Integrated Teaching of Programming Foundations and Software Testing

Ellen F. Barbosa, Marco A. G. Silva, Camila K. D. Corte, and José C. Maldonado
University of São Paulo (ICMC/USP), São Carlos/SP, Brazil 13560-970
{francine, magsilva, camila, jcmaldon}@icmc.usp.br

Abstract – **The importance of software testing is widely recognized, but usually only a small portion of the Computer Science (CS) curriculum is allocated for teaching it. Some experiences have suggested that the teaching of software testing should begin as early as possible so an adequate culture of testing could be created. One way to achieve this is addressing testing practices in conjunction with programming concepts in introductory CS courses. In this paper we explore such idea, working on the integration between the teaching of software testing along with the teaching of programming foundations. We discuss the development of an educational module, and its related learning materials, for integrating such knowledge domains. Besides that, we propose PROGTEST – a Web-based environment for the submission and automatic evaluation of practical programming assignments based on testing activities, aiming at providing an adequate feedback to evaluate the learners' performance concerning programming and testing.**

Index Terms – Educational modules, Programming foundations, Software testing, Supporting tools.

INTRODUCTION

Software testing is the process of executing a program with the intent of finding errors [1]. In the context of Software Engineering, testing is one of the most important activities to guarantee the quality and the reliability of the software under development but, at the same time, it is a difficult topic to learn or teach without the appropriate support.

Traditionally, testing has been taught at the very end of the Computer Science (CS) learning process. Besides that, usually only a small portion of the CS curriculum is allocated for testing in comparison to other activities of the software development process [2]. On the other hand, recent experiences have suggested that the testing activity could be taught as soon as possible in the learning process. Earlier mastering of testing concepts and techniques would: (1) improve the reasoning about the program (and its solution), leading to better high quality products; and (2) induce and facilitate the use of testing throughout the software development process, leading to a better high quality process, in contrast to the current practices.

One of the initiatives which has been investigated refers to the introduction of testing concepts in conjunction with programming foundations in introductory CS courses [3]-[6]. Programming foundations is not an easy subject to be taught – many students have difficulties understanding the abstract concepts of programming [7] and have a wrong view about the programming activity [6]. Thus, the main challenge in introductory CS courses is to make them interesting and relevant for learners [8]. Since testing requires the learners to know the behavior of their programs, such activity could be explored to help them understand the abstract concepts of programming and develop the expected skills [6]. Furthermore, since testing forces the integration and the application of theories and skills of software analysis, project and implementation, learners who start testing earlier could become better testers and developers as well [4][5][9]. In fact, the idea is to create an adequate "culture of software testing" among learners (and developers).

We intend to work on the integrated teaching of software testing and programming foundations in introductory CS courses. Basically, we have investigated two mechanisms to support our ideas. The first one refers to the development of an educational module for teaching testing and programming concepts in a simultaneous way. We chose to explore the Object-Oriented (OO) paradigm and the Java language, following the current tendencies of CS curriculum for teaching programming concepts [10]. We are based on a standard process for constructing educational modules and on a set of models for structuring the related educational content, both proposed in Barbosa's work [11]-[13]. The second one refers to the development of PROGTEST – a Web-based environment for the submission and automatic evaluation of practical programming assignments based on testing activities. In short, such environment can provide an appropriate feedback to evaluate the learners' performance concerning programming and testing.

The remainder of this paper is organized as follows. Section 2 discusses some of the main issues on teaching programming foundations and software testing. In Section 3 we provide a brief overview on the related work regarding the development of educational modules. The construction of an educational module for programming and testing is presented in Section 4. In particular, we focus on the content modeling activity for developing the integrated educational content. Section 5 discusses the main aspects of PROGTEST. Section 6 presents our conclusions and further work.

SOME ISSUES ON TEACHING PROGRAMMING AND TESTING

As said before, the main challenge in introductory CS courses is to make them interesting and relevant for learners [8]. In general, the traditional approach in introductory CS courses is

to provide an overview on Computer Science and then start teaching the programming foundations by using some specific language (such as Pascal, C or Java) [8]. Most of the time, however, the emphasis is in the syntax of the language instead of solving the problem through the development of algorithms. As a consequence, students learn how to program through a trial-and-error practice, without developing the adequate comprehension and analysis skills [6].

Over the last years, many universities have changed their curricula in order to introduce the OO paradigm as a way for better motivating learners on basic programming concepts. However, such initiative is not sufficient to solve the learners' problems concerning programming. They still have difficulties understanding how to design a program to solve a certain task, dividing functionality into procedures and finding bugs in their own programs [7]. Furthermore, the learners have wrong views about the programming activity, such as [6]: (1) once the compiler accepts the code without complaining, all errors have been removed; (2) once the code produces the output expected on a test value or two, it will work well all the time; (3) the code looks "correct" for the students; (4) once the code gives the correct answer for the instructor's sample data, it is finished.

Edwards [6] highlights the application of software testing in conjunction with programming foundations in introductory CS disciplines can make the learners more careful with respect to the development and understanding of algorithms [6]. However, the teaching of testing in introductory CS courses is not a trivial task. Several problems regarding to this subject can be pointed out [5][6]:

- Software testing requires the learners have experience at programming.
- Instructors have to evaluate the program correctness manually. It may not be feasible to evaluate the test cases too.
- Learners need constant and concrete feedback on how to improve their performance on testing at many points throughout the development of a solution rather than just once at the end of an assignment.
- Learners see testing as a boring activity, where much time is spent on performing the tests and the writing of test plans creates a large overhead in the workload.

Despite such limitations, Barriocanal [3] has shown that the teaching of testing earlier can improve the quality of the code implemented and can ease the learning process, both of testing and of programming. Barriocanal has also investigated if the learners are keen on to perform tests in their programs. The majority found the idea valid – although recognizing that testing is a little boring activity, they agreed that it brings benefits, both to the improvement of the quality of the programs constructed as well as to the assimilation of the programming concepts taught in the course.

EDUCATIONAL MODULES DEVELOPMENT: AN OVERVIEW

Educational modules are concise units of study, composed by theoretical and practical content which can be delivered to learners by using technological and computational resources [11]-[13]. For theoretical content, instructors use books, papers, web information, slides, class annotations, audio, video, and so on. Practical content is the instructional activities and associated evaluations, as well as their resulting artifacts (e.g., executable programs, experimental studies, collaborative discussions). Theoretical and practical content are integrated in terms of learning materials. Learning environments, presentation tools and mechanisms to capture classroom lectures and to support discussion spaces and collaborative work are examples of the required infrastructure for delivering the learning materials.

Similar to software products, educational modules require the establishment and integration of methods, tools and procedures into systematic processes aiming at producing reliable, evolvable and quality products. Additionally, some specific aspects must also be considered, such as the subject knowledge domain, the learner's profile, the course objectives, the pedagogical strategies to be adopted, among others.

In this perspective, in a previous work, Barbosa *et al.* have investigated and defined some supporting mechanisms for developing educational modules [11]-[13]. In particular, a Standard Process for Educational Modules and an integrated modeling approach for educational content (*IMA-CID – Integrated Modeling Approach: Conceptual, Instructional and Didactic*) were established.

The Standard Process for Educational Modules is based on the International Standard ISO/IEC 12207, tailored to the context of educational modules by including aspects of content modeling, practices from instructional design, and issues of distributed and cooperative work [11]. The *IMA-CID* approach is composed by a set of models, each one addressing specific issues in the development of educational modules [12]:

- The *Conceptual Model* corresponds to a high-level description of the knowledge domain, representing the main concepts and the relationships among them. According to *IMA-CID*, the construction of a conceptual model focuses on the ideas and rules of Conceptual Mapping [14].
- The *Instructional Model* characterizes what kind of additional information (e.g., facts, principles, procedures, examples, and exercises) can be incorporated to the educational content, relating them to the concepts previously identified. As a support to construct the instructional model, *IMA-CID* adopts the HMBS (*Hypertext Model Based on Statecharts*) model [15], focusing on the mechanisms for hierarchical decomposition it provides. The extended version of HMBS, applied to the instructional level of *IMA-CID*, is named *HMBS/Instructional*.
- The *Didactic Model* is responsible for the establishment of prerequisites and sequences of presentation among the objects previously characterized in the conceptual and instructional models. HMBS is also adopted in the didactic level of *IMA-CID – HMBS/Didactic*. As an extension to the model, we have also introduced the idea of an *open specification*, which provides support for the

definition of dynamic contexts of learning. Depending on aspects such as audience, learning goals and course length, distinct ways for presenting and navigating through the same content can be required. An open specification allows representing all sequences of presentation in the same didactic model. So, from a single model, several versions of the same content can be generated according to different pedagogical aspects. Moreover, when an educational module is implemented based on an open specification (*open implementation*), its navigation paths can be defined by the user, in "execution time", based on the learner's understanding and feedback, for instance.

An educational module for programming and software testing has been developed using the supporting mechanisms defined in Barbosa's work [11]-[13]. Next, we discuss the main characteristics of the module, focusing on the modeling aspects of its educational content.

AN EDUCATIONAL MODULE FOR PROGRAMMING FOUNDATIONS AND SOFTWARE TESTING

One of the mechanisms we have worked on to promote the integrated teaching of software testing and programming foundations refers to the development of an educational module (and its related educational content) on such subjects. The idea is to gradually introduce the testing fundamentals while the programming concepts are being taught to the learners. Basically, we have investigated the main OO concepts taught in introductory CS courses, defining which testing concepts can be taught in conjunction with them.

The module is composed in terms of two submodules – one for programming foundations and other for software testing. Each submodule is implemented as a set of slides, to which HTML pages, text documents, learning environments and supporting tools have been integrated. In the case of the Software Testing submodule, for instance, a coverage testing tool – JaBUTi (*Java Bytecode Understanding and Testing*) [16], which supports coverage analysis to test Java programs and Java components – has been incorporated to foster the practical application of testing into OO programs.

The submodules have been integrated to each other by means of specific links, which represent the integration points between programming and testing. Figure 1 illustrates part of such integration. Starting from the main slide of the module, the user can navigate through the structure of both the Programming Foundations submodule and the Software Testing submodule. While the specific programming concepts are presented, links to their corresponding parts in the Testing submodule are enabled. The same occurs for testing concepts with respect to the Programming submodule. For instance, Figure 1(a) corresponds to the main slide of the module; links for Programming and for Testing submodules are activated. Consider we choose to navigate through the Programming submodule, more specifically looking for information related to control-flow statements (Figure 1(b)). From this point, we are able to explore the concepts related to structural testing, in particular the All-Edges control-flow criterion (Figure 1(c)). Additionally, we can reach the JaBUTi testing tool, which allows to explore in the practice the concepts learned so far (Figure 1(d)).

The module has been developed according to the Standard Process for Educational Modules [11], briefly discussed in the previous section. Its educational content has been modeled with basis on the set of models of the *IMA-CID* approach [12]. Figure 2 shows the integrated conceptual model for programming and testing. Such model was constructed based on two other conceptual models – one dealing specifically with the main concepts of OO programming and other one dealing with the testing concepts.

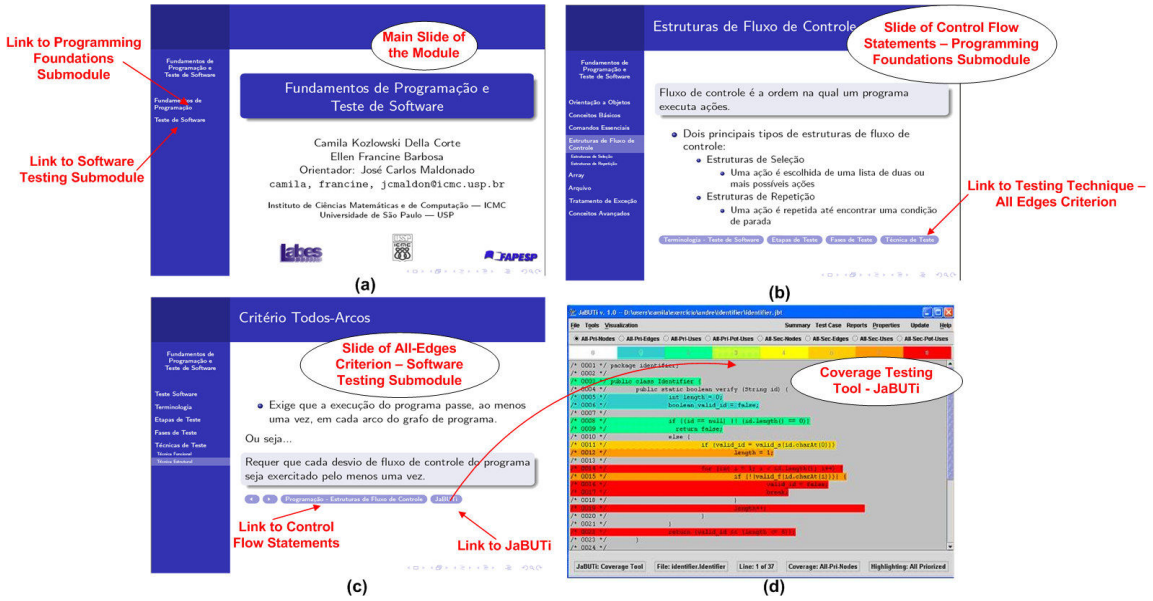

FIGURE 1
SCENARIO OF NAVIGATION THROUGH PROGRAMMING AND TESTING SUBMODULES.

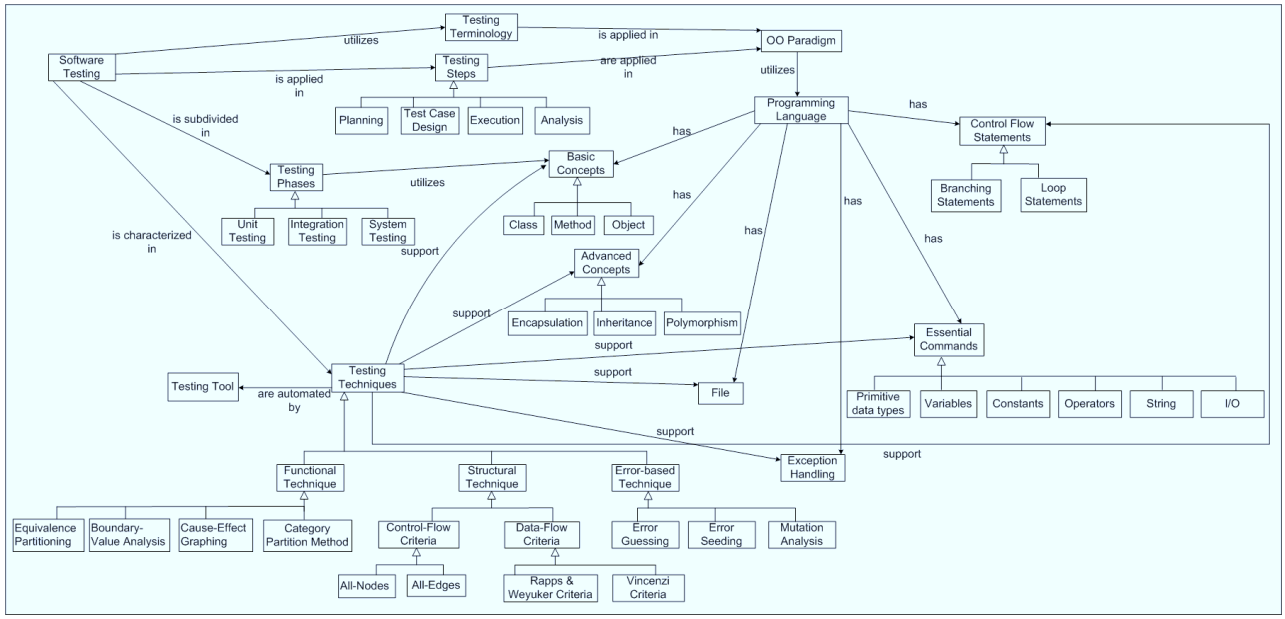

FIGURE 2
INTEGRATED CONCEPTUAL MODEL FOR PROGRAMMING AND TESTING.

Considering the OO programming, we established that Programming Language concepts could be divided into the following categories: (1) Basic Concepts, which deals with objects, methods and classes; (2) Essential Commands, which refers to primitive data types, variables, constants, operators, strings and I/O commands; (3) Control Flow Statements, composed of branching and loop statements; (4) Files; (5) Exception Handling; and (6) Advanced Concepts, which deals with encapsulation, inheritance and polymorphism.

Regarding software testing, we divided its main concepts in the following classes: (1) Testing Terminology, which deals with basic concepts, such as taxonomy of errors, test cases, test suite, adequacy and selection criteria; (2) Testing Steps, addressing the required steps to perform a testing activity (planning, test case design, execution and analysis); (3) Testing Phases, characterized into unit, integration and system testing; (4) Testing Techniques, which refers to the functional, structural and error-based techniques and criteria; and (5) Testing Tools, which refers to the supporting mechanisms applied to automate the testing activity.

After defining both models separately, we established the "connection points" where their concepts could be integrated. For instance, while addressing the basics of the OO Paradigm, the main ideas of a testing activity, covered by Testing Terminology and Testing Steps, can be introduced. The Basic Concepts of OO can be taught in conjunction with the Testing Phases. The relevant point here refers to the discussion about which elements of OO should be taken as the smallest unit of an OO program (method or class), determining which types of OO testing (intra-method, inter-method, intra-class and inter-class) characterize each testing phase.

The Essential Concepts of an OO programming language can be introduced in conjunction with the general concepts of Testing Techniques. At this stage, the emphasis can be on the functional (e.g., Equivalence Partitioning) and on the structural criteria, in particular the All-Nodes control-flow criterion [1]. After introducing the Control Flow Statements, the testing techniques must be reinforced and the All-Edges control-flow criterion [1] can be addressed. The concepts related to specific types of Variables (such as arrays) and Files should be taught in conjunction with the data-flow testing criteria, emphasizing the Rapps & Weyuker criteria (e.g., All-Definitions, All-Uses) [17]. By introducing Exception Handling and Advanced Concepts, the data-flow criteria must be reinforced and deepened. The set of structural criteria defined by Vincenzi (e.g., All-Nodes-Exception-Dependent, All-Nodes-Exception-Independent) [16] as well as the error-based criteria (e.g., Mutation Analysis [18]) are examples that can be explored in advanced stages of the learning process of programming foundations.

After having developed the integrated conceptual model for programming and testing, we defined the integrated instructional and didactic models for such domains, i.e., the *HMBS/Instructional* and the *HMBS/Didactic* models. For the sake of space, these models are not illustrated here. In the instructional model, we focused on representing the integration of supporting tools to the module. For instance, JUnit [19] – a framework that supports the creation and execution of test cases and test suites – was considered to be introduced even at the initial stages of the module. Coverage testing tools, such as JaBUTi [16], were also considered in order to provide mechanisms to exercise and explore specific theories and skills, motivating the students to put in practice the concepts they have learned. Such tools can be introduced at intermediary and advanced stages. Figure 1 illustrates part of the integration among programming concepts (control flow statements), testing concepts (All-Edges criterion) and tools application (JaBUTi) we have explored in the module.

In the didactic model, we established: (1) the sequences of presentation for each submodule; and (2) the sequences of presentation among them. At any time we can stop exploring the Testing submodule and return to the programming concepts. The same is valid for the Programming submodule in relation to the testing basics. Thus, the free navigation from the Programming submodule to the Testing submodule (and vice versa) gives flexibility for the instructor to adequate the educational content according to the learners' performance in "execution time". Such flexibility is due to the fact that the module is implemented according to an open specification (explored in the didactic model), which addresses all the possible presentation sequences in the same model.

PROGTEST: AN ENVIRONMENT FOR THE SUBMISSION AND EVALUATION OF PRACTICAL ASSIGNMENTS

A critical issue for the success of the integrated teaching of programming foundations and software testing is how to provide an appropriate feedback and to evaluate the learner's performance. In this learning scenario, the instructors' workload is doubled since both the test cases and the code must be evaluated. An alternative aiming at reducing such workload is the use of an automated environment to evaluate practical assignments [6]. Moreover, the use of such environment can bring additional benefits in terms of consistence, efficacy and efficiency. Submitted programs are analyzed as a whole in the same level of efficiency and the results of the evaluation are based on the same standards. After the evaluation, the environment can also generate reports, so each learner is informed of his/her own performance and can be compared with the average and most productive ones.

In this perspective, we are developing an environment for the submission and automatic evaluation of practical programming assignments based on testing activities – PROGTEST. The idea is to provide an automated support to evaluate the programs and the test cases submitted by the learners. Actually, both the code quality and the testing activity can be analyzed based on the testing criteria adopted. Coverage testing tools should be integrated to the environment as a support to apply the testing criteria and to evaluate the coverage of the test set, obtained from the programs' execution. For the integrated teaching of programming foundations and software testing, we are using JaBUTi [16] as the coverage testing tool to be integrated to PROGTEST.

Figure 3 shows the main features of PROGTEST. Given a program P_{St_i} (provided by the students) and its respective test case set T_{St_i} (produced based on a criterion C_K previously established – T_{St_i} is C_K–adequate), the environment, integrated to a coverage testing tool, must be able to:

1. Execute the program P_{St_i} against the test case set T_{Inst} (provided by the instructor);
2. Utilize the test cases set T_{St_i} to test the "oracle program" P_{Inst} (provided by the instructor);
3. Compare the behavior of P_{Inst}, executed against the test cases set T_{Inst}, to the behavior of P_{St_i}, executed against the test cases set T_{Inst}; and
4. Compare the behavior of P_{Inst}, executed against the test cases set T_{St_i}, to the behavior of P_{St_i}, executed against the test cases set T_{St_i}.

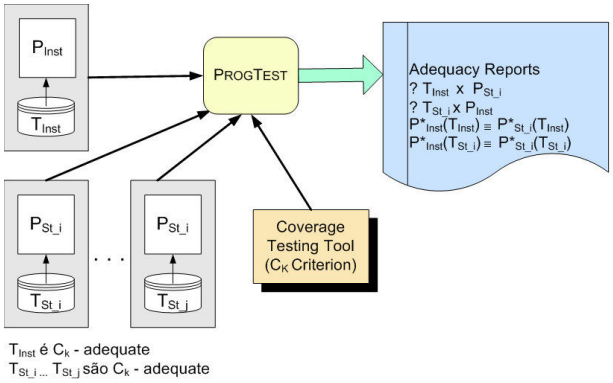

FIGURE 3
PROGTEST: MAIN FEATURES.

By performing such executions, PROGTEST provides the code coverage adequacy analysis of the test cases used. Both functional (by using JUnit) and structural testing are considered. Based on the results obtained, PROGTEST is able to accept or reject the program and the test cases set provided by the learners as well as to suggest a grade to the assignment. It is important to highlight that right after having submitted his/her assignment, the learner can visualize its evaluation report. Figure 4 illustrates the evaluation reports provided by PROGTEST[1].

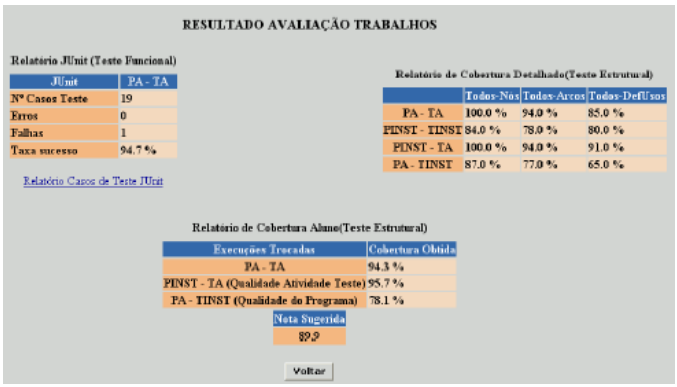

FIGURE 4
PROGTEST: EVALUATION REPORTS.

PROGTEST is a Web Java application, developed as an open source initiative. Three modules are responsible for implementing the main functionalities of the environment: (1) course management module, which deals with the management of users (learners and instructors), classes and assignments; (2) submission module, which deals with the management of the learners' assignments and the instructors' oracle programs; and (3) evaluation module, which deals with the management and the execution of each evaluation.

As a Web application, PROGTEST follows the *Model-View-Controller* (MVC) architecture. The SUN's JSF

[1] PROGTEST was originally developed in Portuguese. An English version of the environment is under development.

framework handles the *View* and *Controller* components. The *Model* is implemented as plain Java classes and the data is persisted using the DAO pattern, handled by JDBC statements, and direct access to XML documents.

PROGTEST is under development and we are now defining systematic and controlled experiments to validate it into the context of the integrated teaching of programming and testing. Such experiments have already been planned for the next term, involving different courses offered to graduate and undergraduate students at ICMC/USP. Furthermore, both learners and instructors' attitudes toward the educational module produced should also be evaluated.

CONCLUSIONS AND FURTHER WORK

In this paper we investigated some supporting mechanisms to integrate the teaching of software testing along with the teaching of programming foundations. From the point of view of programming, the testing activity can contribute to enhance the learners' capabilities of understanding and analysis. From the point of view of software testing, a testing culture can be created earlier, so it may become a common practice among developers, motivating them to apply it from the very beginning of software development.

In this perspective, we discussed the development of an educational module for integrating programming and testing, focusing on the modeling activity of its related educational content. Also, we proposed PROGTEST – an environment for submission and automatic evaluation of practical assignments, whose main objective is to provide an adequate feedback to evaluate the learners' performance concerning programming and testing.

As a further work, we intend to apply the educational module, in conjunction with the PROGTEST environment, in introductory CS courses for undergraduate students. The goal is to evaluate the practical use of our mechanisms and ideas in real projects and learning scenarios. Further studies have also been planned in order to investigate the use of conceptual models in the development of domain ontologies and vice versa. In this sense, the conceptual models for programming foundations and software testing can be explored in the development of an integrated ontology for these domains.

ACKNOWLEDGMENT

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES and CNPq) and the QualiPSo Project (IST- FP6-IP-034763) for their financial support.

REFERENCES

[1] G. J. Myers, Corey Sandler, Tom Badgett, and Todd M. Thomas. *The Art of Software Testing*. John Wiley & Sons, 2nd. edition, 2004.

[2] T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Communications of the ACM*, 44(06):103–108, 2001.

[3] E. G. Barriocanal, M. A. S. Urbán, I. A. Cuevas, and P. D. Pérez. An experience in integrating automated unit testing practices in an introductory programming course. In *ACM SIGCSE Bulletin*, volume 34, pages 125–128, 2002.

[4] E. F. Barbosa, J. C. Maldonado, R. Leblanc, and M. Guzdial. Introducing testing practices into objects and design course. In *16th Conference on Software Engineering Education and Training (CSEE&T'03)*, pages 279–286, Madrid, Spain, 2003.

[5] A. Patterson, M. Kölling, and J. Rosenberg. Introducing unit testing with BlueJ. In *8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'03)*, Thessaloniki, Greece, 2003.

[6] S. H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *35th SIGCSE Technical Symposium on Computer Science Education*, pages 26–30, Norfolk, Virginia, USA, 2004.

[7] E. Lahtinen, K. Ala-Mutka, and H. Järvinen. A study of the difficulties of novice programmers. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 14–18, New York, NY, USA, 2005. ACM Press.

[8] T. J. Hickey. Scheme-based web programming as a basis for a CS0 curriculum. In *35th SIGCSE Technical Symposium on Computer Science Education*, pages 353–357, Norfolk, Virginia, USA, 2004.

[9] E. L. Jones. An experimental approach to incorporating software testing into the computer science curriculum. In *31st ASEE/IEEE Frontiers in Education Conference*, pages 7–11, Reno, Nevada, 2001.

[10] S. Cooper, W. Dann, and R. Pausch. Teaching objects-first in introductory computer science. In *34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'03)*, volume 35, pages 191–195, Reno, Navada, USA, 2003.

[11] E. F. Barbosa, and J. C. Maldonado. An integrated content modeling approach for educational modules. In *IFIP 19th World Computer Congress – International Conference on Education for the 21st Century*, pages 17–26, Santiago, Chile, August 2006.

[12] E. F. Barbosa, and J. C. Maldonado. Towards the establishment of a standard process for developing educational modules. In *36th Annual Frontiers in Education Conference (FIE'06)*, San Diego, CA, October 2006. CD-ROM.

[13] E. F. Barbosa, S. R. S. Souza, and J. C. Maldonado. An Experience on Applying Learning Mechanisms for Teaching Inspection and Software Testing. In *21st Conference on Software Engineering Education and Training (CSEE&T'08)*, pages 189-196, Charleston, SC, April 2008..

[14] J. D. Novak. Concept mapping: A useful tool for science education. *Journal of Research in Science Teaching*, 27:937–949, 1990.

[15] M. A. S. Turine, M. C. F. Oliveira, and P. C. Masiero. Designing structured hypertext with HMBS. In *VIII International ACM Hypertext Conference (Hypertext 97)*, pages 241–256, Southampton, UK, April 1997.

[16] A. M. R. Vincenzi, J. C. Maldonado, M. E. Delamaro, E. S. Spoto, and W. E. Wong. Component-based software: An overview of testing. In *Component-Based Software Quality: Methods and Techniques*, pages 99–127, New York, NY, June 2003. Springer-Verlag. Lecture Notes in Computer Science, v. 2693.

[17] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering (TSE)*, 11(04):367–375, 1985.

[18] R. A. Demillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(04):34–41, 1978.

[19] T. Husted and V. Massol. *JUnit in Action*. Manning Publications, 2004.