An Experience on Applying Software Testing for Teaching Introductory Programming Courses

Maria A. S. Brito, João L. Rossi, Simone R. S. de Souza, Rosana T. V. Braga
Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação
São Carlos (SP), Brazil, 13560-970
{*masbrit,rossi,srocio,rtvb*}*@icmc.usp.br*

Abstract

Previous work has demonstrated that the use of software testing concepts for teaching introductory programming courses may be a good practice for the students. Indeed, these courses provide limited opportunities for the teacher regarding dynamic activities that could help students in the learning process. This paper describes an approach in which test sets are reused in the context of teaching introductory programming courses, as an alternative to increase the quality of the programs generated by students. An experimental study was carried out to investigate the impact of reusing test cases during the programming learning. The objective is to evaluate if the use of test cases might improve the quality of programs implemented by the students. Sixty undergraduate students participated in the experiment, implementing programs in the domain of vectors. A set of reference programs was used to generate test cases, based on functional testing, to be reused by the students to test their programs. Considering a range of $1 - 10$, the programs' quality increased from 5.3 to 7.4 using this approach. The results provide evidences that the reuse of test cases during introductory programming courses may help to increase the quality of the programs generated by students, motivating them to apply software testing during the development of the programs.

Keywords: Experimental study, Functional testing, Reuse, Teaching introductory programming courses

1 Introduction

The teaching of introductory programming courses differs from the teaching of other subjects, because in order to teach programming, first it is necessary to teach how to solve problems and build ideas [1]. Often, the students' inexperience in programming logic implies that teachers need to find ways to improve their ability to understand the logic behind the mere codification.

In traditional teaching methods, students initially receive the rules (programming language structures) so that problems can be solved. Then the teacher provides the student the program specification and (s)he waits for the student to build a program able to solve the proposed problem. In this case, the student has got just the program specification and, based on it, (s)he should achieve the expected result, which is the code of a correct program. In such approach, the student creates and applies some tests which (s)he judges enough and admits that the program is correct.

This paper presents a study where we propose an approach for teaching in which the student receives from the teacher not only the problem specification, but also a test set that should be used during the testing. The test set is suitable for a reference program which is equivalent to the one that will be created by the student. Two programs P_1 and P_2 are defined as equivalent if the same output o_1 resulting from processing the input i_1 to P_1 is obtained by the program P_2 considering the input i_1 [2]. However, the functionality in equivalent programs can be implemented in different ways. Based on this idea, when the program is built, the student can reuse the test cases and evaluate whether the program under construction is free of defects. Thus, the method contributes to improving the quality of the programs generated by students. Quality here refers to the conformancy of the program implementation with its specification, i.e., the program implementation is totally compliant with its specification. Another aspect of this approach is that, as the student learns to program, at the same time (s)he understands the importance of software testing activity. This is an activity with a small portion allocated in the Computer Science curriculum in comparison to other activities of the software development process [3].

The experimental study performed in this work involved two programming themes: vectors and matrix manipulation and was conducted with 60 undergraduate students of two computer science courses at ICMC/USP (Institute of Mathematics and Computer Science/University of São Paulo). In each classroom, students were randomly divided into two groups, in which a group of students received only the program specification and the other group received the program specification and the test sets. Each group was instructed to use the specification to build a program in conformance to it. The group that has received the test set was instructed to use it to validate their programs and to record the results of this validation. At the end, 60 programs were implemented by the students. The ANOVA statistical tests were applied to evaluate the experiment hypotheses.

This paper is structured as follows. In Section 2, we describe related work on the reuse of software testing and experimental studies. In Section 3, we present the important concepts about software testing that are required to understand the present work. In Section 4, we describe details about our experimental study and, in Section 5, the results obtained are discussed. In Section 6, we present the validity threats of our study. In Section 7, we outline a proposal for a data repository to ease the use of testing during programming learning. Finally, in Section 8 we present our conclusions and future work.

2 Related Work

Although there are papers reporting empirical studies about reuse of test cases [4, 5, 6], in the teaching context there are many works about the insertion of software testing as a resource to teach programming.

In Schulte et al. [7] the authors compare and contrast the way in which different models conceptualize program comprehension from an education point of view. Jones [8, 9, 10, 11] is the most outspoken educator on this area who suggests systematically incorporating testing practices across the undergraduate curriculum, thinking that the students can be more prepared for real world testing tasks upon graduation. Stephen [12, 13] describes the use of software testing to teach programming introdutory courses with the purpose of encouraging students to write tests as they code (in the spirit of TDD – Test-Driven Development).

Few works report empirical studies about reuse of test cases in teaching introductory programming courses. In Souza and Barbosa [14] it is presented an environment for submission and evaluation of practical work performed by students. The students submit their program and their test cases and the environmet analyses them using a reference program and a test case set, defined by instructor. The objective is to evaluate the coverage of the student´s test cases in relation to reference program and the coverage of the instructor´s test cases in relation to student´s program. Thus, the tool allows the reuse of test cases generated by the instructor to verify the correctness of the student's program. Differently of our study, this approach is defined to evaluate the program correctness, mainly related to the coverage of testing criteria. Currently, this environment is configured for Java programs. In [15] is presented a tool, in which the student's programs are tested using test cases generated by the instructor. The tool MARMOSET [16] uses an approach that is similar to our work as it uses test cases generated by the instructor, but it allows students to create their own tests. The tests generated by the instructor are delivered to the students in three different moments: before they start to code their programs, during the implementation (with limitation in how many tests they can perform) and after they finish the program implementation. The authors believe that if part of the tests are a limited resource, this will encourage students to start tests earlier in their projects. This approach is also defined to evaluate the program correctness and the generation of test cases by students.

Campanha et al. [17] describe an experimental study where test case sets are reused to validate alternative versions of a program. In this study, two master students developed reference programs for sorting algorithms, such as bubble sort, selection sort and insertion sort, and subsequently, they generated, for each program, test cases sets adequated to the Mutation Testing. In an introductory programming course, undergraduate students implemented different programs to solve the data sorting problem. These programs were tested using the test case sets generated from the reference programs, evaluating if this sets are adequated to test equivalents programs. The authors concluded that the reuse of test sets adequated to Mutation Testing to validate equivalent programs can be truly effective, reducing the testing activity costs. Our study, presented in this paper, is inspired on the Campanha et al. [17] study.

3 Software Testing

The software development process involves the use of methods, techniques and tools, but this does not prevent the occurrence of errors in the product in development. In order to reduce these errors, there are VV&T - Software Verification, Validation and Testing activities, which are employed during the software development cycle. Verification is the process of identifying whether the software construction is being done in the right way. Validation involves evaluating whether the software developed meets the requirements proposed by the client. The testing activity involves discovering the presence of defects, if they exist [18].

The testing activity should be present throughout the development cycle, and it has the following testing phases: 1) unit testing, in which the smallest units of the software are tested, such as procedures, functions or methods and classes; 2) integration testing, where the integration between modules, components or other forms of units of the software is tested, and; 3) system testing, in which the system is tested as a whole. The tests applied at this phase include stress testing, security testing, load testing, among others. In addition to these three phases, there is also the regression testing performed when the program is modified and it is necessary to guarantee that previous tests are still applied, and also to test new requirements introduced by modification [19].

A test case is a pair formed by one or more input values and an expected output for this entry. A test criterion defines which properties or requirements need to be tested to evaluate the test cases quality [20]. The selection of test cases should be performed based on testing criteria, which are divided in different testing techniques: structural, functional and error-based testing [19]. In the functional technique, the software is considered a black box and the test cases are built based on the software specification. For the structural testing, the internal structures of software are tested, so the test cases are built based on the software implementation. In the error-based testing technique, the most common faults that can be made during the development process are used for selecting the test cases.

The Equivalence Partitioning Functional criterion divides the domain of program entry data in classes, in which the test cases are derived [21, 18]. In this criterion, the division is carried out into valid and invalid equivalence classes and the objective is to produce test cases that cover each class, reducing the total number of test cases needed. The principle is that, if a test case of a particular equivalence class reveals an error, any other test case in this class should reflect the same error. The Boundary Value Analysis Functional criterion complements the Equivalence Partitioning through test cases that exercise boundary values, for instance, the biggest value of a variable. Boundary values refer to situations that occur within an equivalence class, directly above or below.

The experiment described in this paper uses the functional testing criteria to generate the test cases for the reference programs. As the programs developed by students were equivalent to the reference programs, we believe that these testing criteria are the best choice to generate the test cases for this experiment. The functional test sets are simpler to generate, facilitating the reuse and the use of this approach in introductory programming courses.

4 Experimental Study Planning

The experimental study was defined based on the Wohlin et al. [22] process. Next, we summarize the experiment planning.

Experiment Definition:

To investigate the reuse of test case sets as an mechanism to teach programming, checking whether or not this mechanism allows an increase in the quality of the programs implemented by students in introductory programming courses.

Context:

Evaluation of programs for vectors and matrices manipulation, implemented in *C* language by undergraduate students of computer courses. The master student who is leading this experiment defined the reference programs and the test case sets to be reused by the subjects. The experiment was performed by two groups of students (subjects) randomly selected from each classroom; two programs for vector and matrices manipulation P_1 and P_2 and functional test case sets T_1 and T_2 adequate to test the reference programs. During a practice class, half the subjects received only the program specification, while the other half additionally received the test case sets T_1 and T_2 to validate their implementation. The test case sets were given only after the students finished the program implementation. The evaluation of the programs developed by the subjects was done by applying a score, ranging from 0 to 10, considering the correctness of the program in relation to test case set.

Hypotheses:

- *Null Hypothesis* H_0: Students who received test case sets T_1 and T_2 will produce programs whose quality is equal to or lower than students who received only the program specifications.

- *Alternative Hypothesis* H_1: Students who received test case sets T_1 and T_2 will produce higher quality programs than students who received only the program specifications.

Although our interest is more focused on the effect of test case sets in the quality of the generated programs, hypotheses considering other factors were also taken into consideration. They are described in Section 5.

Independent Variables:

Program specifications, reference programs, test cases sets, questionnaires and programs implemented by the students.

Dependent Variables:

Score of the programs implemented by the students and answers to the questionnaire.

Subjects:

Sixty undergraduate students from *Introduction to Computer Science* courses of the Institute of Mathematics and Computer Science of University of São Paulo, divided in three groups: classrooms I (28 students) and II (16 students) formed by students of the course Bachelor of Computer Science and classroom III (16 students) formed by students of the course Bachelor in Informatics. Each of these 60 students developed his/her individual version of the programs to be evaluated. A master student who is leading this experiment generated the test cases sets for a reference program set.

Experiment Description:

The programming topic used in the experiment (vectors and matrices manipulation) was previously taught for the students, in this way, it was considered that the students had the necessary knowledge to develop the required programs. We did not provide training on software testing. During the experiment, subjects from each classrooms were randomly separated into two groups. (in this case, following the alphabetical order).

All subjects in the classroom completed a questionnaire, providing information about sex, background and previous experience in programming, which was used in the analysis. Then, one group received only the programs specification and the other one received also the test cases sets (these were distributed only after the programs were implemented). Each subject was responsible for implementing each program from specification. The subjects that received also the test case sets, had to use them to test the programs and take notes of the results of the tests (if they fail, then they fixed the program and tested again). The subjects that did not receive the test case sets could test the programs in an ad-hoc way.

This activity was carried out during a classroom practice, under teacher supervision. The duration of the classroom is as follows: classroom I - 1 hour, classroom II - 2 hours and classroom III - 2 hours. In this phase of the experiment it was not allowed the communication between the students and the consultation of other materials. Before this activity, the master student implemented the reference programs, based on the specifications available in [23]. Then, she generated test case sets to be reused by the subjects, based on the Equivalence Partitioning and Boundary Value Analysis functional criteria. The experiment material were all available electronically and hence could be accessed easily[1].Table 1 shows the programs selected for the experiment, the programs size (lines of code) and the number of test cases generated for each one. The table also shows which programs are used in each classroom.

Table 1: Programs used in the experimental study

Programs	**Classroom**	**LOC**	**Test Cases Set**
Vector manipulation	I	31	9
String manipulation	II, III	29	6
Matrix manipulation	II, III	42	11

In the second part of the experiment, the implementations and questionnaires completed by the subjects were analyzed. The master students compiled each program and executed it using the test cases sets generated. Thus, the correctness was evaluated and a score, from 0 to 10, was assigned for each program. To evaluate the hypotheses, statistical tests were performed, using analysis of variance (ANOVA) to cases that the sample was normal and Kruskal-Wallis tests when the sample was not normalized.

[1]http://www.icmc.usp.br/˜masbrit/TestCaseExperiment.rar

5 Results and Discussion

Data analysis was conducted considering the programs written by students and the answers of the questionnaires. Table 2 presents the population involved in the experiment, showing for each classroom the number of programs and the LOC, in average. In classroom I, the students received both specifications (vector and matrix), however, due to the short time (1 hour) available, only 42.86% of the students completed the two programs on time. Due to this, the matrix implementation was not considered in the analysis. In classrooms II and III the two specifications were implemented by the students. Comparing with Table 1, the implementations of the students have, in average, more lines of code than the reference programs.

Table 2: Information about the Experiment Population

Classroom	Program	Number of Programs	LOC (average)
I	vector manipulation	28	48.5
II	string and matrix manipulation	16	31/61
III	string and matrix manipulation	16	34/60

The students programs were analysed, observing their correctness and a score from 0 to 10 was assigned to each program (Table 3). As can be seen in the scores of Table 3, the students groups that reused the test case sets (TC) have obtained better scores than the students that did not reuse the test case sets.

Table 3: Correction of the programs

Classroom	Group	Score (in average)
I	Not using TC	4.39
	Using TC	8.14
II	Not using TC	6.75
	Using TC	7,69
III	Not using TC	4.75
	Using TC	6,5

The students reported some defects revealed by the test case sets. In some cases, the proposed problem was simplified and, hence, the solution did not match with the program specification. As an example, considers the following specification: *the program must read a set of n numbers and insert these numbers in a set X. Then, the program must print: 1) all the even numbers of X and their sum; and 2) all the prime numbers of X and their sum.* In some programs implemented by the students, the test to check if the number is even or prime is done when the number is read, without storing them in a set.

Table 4 presents the main defects found in the programs generated by the students. Most of them are related to defects in programming logic and in the design of the solutions.

Table 4: Mistakes made by students

Class	Group	Defects
I	Not using TC	Counter incorrect, incorrect calculation, incorrect use of variables, printing of incorrect outputs, incorrect loops
	Using TC	Printing of incorrect outputs, wrong output messages, incorrect calculation;
II	Not using TC	Dynamic allocation missing, incorrect calculation, infinite loop, wrong output messages, incorrect counter
	Using TC	Incorrect counter, dynamic allocation missing, incorrect calculation, wrong output messages
III	Not using TC	Dynamic allocation missing, wrong output messages, incorrect calculation, infinite loop and incorrect counter
	Using TC	Incorrect calculation, infinite loop, wrong output messages

Table 5 shows the data extracted from the questionnaires filled by students, which were also considered in the analysis of the results. For each classroom and group (not using TC and using TC) it is presented the quantity of males and females, the number of students from public and private school, and, finally, the number of students with previous experience in computer programming and students without previous experience. In the following, these results are statistically analyzed and discussed.

Table 5: Data extracted from questionnaires

Classroom	**Group**	**male/female**	**public/private school**	**experience/no experience in programming**
I	Not using TC	14/0	4/10	6/8
I	Using TC	13/1	3/11	7/7
II	Not using TC	8/0	1/7	3/5
II	Using TC	7/1	1/7	4/4
III	Not using TC	6/2	1/7	2/6
III	Using TC	7/1	6/2	4/4

During the data analysis, the influence of some factors in the programs quality was analysed. Examples of factors analysed are classroom and previous programming experience. We have conducted two analysis of variance, through two-factor models for unbalanced data. At first, it was studied the influence of the group and classroom factors. Then, it was studied the influence of scores and previous experience. In both cases, it was also checked if there was a significant relationship between these factors.

The models for the variance analysis are described in the following form:

$$\begin{aligned} Y_{ijk} &= \mu + \alpha_i + \beta_j + \gamma_{ij} + \varepsilon_{ijk} \\ &= \mu_{ij} + \varepsilon_{ijk} \end{aligned} \tag{1}$$

$$i = 0, 1; \qquad j = 1, 2, 3; \quad \text{or} \quad j = 0, 1; \qquad k = 1, 2, ..., n_{ij};$$

where,

- i is the index related to the group, where 0 indicates non-reuse of TC and 1 indicates reuse of TC;
- j is the index related to second specific factor of each analysis; in the first analysis, j is the classroom 1, 2 or 3. In the second analysis, j is the previous experience, equal 0 for students without previous experience or 1 for students with previous experience;
- n_{ij} is the number of replicas for the score, considering the factors i and j;
- k is the index related to number of replicas, with the value between 1 to n_{ij};
- Y_{ijk} is the value of score variable for i, j and k index;
- μ is the mean value of score;
- α_i is the effect of factor group on the mean;
- β_j is the effect of the second factor of the analysis (class or experience) on the mean;
- γ_{ij} is the effect of interaction between analysis factors;
- ε_{ijk} is the error attributed for the score variable considering the index i, j and k. These errors are normal, independent and identically distributed, with mean 0 and unknown variance σ^2;
- $\mu_{ij} = \mu + \alpha_i + \beta_j + \gamma_{ij}$, is the mean value of k replicas of the score variable, for the fixed index i and j;

It was assumed a confidence level of 95% (therefore rejecting the hypothesis H_0 for p-values $< 0,05$). In several studies, permutations were performed considering the variables studied to obtain evidence on the results of the study and the degree of suitability of each variable.

Before carrying out the hypothesis test, we have performed some descriptive analysis of the scores obtained by students considering the different levels of each factor. It is possible to observe that the mean score of the students belonging to group 1, which have reused test case sets, is higher than the mean score of the students in group 0 (Table 6 and Figure 1).

Table 6: Summary of the statistics considering score per group

Group	**Observations**	**Mean**	**Standard deviation**
0	30	5.12	2.54
1	30	7.33	2.71

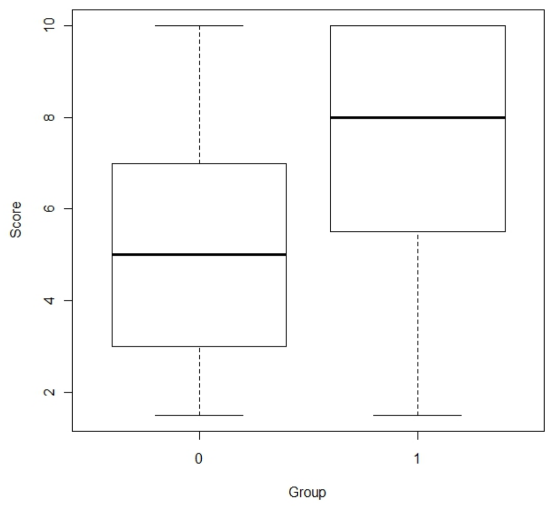

Figure 1: Box plot for the score per group

The analysis has shown that there is a difference between the mean scores obtained by students from different classrooms, mainly from classroom II to the others, as can be seen in Table 7 and Figure 3. However, the difference in the means was not meaningful in the variance analysis.

The results obtained through the analysis of variance for the group and classroom factors are shown in Table 8. It can be observed that the influence of the group factor in the mean score was significant, i.e., it can be stated that the reuse of test cases provided a significant increase in the students scores. The relationship between group and classroom factors was not considered significant.

Table 7: Summary of the statistics considering score per classroom

Classroom	**Observations**	**Mean**	**Standard deviation**
I	28	6.27	2.98
II	16	7.22	2.65
III	16	5.16	2.51

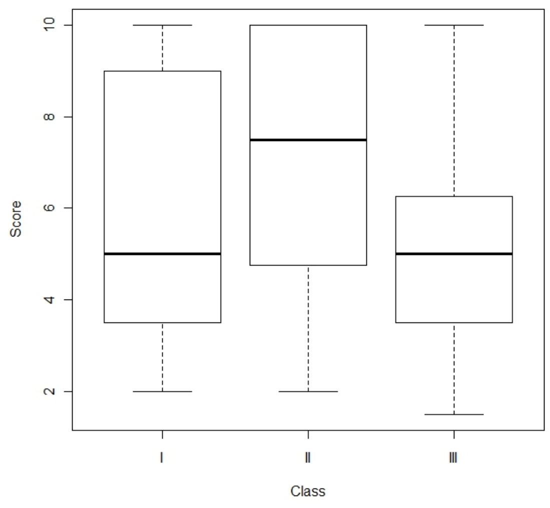

Figure 2: Box plot for the score per classroom

Table 8: ANOVA results for the score by source group, classrooms and the relationship between group and classroom

Source	**D. F.**	**Partial S.S.**	**Mean Square**	**F**	**P-value**
group	1	19.9783	19.9783	7.2606	0.0094
class	2	14.6466	7.3233	2.6615	0.0790
group x class	2	12.7942	6.3971	2.3249	0.1075
residual	54	148.5868	2.7516		
Total	59	196.0059	36.4503		

Based on the descriptive analysis of the scores according to the experience factor (Table 9 and Figure 2) we have evidence that the mean score may differ for the two levels of the factor. The second analysis of variance showed that the mean score was significantly different (Table 10). The hypothesis that previous programming experience does not produce an increase in score was rejected, i.e., students with previous experience obtained a highest score compared to those without previous experience. Considering the interaction between the factors, it was possible to conclude that their relationship is not significant. The influence of the group factor was also considered significant to the increase of score in this analysis.

Table 9: Summary of the statistics considering the score per experience

Experience	**Observations**	**Mean**	**Standard deviation**
0	34	5.57	2.76
1	26	8.00	2.75

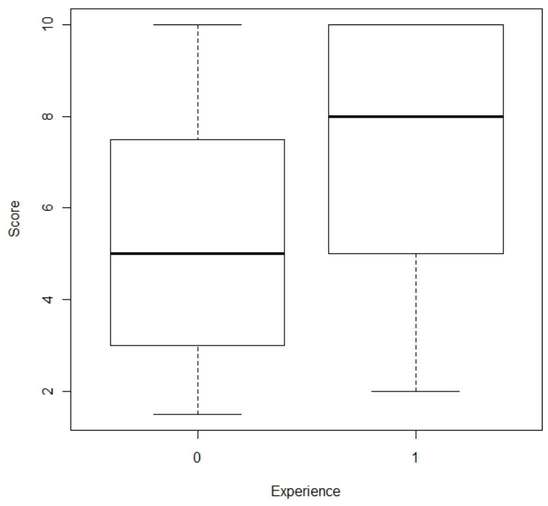

Figure 3: Box plot for the score per experience

Table 10: ANOVA results for the score by sources group, experience and the relationship between group and experience

Source	**D. F.**	**Partial S.S.**	**Mean Square**	**F**	**P-value**
group	1	28,1550	28,1550	9,6025	0,00304
experience	1	11.7956	11.7956	4.0230	0.04972
group x exp.	1	0.0167	0.0167	0.0057	0.94014
residual	56	164.1952	2.9321		
Total	59	204.1625	42.8994		

In addition, an analysis was performed to verify if the use of test cases has significant influence on the scores of the subjects, considering the group without previous experience in programming. This analysis was conducted using the Kruskal-Wallis test, because it was not possible to obtain normality in the data by transformations. It was observed that the reuse of test cases provided a significant increase in scores of the subjects without previous experience ($p - value = 0.0006$). Therefore, the main objective of our approach has been reached, i.e., our approach contributes to leverage the programming knowledge of unexperienced students.

After finishing the analysis of tests based on groups, it was obtained a $p - value < 0.05$ (Table 8 and Table 10), allowing us to reject the null hypothesis (formulated at Section 4) and to conclude about the existence of significant differences between groups. Thus, there is evidence that the reuse of test cases generated based on reference programs can contribute to improve the quality of equivalent programs implemented by students in introductory programming courses. As expected, we have also observed that the previous experience in computer programming can contribute to improve the programs quality. Other tests were performed considering the sex and background variables, but we have not found evidence that these variables can contribute to test the hypothesis for the sample considered.

6 Threats to Validity

Internal validity threats concern external factors that may affect an independent variable. Appropriate analysis - i.e., ANOVA and Kruskal-Wallis - were performed to analyse the effect of these factors. However, it was not possible to say that the students' experience, for example, properly discerns the effect of results.

Another important threat is the possibility of fraud, where identical implementations could be delivered by undergraduate students. Precautions were taken during the execution of the experiment so that subjects did not communicate while implementing the programs. In addition, the codes were examined individually and no evidence of identical programs was found.

Another threat is the simplicity of the tested programs, as it can be assumed that any test case sets would be sufficient to achieve a satisfactory level of correctness for the programs. Treatment was not found for this threat, since the level of the students did not allow very complex programs and the time available for implementation was also reduced.

Construct validity threats concern the relationship between theory and observation. In particular, this threat is related to how the students carried out the tests. To avoid any subjective evaluation, in the moment of the experiment the master student was present and answered the subjects' doubts during the experiment. The great majority of subjects had no practice in software testing, so it avoided that a subject could favor some technique. In the classroom where the study was applied twice, the subjects and groups were interspersed, i.e. students who used test case sets in first study did not used them in the second study.

We did not provide treatment to the threat in which people try to look better when being observed, because it would require isolating each student, with only the presence of the instructor, but this is not feasible for this kind of study.

The conclusion validity is concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the results of an experiment. In this case the quality that is being measured is subjetive and contains human judgement, so the quality was measured by the scores of the subjects' programs. The master student who conducted the experiment analysed the programs and assigned them a score. To improve this reliability, she has used a criteria list to give scores to the subjects programs. Attention was paid to not violating assumptions made by statistical tests. In situations that allowed to use of parametric test, ANOVA was used, and in situations that did not allow it, the Kruskal-Wallis test was used. The size of subjects set was selected based on classes where it was possible to carry out the study, this is a threat, but this set must be large and more representative to generalization of the study.

The external validity of the experiment refers to the ability to generalize the results. Since the programs studied were in the specific domain of vectors and matrices manipulation, the results cannot be applied to other areas of programming. The generalization can be achieved if the experiment is replicated in other classrooms for other levels of undergraduate students and other domains of programs.

7 Proposed Data Repository

Motivated by the results of the experimental study described above, in this section we propose the implementation of a repository to store the reference programs and the associated test case sets, so that they can be easily retrieved to be used during computer programming classes.

Repositories are information systems containing features that allow to import, store, preserve, extract, and export digital objects. The current increased of use of repositories has been possible due to the reduction of physical storage prices, the use of standards such as metadata protocol of the Open Archives Initiative (OAI-PMH), and advances in the development of metadata standards that support the open communication model. The term *learning object repository* comprises a database in which learning objects are stored to facilitate their search and reuse. Learning objects are any digital resources that can be reused to assist with learning and distributed network on demand [24].

The *DSpace Institutional Digital Repository System* is a system of electronic publications management developed in a collaboration between MIT (Massachusetts Institute of Technology) and HP (Hewlett-Packard Co.). The DSpace has BSD Open Source license and is available since November 2002. It is currently the most used software for building institutional repositories [25]. With this software, the user can manage the technical and scientific information, ensuring a permanent access to this information [26].

Considering the results of the experimental study described in this paper, we propose a repository that will be implemented with the instantiation of DSpace. The objective is to enable reuse of teaching materials generated, such as: sets of reference programs and its associated test case sets, helping to optimize the teaching process.

Two phases are necessary to implement the repository: 1) exploratory and descriptive theoretical basis for the use of DSpace; and 2) experimental research consisting on evaluation of the software configured. These two phases can be applied in the steps: installation and configuration of the tool, metadata customization, maintenance of the external digital object, operation of the data provided, study of the license distribution model. In this direction, we are building a repository that can be used to store programs and test case sets, while they are being generated, aiming at reuse and improvement of the materials quality. For this proposal, we require a special feature, which is a free access provided only to teachers previously registered on the system, who can download the material available in the repository.

A schema for the proposed repository is shown in Figure 4. The complete deployment and configuration of DSpace is planned as a future work. The repository is configured to store test case sets and the associated reference programs, considering topics taught in introductory programming disciplines. The repository is organized to enable user interaction, so that it does not act merely as a storage compartment; users can add new learning objects and, for each download done, they should provide feedback about the use of the downloaded object. The teacher can look for materials in the repository using a sort key, as for example the name of the program. The information initially available in the repository are related to the discipline of Introduction to Computer Science, such as: program structures: decision and repetition statements, primitive data type, modularization of programs, procedures, functions and parameter passing; compound data types: vectors, matrices, strings, records, dynamic structures (pointers and files). When the user selects a particular topic, the system will retrieve all the programs registered in the repository that are related to the chosen topic. Selecting a program, the user obtains the following information: program specification, programming language, courses where it can be used, level of difficulty and estimated time for implementing the exercise. Information about the test case sets related to the program selected are also avaliable, such as: testing criteria used, number of test cases, the test cases (input and output data) and coverage obtained.

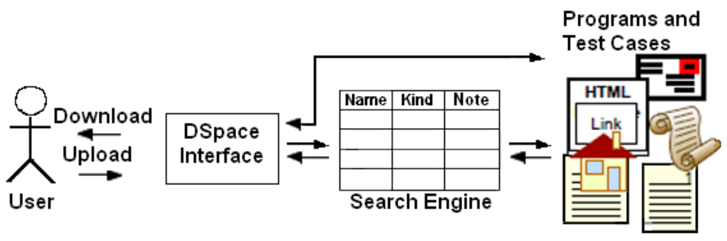

Figure 4: Schema of the proposed repository. Source: Adapted from [27]

8 Concluding Remarks

This paper presented an experimental study to evaluate the reuse of test case sets to help the teaching of introductory programming courses. The objective is improving the quality of the programs generated by the students. It is not expected that every aspect of the student's program be tested, but that test cases are able to detect typical errors made by new students in computer programming and, thus may contribute to their learning.

The development of the proposal followed a methodology based on experimentation, according to Wohlin [22]. The conducted experimental study aimed to validate this proposal. The ANOVA test provided evidence that sets of functional test cases reused can help the student in obtaining an improvement in the quality of the programs. It is expected that the experiment has motivated the participating students to use testing activities during the program development and, as a consequence, that they will give a greater importance to this activity. It was also analysed the effect of previous experience factor, which indicated evidence of positive influence of the reuse of test cases by subjects with no previous experience in programming. This is the biggest benefit of the approach, because it shows that newbie students can develop better quality programs.

One limitation of our proposed approach is that the test cases are generated considering reference programs and they are reused by the students to test equivalent programs. Thus, it is not expected that all aspects (for instance, all statements) of the student's program are tested, but that the test case sets are able to validate only functional aspects of the programs.

As future work we intend to conduct the same experiment using other courses and periods of the computer science courses and thereby obtain more general results about the hypotheses studied. Other studies can be developed considering covariates (extra information) possibly explaining the relationship between the results of the statistical tests.

Acknowledgment

The authors would like to thank the Brazilian funding agencies CAPES and FAPESP, for the financial support, under CAPES process DS-7109935/M and FAPESP processes: 2009/04517-2.

References

[1] E. W. Dijkstra, "Structured programming," O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds. London, UK, UK: Academic Press Ltd., 1972, ch. Chapter I: Notes on structured programming, pp. 1–82.

[2] E. J. Weyuker, "Axiomatizing software test data adequacy," *IEEE Transactions on Software Engineering*, vol. 12, no. 12, pp. 1128–1138, 1986.

[3] T. Shepard, M. Lamb, and D. Kelly, "More testing should be taught," *Commun. ACM*, vol. 44, pp. 103–108, June 2001.

[4] Y. D., Y. W., M. Lau, and S. yu L., "Experiments on test case reuse of test coverage criteria," in *Ubiquitous Intelligence Computing and 7th International Conference on Autonomic Trusted Computing (UIC/ATC), 7th International Conference on*, 2010, pp. 277–281.

[5] L. Lima, J. Iyoda, A. Sampaio, and E. Aranha, "Test case prioritization based on data reuse an experimental study," in *3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09, 2009, pp. 279–290.

[6] M. Mirzaaghaei, "Automatic test suite evolution," in *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ser. ESEC/FSE '11, 2011, pp. 396–399.

[7] C. Schulte, T. Clear, A. Taherkhani, T. Busjahn, and J. H. Paterson, "An introduction to program comprehension for computer science educators," in *2010 ITiCSE working group reports on Working group reports*, ser. ITiCSE-WGR '10, 2010, pp. 65–86.

[8] E. L. Jones, "Software testing in the computer science curriculum – a holistic approach," in *Australasian conference on Computing education*, ser. ACSE '00, 2000, pp. 153–157.

[9] ——, "Grading student programs - a software testing approach," in *fourteenth annual consortium on Small Colleges Southeastern conference*, ser. CCSC '00, 2000, pp. 185–192.

[10] ——, "Integrating testing into the curriculum – arsenic in small doses," in *Thirty-second SIGCSE technical symposium on Computer Science Education*, ser. SIGCSE '01, 2001, pp. 337–341.

[11] E. Jones, "An experiential approach to incorporating software testing into the computer science curriculum," in *Frontiers in Education Conference, 2001. 31st Annual*, vol. 2, 2001.

[12] S. H. Edwards, "Teaching software testing: Automatic grading meets test-first coding," in *Addendum to the 2003 Proceedings of the Conference on Object-oriented Programming, Systems, Languages, and Applications*, 2003, pp. 318–319.

[13] A. Tinkham and C. Kaner, "Experiences teaching a course in programmer testing," in *Agile Development Conference*, 2005, pp. 298–305.

[14] D. M. Souza and E. F. Barbosa, "Progtest: An environment for the submission and automatic evaluation of programming assignments based on testing activities," in *Simpósio Internacional de Iniciação Científica da USP*, 2009, (in Portuguese).

[15] S. H. Edwards and M. A. Perez-Quinones, "Web-cat: automatically grading programming assignments," in *13th annual conference on Innovation and technology in computer science education*, ser. ITiCSE '08, 2008, pp. 328–328.

[16] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez, "Experiences with marmoset: designing and using an advanced submission and testing system for programming courses," in *11th annual SIGCSE conference on Innovation and technology in computer science education*, ser. ITICSE '06, 2006, pp. 13–17.

[17] D. N. Campanha, S. R. S. Souza, O. A. Lemos, E. F. Barbosa, and J. C. Maldonado, "Reuse of test case sets: A study of domain of algorithms sorting," in *VI Experimental Software Engineering Latin American Workshop*, 2009, pp. 114–123.

[18] G. J. Myers, C. Sandler, T. Badgett, and T. M. Thomas, *The art of software testing.* John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

[19] M. E. Delamaro, J. C. Maldonado, and M. Jino, *Introduction to software testing*, ser. Campus. Elsevier, 2007, vol. 394, (in Portuguese).

[20] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, 1997.

[21] R. S. Pressman, *Software Engineering*, 5th ed. McGraw-Hill, 2002.

[22] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction.* Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[23] A. F. G. Ascencio and E. A. V. Campos, *Fundamentals of computer programming*, 2nd ed. Prentice Hall, 2007, (in Portuguese).

[24] D. A. Wiley, "Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy," *The Instructional Use of Learning Objects Online Version*, vol. 2830, no. 435, pp. 1–35, 2000.

[25] C. L. M. Viana, M. A. M. A, and M. Shintaku, "Institutional repositories in science and technology: an experience of customizing of the dspace," in *Simpósio de Bibliotecas Digitais*, São Paulo, SP, Brasil, 2006, (in Portuguese).

[26] M. A. A. Mardero, "Dspace repositorie and digital preservation," in *Encontro de Informação em Ciências da Comunicação*, 2004, p. 14, (in Portuguese).

[27] G. R. B. Fachin, J. Stumm, R. L. Comarella, F. A. P. Fialho, and N. Santos, "Knowledge management and cognitive view of institutional repositories," *Perspect. ciênc. inf.*, vol. 14, no. 2, pp. 220–236, 2009, (in Portuguese).