approach. For example, Table 5.4 presents a increase of 789.5% in the requirement changes time when using the non-progressive approach, or a decrease of 88.8% in the requirement changes time when using the progressive approach.

NON-PROGRESSIVE			
S. ID	**Impl**	**Test**	**RC**
1	252	150	18
2	372	294	39
3	405	228	30
5	199	124	22
9	316	127	20
10	180	93	14
11	462	229	59
Mean	312	178	29
Diff (%)	+3.4	+10.7	+124.9

PROGRESSIVE			
S. ID	**Impl**	**Test**	**RC**
4	359	110	13
8	273	334	14
12	294	151	11
7	347	94	12
6	276	123	16
13	263	152	11
Mean	302	161	13
Diff (%)	-3.3	-9.7	-55.5

Table 5.5: Iteration 2 data.

NON-PROGRESSIVE		
S. ID	**Impl**	**Test**
1	218	69
2	242	218
3	358	110
5	163	63
9	171	274
10	237	61
11	529	91
Mean	274	127
Diff (%)	-3.7	+19.0

PROGRESSIVE		
S. ID	**Impl**	**Test**
4	229	123
8	368	167
12	234	118
7	300	124
6	271	50
13	306	56
Mean	285	106
Diff (%)	+3.9	-16.0

Table 5.6: Iteration 3 data.

NON-PROGRESSIVE			
S. ID	**IT 1**	**IT 2**	**IT 3**
1	973	420	287
2	1142	691	460
3	1718	661	468
5	674	338	226
9	1599	458	445
10	501	283	298
11	1427	743	620
Mean	1147.7	513.4	400.6
Diff (%)	+51.7	+8.9	+2.4

PROGRESSIVE			
S. ID	**IT 1**	**IT 2**	**IT 3**
4	592	479	352
8	921	614	535
12	975	452	352
7	1078	452	424
6	447	409	321
13	526	424	362
Mean	756.5	471.7	391.0
Diff (%)	-34.1	-8.1	-2.4

Table 5.7: Total iterations data.

	Use-case scenarios										
S. ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	603	172	38	45	131	188	53	53	70	61	50
2	628	235	76	76	315	185	100	80	123	82	75
3	937	188	67	100	159	305	62	102	146	87	71
5	358	164	64	48	81	123	46	28	66	51	35
9	920	346	51	75	134	178	71	116	107	119	32
10	281	23	50	40	109	70	31	126	85	35	21
11	839	211	85	49	264	286	97	166	158	130	69
Mean	652.3	191.3	61.6	61.9	170.4	190.7	65.7	95.9	107.9	80.7	50.4
Diff (%)	+480	+131	+83	+117	+459	+423	+412	+233	+141	+156	+159

Table 5.8: Times to yield pre-validation prototype without progressive approach.

	Use-case scenarios										
S. ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
4	107	89	68	35	34	37	9	28	43	28	51
8	136	136	34	41	31	37	17	38	60	58	13
12	136	11	13	15	51	35	20	22	42	23	15
7	141	179	32	32	18	77	6	30	52	28	8
6	74	18	24	27	28	17	10	24	33	42	7
13	81	63	31	21	21	16	15	31	39	10	23
Mean	112.5	82.7	33.7	28.5	30.5	36.5	12.8	28.8	44.8	31.5	19.5
Diff (%)	-83	-57	-45	-54	-82	-81	-81	-70	-58	-61	-61

Table 5.9: Times to yield pre-validation prototype with progressive approach.

NON-PROGRESSIVE		
S. ID	**Scenario 1**	**Scenario 3**
1	801	56
2	907	115
3	90	97
5	510	86
9	1253	71
10	478	64
11	1216	144
Mean	750.7	90.4
Diff (%)	+398.3	+94.5

PROGRESSIVE		
S. ID	**Scenario 1**	**Scenario 3**
4	137	81
8	174	48
12	174	24
7	198	44
6	102	40
13	119	42
Mean	150.7	46.5
Diff (%)	-79.9	-48.6

Table 5.10: Times to yield post-validation prototype.

5.6.3 Statistical analysis

As mentioned in Section 5.4 we use a t-test [40, 104] to analyze the data. The test consists in checking if the samples' mean difference is significant. This can be made by a hypothesis testing, and complemented by determining the confidence interval, as mentioned in Section 5.4.

Iteration	$H0_1$	$H0_2$	$H0_3$	$H0_4$	$H0_5$
1	TRUE	TRUE	FALSE	FALSE	FALSE
2	TRUE	TRUE	FALSE	FALSE	FALSE
3	TRUE	TRUE	—	FALSE	FALSE

Table 5.11: Null Hypotheses test.

We first used hypothesis testing to identify if the samples are significantly different. Table 5.11 presents the null hypotheses tests per iteration, where FALSE means the null hypothesis in question is false. When the null hypothesis is false, the sample means are significantly different. According to the presented results, at all iterations, implementation and tests time using a progressive approach are not significantly different from the non-progressive approach. On the other hand, the statistical test rejected the null hypotheses for the time to change requirements, and yield pre and post-validation prototypes.

Table 5.12 presents the values of the $|t_0|$ distribution of the t-test for each null hypothesis per iteration. The test consists in checking if $|t_0| > t_{\alpha,f}$, where $t_{\alpha,f}$ is the t distribution at a α percentage point, or significance level, using f degrees of freedom, where $f = n + m - 2$, in our case, $f = 6 + 7 - 2$ and $\alpha = 0.05$. Therefore, we should compare $|t_0|$ values with the $t_{0.05,11}$ distribution, which is 2.201. The $H0_4$ and $H0_5$ values in the third iteration are the same since there were no requirement changes.

Iteration	$H0_1$	$H0_2$	$H0_3$	$H0_4$	$H0_5$
1	0.555	0.985	4,551	5.960	3.497
2	0.221	0.385	2.470	4.649	4.858
3	0.188	0.522	—	4.601	4.601

Table 5.12: $|t_0|$ values for the Null Hypotheses test.

The $|t_0|$ values for the requirement changes in the first iteration and for the pre and post-validation prototypes in all three iterations are significantly different even using a higher confidence level. The $t_{0.025,11}$ distribution, which is the distribution for a 99% confidence level, or at the 0.025 significance level, is 3.106.

In order to allow a more effective analysis, we also determined the confidence interval, at the 0.05 significance level, presented by Table 5.13, for the mean difference when the null hypotheses could not be rejected by the test. We can notice wide confidence intervals, which indicate that the null hypotheses tests were obtained with a low precision. In fact, we expected benefits from implementing and testing the software with the progressive approach. When progressively implementing and testing functional

requirements, the programmer does not have to worry with persistence and distribution issues and vice-versa, decreasing implementation and test complexity. These wide confidence intervals suggest that other studies should be performed to better evaluate the progressive approach impact in implementation and tests.

Iteration	$H0_1$ — implementation time	$H0_2$ — test time
1	(-206.6 ; 353.9)	(-86.2 ; 233.6)
2	(-88.6 ; 109.2)	(-81.5 ; 115.9)
3	(-131.2 ; 109.9)	(-61.2 ; 101.6)

Table 5.13: Confidence interval for Hypotheses $H0_1$ and $H0_2$.

We can notice some interesting data at the table's data. Before commenting these data, it is important to mention that the first iteration is the hardest one, which can be noticed by the mean values of amount of time of each iteration, presented in Table 5.7. Notice the time to implement Iteration 1, which is higher than the sum of the others, in the non-progressive approach, and almost the sum of the other in the progressive approach.

Despite not being significantly different, there are some interesting data about implementation and testing times that might help understanding the study. For example, we can clearly notice a discrepancy at Table 5.4 regarding Subject 10's implementation and testing time, which were much faster than the others. In fact, implementation time was more than twice faster than the group mean, and testing time was more than five times faster. He definitely has superior programming skills, which can be confirmed by his academic and industrial expertise (Figures 5.2 and 5.3) that show he is one of the most experienced with AspectJ. On the other hand, his skills did not make difference when performing requirement changes, which reinforces the huge impact requirement changes might have on the development process. Such discrepancy did not happened with the progressive approach group.

Table 5.4 also presents a huge difference (789.5%) between the approaches' requirement changes time. This happened because the requirement changes at this iteration had a great impact in the persistence code. For example, one of them demanded changing the database scheme as well as writing migration scripts to move the data already inserted to the new scheme. On the other hand, the progressive approach does not implement persistent code until validating the functional requirement, when the changes are usually required. However, despite this huge difference at requirement changes, when considering the whole iteration time (see Table 5.7), the difference decreases to 51.7%, which is significantly different when performing a t-test. However, this value could be worst if we had more requirement changes.

Besides the null hypotheses there are also alternative hypotheses (see Section 5.2). Table 5.14 shows the alternative hypotheses test based on the values of the $|t_0|$ distribution already presented by Table 5.12. There are two alternative approaches set ($H1_{1..5}$ and $H2_{1..5}$) stating that the time to implement, test, change, and yield pre and post-validation prototypes using a progressive approach is different ($H1_{1..5}$), and smaller ($H2_{1..5}$) than using a non-progressive approach. When there is a significance difference between the approaches, the progressive approach times are smaller than the

Iteration	$H1_1$	$H1_2$	$H1_3$	$H1_4$	$H1_5$
1	FALSE	FALSE	TRUE	TRUE	TRUE
2	FALSE	FALSE	TRUE	TRUE	TRUE
3	FALSE	FALSE	—	TRUE	TRUE
Iteration	$H2_1$	$H2_2$	$H2_3$	$H2_4$	$H2_5$
1	FALSE	FALSE	TRUE	TRUE	TRUE
2	FALSE	FALSE	TRUE	TRUE	TRUE
3	FALSE	FALSE	—	TRUE	TRUE

Table 5.14: Alternative Hypotheses test.

non-progressive approach times (see sample data tables). For example, Table 5.6 shows that the implementation time of Iteration 3 is 3.9% worst when using the progressive approach. However, neither the null hypothesis nor the alternatives considered the mean differences for implementation time of Iteration 3 significantly different. In fact, this is the only case where the collected data shows a progressive time, in minutes, higher than the non-progressive time.

The data presented by Tables 5.5 and 5.6 show that these iterations were simpler than the first one. In fact, Iterations 2 and 3 had less use-case scenarios to implement and most of the aspects were implemented in the first iteration, which requires only simple modifications, as the software is incremented.

There was another requirement change in the second iteration, however, this change did not demand the same kind of tasks performed in the changes of the first iteration, such as changing the database scheme, and therefore, writing a data migration script. Despite being simpler than the first iteration, the single requirement change performed at the second iteration was also significantly different, according to Table 5.14, showing that the software development can be benefited from the progressive approach when there requirement changes during implementation activities. Therefore, although there is not a significant difference between the progressive and non-progressive approaches during implementation and tests activities, the progressive approach can be used to avoid unnecessary delay if there are requirement changes. Moreover, the progressive approach decreases the implementation complexity, since it does not deal with all the concerns at the same time, as discussed at the end of this chapter.

Another important data was about the times to yield pre and post-validation prototypes. The t-test showed that the times to yield pre and post-validation prototypes of all use-case scenarios (see Tables 5.8, 5.9, and 5.10) were significantly different, and therefore, the progressive approach has unbeatable results. In fact, this was expected since early validation of functional requirements is one of the pillars of the progressive approach, which is reached by first abstracting the implementation of some non-functional requirements. This early validation anticipates requirement changes, also helping to understand the problem before implementing some non-functional requirements. Moreover, the effort to create such prototype is lower, decreasing the budget impact, for example, if the requirements were not well understood by the requirements engineering or the customers had just change his mind.

5.6.4 Qualitative data

After finishing the study, we applied another questionnaire with more general questions in order to get some feedback from the subjects about the study, the technology used, and the implementation approaches. From a total of 13 subjects, 12 (92%) said that AOP and AspectJ helped the development, and 1 (8%) stated that AOP involves several new constructs, such as join points and pointcuts, that complicate debugging. He concludes that it is maybe better to use OO and design patterns. According to this subject, he had this feeling because he has a great experience with OO. The difficulty to learn a new paradigm (AO) was reported by 11 (85%) subjects. Another interesting information collected about the AspectJ language is about the debugging support, where 5 (38%) subjects complained about AspectJ's debugging. In fact, this is a limitation of the AspectJ tool used, but new versions of the tool have debugging support.

We also asked the subjects if they felt the progressive approach increases productivity. An expressive number of subjects, 7 (54%), explicitly said that they believe so, 2 (15%) suggested to believe, 3 (23%) gave neutral opinions, and 1 (8%) suggested that the progressive approach does not increase productivity.

Additionally, we asked to the 6 subjects that used the progressive implementation approach if they would use the approach in a real software development process. Again, the progressive approach had a great feedback from the subjects, where 4 (67%) would use the progressive approach just like they used in the study, and 2 (33%) would use variations of it.

5.7 Conclusions

In order to evaluate if the progressive approach is appropriate for a given project, a manager or development leader should balance the following forces derived from the study:

- The progressive approach helps to increase implementation productivity by early validating functional requirements, before implementing non-functional requirements. The study showed that there is a great difference between using and not using the progressive approach when there are requirement changes. In fact, requirement changes are common during implementation activities.

- Requirement changes might emerge from different stakeholders, specially after functional requirements validation, and the progressive approach avoids wasting effort writing non-functional code that is wrong or is supposed to be changed. Examples of causes of requirement changes are the following: requirement faults that lead to design faults, and therefore, to implementation faults; underestimated requirements that lead to bad design decisions; the lack of experience of developers; the customer keeps changing his mind, mainly when seeing a functional prototype.

- In fact, good practices and techniques in requirements, analysis, and design help to avoid some of these requirement changes. On the other hand, some requirement changes cannot be avoided by using good practices. For example, some

faults during requirements, analysis, and design activities cannot be identified until implementing the software. Furthermore, customer requests after seeing the prototype are unpredictable.

- The study showed that the time to yield a functional prototype using a non-progressive approach is larger to using the progressive approach in every use-case scenario, effectively providing early validation of functional requirements.

- Although the times to implement and test software using the progressive approach and the non-progressive approach in this study were not different, this means that there is not an overhead in the progressive approach implementation and testing activities. Therefore, the progressive approach can be used to perform requirement changes earlier, if technical problems are detected or customer requests are made after seeing the prototype, which avoids wasting effort writing non-functional code that is wrong or is supposed to be changed.

- Qualitative data from the subjects gave an important feedback about the progressive approach, which might suggest its use.

Besides all those conclusions, another contribution of this study is the documentation of the performed study, resulting in a study framework that allows replication in order to give more evidence of our results. In fact, some variables of this study might be changed to evaluate other factors.