ISSN 2007-9737

Do Novice Testers Satisfy Technique Prescription? An Empirical Study

Diego Vallespir, Carmen Bogado, Silvana Moreno

Universidad de la República,
Computer Science Institute,
Uruguay

{dvallesp, smoreno}@fing.edu.uy, cmbogado@gmail.com

Abstract. White box unit testing focus on examining code portions at unit level. Each testing technique considers and examines specific aspects of the code under test. We have studied whether the test case sets developed by novice testers who apply white box techniques meet the coverage criteria they prescribe. We have conducted an experiment that uses the Statement Coverage (SC) and All Uses (AU) techniques. 21 subjects applied the testing techniques under study, 10 applied SC and 11 applied AU. We analyzed the coverage level achieved by each test case set. 9 out of 10 subjects that apply SC managed to meet the coverage criterion prescribed by the technique. However, none of the 11 subjects who used AU did. This result has implications on professional practice and testing teaching and education.

Keywords. Test technique satisfaction, white box testing, test coverage, empirical study.

1 Introduction

Each white box testing technique prescribes the examination of certain code portions. The prescription of a white box technique is defined by the code coverage. The coverage determines which code portions are expected to be executed with the set of test cases generated with the technique. For example, the Statement Coverage technique prescribes that all the statements of the code should be executed at least once.

We shall say that a set of test cases *satisfies* the prescription of a testing technique if the set meets the coverage criterion it establishes. When a tester develops a set of test cases that satisfies a certain technique, we shall say that the tester satisfies the technique.

We assume that different white box testing techniques may be more or less difficult to satisfy. In other words, we believe that developing test cases that satisfy a specific technique might be more complex than developing test cases that satisfy another technique. For example, it seems more complex to develop a set of test cases that executes all the definition-use relations than one that executes all the statements.

The more complex the criterion established by a technique is it increases not only the difficulty in the development of the set of test cases, but also the risk that the set of test cases might not satisfy the coverage prescribed by the technique.

In this work we study whether novice testers who apply white box techniques manage to satisfy them and if there is any difference in the satisfaction of the prescriptions between techniques with different coverage criteria.

Satisfying a technique is important from the point of view of testing professional practice. On the one hand, not satisfying it might result in a decrease in the effectiveness of the technique. On the other hand, the fact that one technique might be more difficult to satisfy than another one might indicate the need for specific training for that technique. Testing is much more important now than before, considering, for example, continuous integration and continuous delivery. In these context, the continuous testing emerged as an important activity [13]. Continuous testing should

assess the coverage achieved by a tester, and it is important to know how easy (or not) it would be to achieve it.

From the point of view of research it is also important to know if the testers manage to satisfy the techniques. Many controlled experiments with testing techniques have used human subjects. Normally the conclusions of these experiments are: "Technique A has been more effective than technique B." In this context, it is worth wondering whether both techniques have been used correctly by the subjects, that is to say, whether the testers have managed to satisfy the techniques.

We have conducted an experiment with Statement Coverage (SC) and All Uses (AU) techniques. We have chosen a white box technique that is easy to apply (SC) and one that is much more complex (AU). We expect that using techniques of a very different degree of difficulty, in case there are differences in the level of satisfaction achieved by the testers, these will be more easily noticed.

In the experiment, a group of advanced under degree students of Computer Science of the UdelaR (Universidad de la República) develops test cases using SC and AU for testing a simple program written in Java. We analyze the coverage reached by the set of test cases developed by every subject to know if it satisfies the technique used to develop it.

This paper is organized in the following way. Section 2 presents the background of SC and AU techniques. Section 3 presents the related work. The experiment setup is presented in section 4. The results are presented in section 5 and the discussion in section 6. Section 7 presents the threats to validity. Finally, section 8 presents the conclusions and future work.

2 Background: Sentence Coverage and All Uses

The SC and AU techniques that are used in this experiment are presented in this section. SC is based on control flow and AU is based on data flow.

In order to satisfy the SC technique the set of test cases must execute, at least once, each sentence of the code. Since this technique is widely known, we do not go deeper into it here.

The AU technique expresses the coverage of testing in terms of the definition-use associations of a program. A *definition* of a variable occurs when a value is stored in the variable (*x := 7*). An *use* of a variable occurs when the value of the variable is read (used). This can be either a p-use or a c-use. A *p-use* is the use of a variable in a bifurcation of the code (*if (x==7)*). A *c-use* is when the use is not in a bifurcation. For example, in *(y := 7 + x)* there is a c-use of *x* (note that here there is also a definition of *y*).

The control flow graph is a representation through a graph of the different execution paths that can be executed in a program. The nodes of the graph represent the sentences (or code blocks) and the edges the bifurcations (if, for, while, etc). We will use i_j to indicate a specific node of the graph.

An execution path in the control flow graph can be represented as a sequence of nodes. For example, i_1, i_4, i_7, represents an execution path where node 1 is executed first, then node 4 and finally, node 7.

A definition of a variable *x* in a node i_d *achieves* a use of the same variable in a node i_u, if there is a definition clear path from i_d to i_u in the control flow graph, the path is executable and there is an use of *x* in node i_u.

A path i_1, i_2,...,i_n is a *definition clear path* for a variable *x* if the variable *x* is not defined in the intermediate nodes of the path (i_2,...,i_{n-1}).

AU requires that at least one definition clear path be executed from each definition (of every variable) to each achievable use (of the same variable).

The classical definitions of the techniques based on data flow and particularly AU are presented in an article by Rapss and Weyuker [14].

In Object Oriented languages the basic testing unit is the class. It is necessary to test its methods in an individual and in a collective way, so as to test the interactions generated through the sequence of calls originated by the invocation of a particular method. AU can be applied both for the tests of individual method belonging to a class and for the methods that interact with other methods of the same class or of other classes.

The tests of a class in AU can be carried out in two levels: Intra-method (Intra) and Inter-method (Inter). In **Intra**, only the method under test is considered for the code coverage. Therefore, in this case, the methods that interact with the method under test are not considered at the moment of developing the test cases. On the other hand, in **Inter**, the methods that interact with the method under test are considered for the code coverage.

Two types of definition-use pairs to be tested are identified in relation to these levels. The **Intra-method Pairs** are those which take place in individual methods and test the data flow limited to such methods. Both definition and use belong to the method under test. **Inter-method Pairs** occur when there is interaction between methods. They are pairs where the definition belongs to a method and the corresponding use is located in another method that belongs to the chain of invocations.

The subjects of the experiment that use AU technique apply the two types of use-definition pairs. In other words, in the context of this experiment, the AU technique is made up of the two types of pairs.

In most of the literature that presents techniques based on data flow, the examples that are given contain simple variables such as integers and Booleans. However, criteria that normally are not treated should be defined at the moment of applying these techniques in arrays or even more difficult in objects. Establishing these criteria is essential in order to know under which conditions the technique is applied. Different conditions can produce different results in the effectiveness and cost of AU since in fact, they are different techniques with the same name. Many of these conditions refer to how the Inter-method Pairs should be considered. This experiment establishes the conditions for the application of the AU technique based on what is proposed in [9, 10, 7].

We only present one of the set criteria as an example: how to consider the arrays. An assignment of an element in a position “x” of an array “a” (a[x]=something), consists of a definition of the variable “a” and a use of each variable that appears in the expression “x”. It should be noted that a use of the complete array is considered and not only the use of the element “a[x]”. For an occurrence of “a[x]” that is not a definition of “a” (y=a[x]), it determines a use of “a” and a use of each variable that appears in the expression “x”. This simplifies the tester’s work since we consider the use or definition at array level as a whole and not by element. That is to say, we consider there is a use of “a” without considering how “x” evaluates to determine which specific element of the array is being used or defined.

3 Related Work

There is a need to have empirical evidence about the effectiveness of the different testing techniques [8, 5]. However, after 25 years of experiments referring to testing techniques, Juristo et al. came to the conclusion that there is limited knowledge of them [11]. Runeson et al. also reach similar conclusions [16].

Within that limited knowledge little do we know about the extent to which testers manage to satisfy the prescriptions of the testing techniques. Counting on this knowledge is interesting from at least two points of view. On the one hand, it will provide an idea of how complex to apply the different testing techniques are. On the other hand, it might question the affirmations of different controlled experiments about the effectiveness and cost of testing techniques. It is worth mentioning that some experiments evaluate the application of the technique independently from whether the technique was used correctly or not. Therefore, the results of effectiveness observed might correspond to a bad application and not the real effectiveness of the technique if it is applied satisfying its prescription.

Basili et al., in one of the first experiments in testing techniques already established the importance of ensuring 100% coverage when evaluating a white box technique [2], some controlled experiments carried out later have not had that requirement.

Also, Briand et al. [6], claim that “*Another important practical issue when planning a testing experiment is related to the fact that when assessing a testing technique, we need to ensure that by the time the experiment completes, the*

testing technique has been fully and properly applied. In other words, if a coverage criterion is involved, we need to make sure we achieve 100 percent coverage if we want the cost-effectiveness results to be usable." We do not share this statement completely.

Although it is interesting to study the techniques when they are totally satisfied by the testers, it is also interesting to study them in their actual application (where many times they are not totally satisfied for different reasons).

The usefulness of the results will depend on how much the context of the experiment is specified, like, for example, whether or not the satisfaction of the prescriptions of the techniques was controlled by the testers of the experiment.

We have not found in the literature articles that specifically address the satisfaction of the techniques by the testers. However, we did find articles indirectly related to our work.

Several articles, and also a review of the literature show that the professional experience is one of the factors that affect the effectiveness of the testing techniques [2, 3, 1, 11]. One of the questions that arises is whether the experience is related to the satisfaction of the technique. Furthermore, is the experience related to the correct use of the technique and the correct use of the technique to the effectiveness?

Juristo et al. presented a study of a set of eight replications of an experiment, in which they differentiate the technique and its use by the subjects. The replications were not identical; there is a set of context factors that varied. They made the following propositions [12]:

- Lack of experience in programming diminishes the effectiveness of the decision coverage technique.
- Knowledge of the technique could affect its effectiveness.
- Techniques could be less effective if the subjects are not motivated.
- Techniques are more effective if the work is done in pairs.
- Work done under pressure does not seem to affect the effectiveness of the decision coverage technique.
- Tiredness does not seem to affect the effectiveness of the technique.
- Having previous information of the program does not seem to affect the effectiveness of the technique.

In the future, we might ask ourselves if each of these situations is related to the satisfaction of the technique. For example, does lack of experience in programming diminish the satisfaction of the technique by the tester because it does not reach the prescribed coverage? Or, if the work is done in pairs, does satisfaction of the prescribed coverage criterion of the technique increase compared to the work done by only one tester?

Barner et al. detected that the introduction of code coverage tools during tests improves the test coverage and that the impact made by the introduction of the tool was different among senior staff from junior staff [4]. Rothermel et al. present that the introduction of a coverage tool when the testers use the criterion definition-use, improves the coverage reached [15]. These experiments show that the use of code coverage tools might have an impact on the satisfaction of the technique prescription. They falso show that testers not always satisfy the testing techniques they use. In our experiment testers do not use code coverage tools.

4 Experiment Setup

4.1 Goals, Hypothesis and Metrics

The goal of our experiment is to compare the test cases developed by a group of testers using the SC and AU techniques as regards the satisfaction level of the coverage criteria prescribed by the techniques.

The satisfaction level of a coverage criterion of a set of test cases is defined as the coverage percentage achieved in relation to the one prescribed by the technique. This percentage is calculated as the number of items (of the

type prescribed by the technique) covered by the set of test cases divided by the total number of executable items (of the type prescribed by the technique) of the program under test. A set of test cases satisfies a technique when the satisfaction level of the coverage criterion prescribed by the technique is equal to 100%:

$$satisfaction\,level\ =\ \frac{covered\,items}{executable\,total\,items}\times 100.$$

The items depend on the technique that is being considered. For the SC techniques the items are statements in the program. One of the response variables of our experiment is the number of statements covered by a set of test cases developed by a tester who used the SC (M1) technique. Besides, we must consider the total number of statements of the code being tested (M2)

In the case of AU the items are the definition-use pairs that contain at least one executable definition-clear path, and both the intra-method and the inter-method are considered. A response variable of the experiment is the number of definition-use pairs with at least one definition-clear path covered by a set of tests developed by a tester who used the AU technique (M3). We should also consider the total number of definition-use pairs that contains at least one executable definition-clear paths of the code being tested (M4).

Our research question is "What is the level of satisfaction of a coverage criterion achieved by a set of tests developed by a tester?" From this question we derive the following null and alternative hypotheses.

The null hypothesis establishes that the median of the level of satisfaction of the techniques is the same. The corresponding alternative hypothesis indicates that the medians are different:

$$H0\ =\ Mdn(L.\,Sat.\,SC)\ =\ Mdn(L.\,Sat.\,AU),$$

$$H1\ =\ Mdn(L.\,Sat.\,SC)\ <>\ Mdn(L.\,Sat.\,AU).$$

The factor of this experiment is the testing technique and the alternatives are the techniques to be evaluated: SC and AU. The response variable considered in this experiment is the level of satisfaction of the coverage criteria. The metrics associated with this variable are M1, M2, M3 and M4.

The response variables are the following:

$$satisfaction\,level\,(SC) = \frac{M1}{M2},$$

$$satisfaction\,level\,(AU) = \frac{M3}{M4}.$$

4.2 Experimental Task

The experimental material consists of the program under test, its source code, its Javadoc and a script the subjects must follow when they perform the testing activity. The script is presented in this section, and the program is presented in the following.

The subjects use a script to make the tests during the experiment. This script is explained to the subjects during the training they receive as part of the empirical study.

The process the subjects must follow is presented in the script. The process is made up of three phases: Preparation, Design and Execution. This section presents a summary of the process which is presented in full in the Annex 5.

During the Preparation phase the subject must perform an initial checkup to guarantee that the test can be carried out. He must verify that he has the source files of the program and the Javadoc of the classes.

In the Design phase the subject develops the cases trying to satisfy the prescriptions of the testing technique assigned.

Once he has designed the test cases, the subject codifies them in JUnit. During this phase the subject might find defects in the program without executing even one test case.

When a subject finds a defect, the same as when he is in the Execution phase, he must warn a member of the research team in order to correct the defect. After the correction the subject continues his work. The aim of this way of working

is to simulate as far as possible a tester's real work. Testing in the industry is produced iteratively interacting with a code debugging task.

During the Execution phase the test cases are executed. This phase ends when there are no test cases that fail. While there are test cases that fail the subject must:

1. Choose a test case that fails.
2. Look for and find in the program the defect that produces the failure.
3. Request the correction of the defect from the research team.
4. Run the test case again to confirm that the correction was made correctly.

The correction of the defects found during the tests is not something usual in the experiments that study testing techniques. Incorporating the correction of defects simulates the practice of unit testing better. However, the correction of the defects found by the subjects was done by the research team in order not to lose control of the experiment. The aim of this is that when different subjects find the same defect its correction should be identical for all of them.

4.3 Experimental Object: The Program

The program used in this experiment is written in Java. It receives an array of integers as a parameter and it gives it back without repeated elements and ordered from small to large. It has two classes (Orderer and OrdererWithoutRep). The interaction between the classes is simple: the class OrdererWithoutRep invokes a method of the class Orderer for the array to be ordered before the repeated elements are eliminated.

The signature, specification and source code of the method *order* of the class *Orderer* and the method *orderWithoutRep* of the class *OrdererWithoutRep* are presented here (the specification and program is exactly what the students receive):

---

public static void order(int[] a)

The method returns the array ordered from smaller to larger. In case the array is null or empty, it remains unchanged.

a: entry parameter that contains the integers to be ordered.

```
public static void order(int[] a){
    for(int i=a.length-1; i>0; i--){
        int swapped = 0;
        int find = 0;
        for (int j=0; j<i; j++){
            if (a[j] > a[j+1]){
                int aux = a[j];
                a[j+1] = a[j];
                a[j] = aux;
                swapped=1;
            }
        }
        if (swapped == 0) {
            return;
        }
    }
}
```

---

public static int orderWithoutRep (int[] a)

The method returns the array ordered from smaller to larger and without repeated elements from the position 0 to the position a.length - the number of elements repeated - 1. And from there up to a.length -1 the values are unknown (that is to say, they are irrelevant).

```
public static int orderWithoutRep(int[] a){
    int countElim = 0;
    Orderer.order(a);
    for(int i=0; i<a.length-1; i++){
        if (a[i] == a[i+1]) {
            move(a, i+1);
            countElim++;
        }
    }
    return countElim;
}

private static void move(int[] a, int i){
```

```
    for(int j=i; j<a.length-1; j++){
        a[j]=a[j+1];
    }
}
```

Example : a = [5, 4, 5, 6, 6, 5]

Number of repeated elements = 3. Number 5 is repeated twice and 6 one. The array a (after the method is executed) from the position 0 to the position 2 is [4,5,6]. Position 2 is calculated as 6-3-1. From position 3 to 5 (a.length -1) the values of a are unknown. In case the array is null or empty, 0 is returned as the number of repeated elements and array a remains null or emtpy depending on the case.

A set of defects has been injected in order to simulate as much as possible the reality of the testing task. Which are the defects injected is not relevant to this research (they can be found in [19]) since they do not affect the satisfaction level of the coverage criterion achieved by a tester when applying one of the techniques.

Each correction of a defect by a researcher is a change in the original program. As different subjects find different defects, each one of them has a version of the program. 11 versions of the program were developed during this experiment. The number of lines of code goes from 20 to 23. The number of items to be covered for AU goes from 113 to 124. The difference in the number of items between SC and AU is not a threat to the validity of the experiment. For the same program it is normal that the AU items be more than the SC items. The number of items to cover is intrinsic to the technique and therefore should not be considered a threat to the validity of the experiment.

4.4 Experiment Design

The design of the experiment corresponds to a design of one factor (testing technique) with two alternatives (SC and AU). 21 subjects participate in it. 10 apply the SC technique and 11 apply the AU technique.

The design is slightly unbalanced due to the fact that the enrollment to the course is not mandatory and 21 people enrolled in it. It was decided to have one more person applying the AU technique.

All the subjects use only one technique (the assigned one). The assignment of the subjects to the technique is random. All the subjects work on the same program.

4.5 Subjects

The subjects of the experiment are undergraduate students of the Computer Science degree program of the Engineering School of the Universidad de la República (UdelaR), located in Uruguay. All of them are advanced students since they are in fourth or fifth year of the program. They have completed a Programming Workshop Course and an Object Orientation Course successfully.

All of them have completed a software engineering course in which, among other things, different testing techniques are studied. We consider that the group that participates in the experiment is homogeneous due to the fact that they are at a similar stage in the program and due to the training they received as part of the experiment.

The students take part in the experiment in order to get credits and this is one of their motivation. It is mandatory for them to attend the training sessions and they must also perform the testing technique following the material provided by the researchers. The students do not know that they are taking part in an experiment; they think they are taking a course with an important laboratory practice component.

However, at the end of the course, we asked the students for their consent to use the test cases their produced with research purpose. We fully informed the students about the research we were doing and how the data would be used. This is one of the most important ethical issues in empirical studies in software engineering [17, 20].

4.6 Training

The subjects who enroll in the course undergo training that aims at ensuring they have the necessary knowledge and practice to test the program of the experiment with the selected techniques. The training is made up of two parts: JUnit learning session and technique learning session. Figure 1 presents the different training activities.

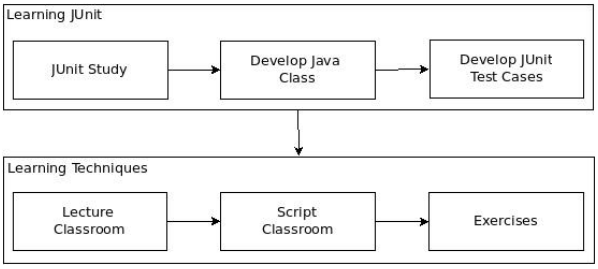

Fig. 1. Training of the subjects

The aim of the JUnit Learning session is for each subject to learn how to use the JUnit tool that they will be using to codify the test cases. As a task all the subjects are provided with a specification of a simple program and they are asked to implement that specification in Java and to test its functionality using Junit. The subject is supposed to study JUnit individually since this tool is not explained during the theoretical training classes.

This session is taken by the students out of the classroom and has a one-week duration. Once it has been completed, the students hand in the implementation in Java and the implementation of the test cases using JUnit. These submissions are revised by the researchers to check whether the student has acquired the necessary knowledge of JUnit to participate in the experiment.

Once the JUnit Learning session has been completed the Techniques Learning session takes place. This session aims at allowing the subjects to learn the SC and AU techniques. A theoretical/practical course of 9 hours is conducted during a day. In the first part of the day there is a theoretical class where the testing techniques to be used are presented and explained in detail.

The verification script they must follow when they perform the tests is also presented. During the second part of the day practical exercises to be solved in groups and individually are done. These exercises are practice in the use of the experiment techniques.

4.7 Operation

The tests on the experiment program are performed in a session of one day: 4 hours in the morning and 5 in the afternoon with a one-hour pause to have lunch. This session takes place seven days apart from the training session. In this session the subjects individually apply the technique allocated to each one, developing the necessary test cases and executing them.

In order to complete the work the subjects follow the script provided in the Learning Techniques session. At the end of the session the subjects hand in the JUnit classes developed and the notes they have made in order to be able to apply the technique (control flow graphs, identified paths, etc). Once all the works have been submitted, the researchers revise them and return them individually to each student.

Both the training and the tests on the program were conducted twice. This was due to the fact that the students who enrolled for the first experiment were only 10, so we decided to make a replication. The first one had 10 subjects (5 SC and 5 AU) and the second one had 11 subjects (5 SC and 6 AU). The two training sessions and the execution took 3 weeks in total. There were only a few weeks between the works done by one group and the other. All the conditions (trainers, researchers, materials, training, etc) were identical for the two instances carried out. Due to this, data analysis is done on the total of subjects that participated (10 SC y 11 AU).

During the execution of the experiment no significant deviations from what was planned were found.

5 The Testing Script

In tables 2-4, we present the scripts of white box techniques used in the experiment. The script describes the process to perform the tests by the subjects. It involves three phases: Preparation, Design and Execution.

6 Results

6.1 Descriptive Statistics

The total of items of the program under test for SC depends on the final version of the program each subject has. As it has already been mentioned, those totals go from 20 to 23 items (LOCS counted by the CodeCover tool).

This situation is the same with the total of items of the program when the AU technique is considered. In this case is the number of definition-use pairs that have at least one executable definition-clear path. This measurement was done manually. The different variants of the programs contain from 113 to 124 items.

The number of items covered by each set of test cases the subjects developed is calculated. Table 5 shows the proportion (ratio) of items covered in relation to the total items the final version of the program has for each one of the subjects (satisfaction level of a coverage criterion expressed in percentage).

Figure 2 presents the box and whisker graph of the level of satisfaction achieved using SC and using AU. It is clear that the distributions of both techniques are very different. Only one subject in SC does not manage to satisfy the technique and his level of satisfaction is over 90% . On the other hand, the AU technique has a minimum of 56% and a maximum of 98%.

Table 6 presents the number of subjects that used each one of the two techniques, the median of the level of satisfaction, and the interquartile range.

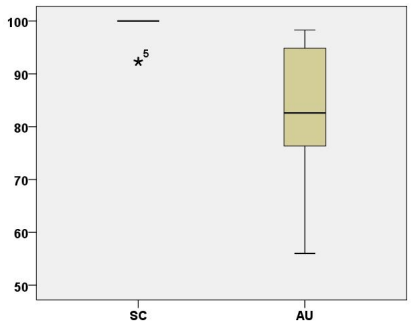

Fig. 2. Box and Whisker of the Level of Satisfaction of the Techniques

6.2 Hypothesis Testing

The observations we have (10 and 11) are too few to carry out parametric tests. Therefore we applied the non-parametric Mann-Whitney U test. The null hypothesis indicates that the medians of the satisfaction level of both techniques is the same (see 4).

The Mann-Whitney test indicates that there is statistical evidence to reject the null hypothesis. The significance level obtained is of 0.0002. That is to say, with more than 99% confidence we show that **there is a significant difference in the satisfaction of the degree of coverage the testers achieved applying SC and AU, being significantly higher the satisfaction of SC**.

7 Discussion

The results are compelling. 90% of the subjects who used the SC technique managed to satisfy the prescribed coverage criterion. On the other hand, none of the subjects who employed the AU technique managed to satisfy it (the maximum level of satisfaction was 98.3%). This clearly indicates that satisfying AU is much more complex than satisfying SC. In other words, developing test cases sets that satisfy AU is more complex than developing test cases sets that satisfy SC.

The aim of this work is to know whether novice testers with training in the techniques are able to comply with the prescriptions of the techniques when they develop test cases. This experiment

Table 1. Script of the testing process

Step	Phases	Description
1	Preparation	— Prepare activities for verification.
2	Design	— Develop the test cases that meet the criterion of the requested technique. — Implement the cases in JUnit. — Record the required data of the phase.
3	Execution	— Execute the designed test cases. — Find the defects associated with the test cases that fail. — Record the required data of the phase.

Table 2. Script for the Preparation phase: Conduct an initial check up guaranteeing that the required verification can be carried out

Step	Activities	Description
1	Files	— Verify you have the source files to test. — Verify you have the Class Diagram. — Verify you have the javadoc of the classes to test. — Verify you have the defects and time record form.

Table 3. Script for the Design Phase: Design the test cases satisfying the criterion of the testing technique to apply. The cases must be designed in bottom-up form

Step	Activities	Description
1	Define the Data Set	— Record the start time of this activity. — For each method to test define the set of input values that meet the established criterion.
2	Define the expected results	— Define the expected outcome (or expected behavior) for each element of the Set of Data.
3	Reduce	— Eliminate the test cases that cannot be executed (impossible paths, etc.) Check whether the criterion is still met. In case it is not met, return to step 1 trying to meet the criterion.
4	Implementation of the test cases in JUnit	— Implementation of all the test cases designed in JUnit.
5	Recording of the end of design	— Record the end of design time.

NOTE 1: If there are pauses during design they should be subtracted from the total time. Due to this the total time of the pauses during design should be entered.

NOTE 2: Should any defect be found during the design phase it must be recorded as it is explained in the execution phase.

Table 4. Script for the Execution phase: Execute the designed test cases. The cases must be executed bottom-up

Step	Activities	Description
1	Execute test case	— Execute the test cases
2	Analize the obtained outcome	— If no test case has failed the phase is finished.
3	Find defects	— While there are test cases that failed: – Choose one of the test cases that fail. – Search and find the defect in the program that originates the failure. A debugging tool can be used. – Record the defect following step 4. – Request the correction of the defect from the research team. – Execute the test case again to confirm that the correction by the research team has been done properly. – The correction changed the program. So, you should analyze if you need to design more test cases in order to meet the stablished criterion (use the design phase script again)
4	Recording of Defects	— For each defect found record the following data: – General description of the defect. It is important that the description should be clear and precise. – Name of the file that contains the defect. – Line in which the defect is found. In case the defect is not in a specific line enter 0 (zero). Warning: if the file being testes was modified in relation to the original (because a defect was corrected), the line of the original file must be indicated. – Structure associated with the defect (ej.: IF, FOR, WHILE, name of the method, etc.). If the defect does not have an associated line this field must be filled in compulsorily. – Beginning line of the structure (compulsory if an associated structure is indicated.).

shows that the answer depends on the technique under consideration. It can be stated that the testers normally manage to satisfy the SC technique when they apply it (at least for very small programs such as the one used in this experiment). However, normally they do not satisfy the AU technique; particularly in this experiment none of the testers managed to satisfy the AU technique (not even for a very small program). We believe that in more complex programs the difference in satisfaction of the techniques will increase even more. In fact, AU will be even more difficult to satisfy than SC.

These results have impact on the practice of unit testing, on the empirical research of white box testing techniques, and in the way we teach the techniques. Learning how to manage techniques like AU is not easy.

If a software development team decides to perform complex unit tests (like AU), it is investing more time in the design of the test cases than it would if it used simpler techniques (like SC). The cost analysis of the execution of these techniques is presented in another article [18]. Such analysis presented that the cost measured in minutes to design the test cases using the AU technique was almost 2.5 times longer than for the SC technique.

Apart from the fact that AU is more expensive than SC, it is probable that the coverage criterion prescribed by the technique is not being satisfied. Therefore, the team might not get the expected

Table 5. Level of Satisfaction of the Coverage per subject in Percentage

Technique SC			Tecnnique AU		
Sub.	Ratio	%Sat.	Sub.	Ratio	%Sat.
1	21/21	100.0	11	91/116	78.4
2	20/20	100.0	12	93/117	79.5
3	21/21	100.0	13	114/116	98.3
4	20/20	100.0	14	65/116	56.0
5	20/21	92.3	15	100/121	82.6
6	20/20	100.0	16	84/113	74.3
7	21/21	100.0	17	111/116	95.7
8	23/23	100.0	18	112/124	90.3
9	22/22	100.0	19	109/116	94.0
10	23/23	100.0	20	115/120	95.8
			21	88/124	71.0

Table 6. Median and Interquartile Range of the Level of Satisfaction of the Techniques

	#Subjects	Median	IQR
SC	10	100%	0
AU	11	82.6%	21.4

return as regards effectiveness since the real coverage is less than the theoretical one. In the same article mentioned before [18], we show that we could not distinguish with statistical validity between the effectiveness of these two techniques. Would it be possible that AU were not more effective than SC because the testers did not manage to satisfy its prescription?

These results also have an impact on the experiments that compare the effectiveness of different testing techniques. Normally in these experiments it is concluded, with statistical evidence, that a certain technique is more effective than another. In the light of the results we have obtained, it is the application of the techniques by the testers and not the technique itself that is more or less efective

In other words, if in an experiment that aims at knowing the effectiveness and/or cost of different testing techniques with human subjects applying different techniques, AU turns out to be less effective than SC; it should not be held that AU is less effective than SC. Given the results we have obtained, in fact the degraded version of AU the subjects apply is less effective than the very close version to the SC prescription.

8 Threats to Validity

There are several threats to the validity of this experiment. Due to this, it is important to replicate it in order to find out if the conclusions can be generalized. The threats we consider most important are mentioned below.

The subjects are all under-degree students. Although they are all advanced students and they received careful training, they are not professionals in software development. This might mean that the test cases developed be “worse” than the ones an expert might develop.

Only one program is used in the experiment. Therefore, the obtained results might be due to specific features of the program and not of the studied techniques. It is necessary to replicate the experiment with different types of programs in order to generalize the results.

The program is very small in terms of number of lines of code. It is necessary, like in the previous case, to conduct replications with programs of different size so as to observe if the same results are obtained.

The obtained results are only applicable to SC and AU. However, it seems that the complexity inherent to the technique might serve as an indicator of the level of satisfaction of the coverage criterion that will be achieved. Future replications should consider other white box techniques to confirm it is really the inherent complexity and not the specific technique which affects the level of satisfaction.

9 Conclusions and Future Work

In this article we present an empirical work that aims to know if novice testers manage to satisfy the coverage criteria prescribed by white box techniques. In our particular case we studied the SC and AU techniques.

We found that novice testers normally satisfy the SC technique while they are unable to satisfy the

AU technique. 90% of the subjects that applied SC managed to satisfy the technique. On the other hand, none of the subjects who applied AU managed to satisfy it. We found statistical evidence, applying the Mann-Whitney test, that the SC technique has a higher level of satisfaction than the AU technique.

Our future work consists in replicating this experiment trying to eliminate the most important threats to validity that were presented. It is important to have different programs and of different sizes, and to use both professionals and students in the experiments and to apply other white box techniques.

References

1. **Armour, P. G. (2005).** The unconscious art of software testing. *Commun. ACM*, Vol. 48, No. 1, pp. 15–18.
2. **Basili, V. & Selby, R. (1987).** Comparing the effectiveness of software testing strategies. *Software Engineering, IEEE Transactions on*, Vol. SE-13, No. 12, pp. 1278–1296.
3. **Beer, A. & Ramler, R. (2008).** The role of experience in software testing practice. *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, pp. 258–265.
4. **Berner, S., Weber, R., & Keller, R. (2007).** Enhancing software testing by judicious use of code coverage information. *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pp. 612–620.
5. **Bertolino, A. (2007).** Software testing research: Achievements, challenges, dreams. *2007 Future of Software Engineering*, FOSE '07, IEEE Computer Society, Washington, DC, USA, pp. 85–103.
6. **Briand, L., Di Penta, M., & Labiche, Y. (2004).** Assessing and improving state-based class testing: a series of experiments. *Software Engineering, IEEE Transactions on*, Vol. 30, No. 11, pp. 770–783.
7. **Frankl, P. G. & Weyuker, E. J. (1988).** An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, pp. 1483–1498.
8. **Harrold, M. J. (2000).** Testing: a roadmap. *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, ACM, New York, NY, USA, pp. 61–72.
9. **Harrold, M. J. & Rothermel, G. (1994).** Performing data flow testing on classes. *SIGSOFT Softw. Eng. Notes*, Vol. 19, No. 5, pp. 154–163.
10. **Harrold, M. J. & Soffa, M. L. (1989).** Interprocedual data flow testing. *TAV3: Proceedings of the ACM SIGSOFT '89 third Symposium on Software Testing, Analysis, and Verification*, ACM, New York, NY, USA, pp. 158–167.
11. **Juristo, N., Moreno, A. M., & Vegas, S. (2004).** Reviewing 25 years of testing technique experiments. *Empirical Softw. Engg.*, Vol. 9, No. 1-2, pp. 7–44.
12. **Juristo, N., Vegas, S., Solari, M., Abrahao, S., & Ramos, I. (2012).** Comparing the effectiveness of equivalence partitioning, branch testing and code reading by stepwise abstraction applied by subjects. *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pp. 330–339.
13. **Mascheroni, M. A. & Irrazabal, E. (2018).** Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review. *Computación y Sistemas*, Vol. 22, No. 3, pp. 1009–1038.
14. **Rapps, S. & Weyuker, E. J. (1982).** Data flow analysis techniques for test data selection. *ICSE'82: Proceedings of the 6th international conference on Software engineering*, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 272–278.
15. **Rothermel, K. J., Cook, C. R., Burnett, M. M., Schonfeld, J., Green, T. R. G., & Rothermel, G. (2000).** Wysiwyt testing in the spreadsheet paradigm: an empirical evaluation. *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, ACM, pp. 230–239.
16. **Runeson, P., Andersson, C., Thelin, T., Andrews, A., & Berling, T. (2006).** What do we know about defect detection methods? [software testing]. *Software, IEEE*, Vol. 23, No. 3, pp. 82–90.
17. **Singer, J. & Vinson, N. G. (2002).** Ethical issues in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, Vol. 28, No. 12, pp. 1171–1180.
18. **Vallespir, D., Bogado, C., Moreno, S., & Herbert, J. (2010).** Comparing verification techniques: All

uses and statement coverage. *Ibero-American Symposium on Software Engineering and Knowledge Engineering*, pp. 85–95.

19. **Vallespir, D. & Herbert, J. (2009).** Effectiveness and cost of verification techniques: Preliminary conclusions on five techniques. *Computer Science (ENC), 2009 Mexican International Conference on*, pp. 264–271.

20. **Vinson, N. G. & Singer, J. (2008).** A practical guide to ethical research involving humans. In **Shull, F., Singer, J., & Sjøberg, D. I. K.**, editors, *Guide to Advanced Empirical Software Engineering*. Springer London, London, pp. 229–256.

Article received on 04/03/2019; accepted on 16/08/2019. Corresponding author is Diego Vallespir.