Teaching Software Engineering Fundamentals in an Introductory Computer Programming Course

Eduardo Santana de Almeida[1], **Ivan do Carmo Machado**[1], **Carlos Vinícius Andrade Silva**[1], **Gecynalda Soares da Silva Gomes**[2]

[1]Computer Science Department, Federal University of Bahia – Salvador – Brazil

[2]Statistics Department, Federal University of Bahia – Salvador – BA – Brazil

{esa,ivanmachado,cvas}@dcc.ufba.br, gecynalda@yahoo.com

Abstract. *Programming is an important part of software development and well-educated professionals is critical for the industry needs. However, in general, the computer programming courses are just focused on the language, constructions, structures, and so on. In this paper, we present a new approach to teach an introductory computer programming course based on software engineering fundamentals. The approach has been applied since 2009 in our university and the results are promising.*

1. Introduction

Software-intensive systems have become increasingly essential parts of everyday activity and of business in the global economy and they can be found in products ranging from mobile phones to space shuttles [Shaw 2000]. It means a trend in software development toward bigger and more complex systems [Boehm 2006]. This is due in part to the fact that computers become more powerful every year, leading users to expect more from them. Moreover, this trend has also been influenced by the expanding use of the Internet for exchanging all kinds of information (texts, pictures, multimedia, etc). Thus, some characteristics are strongly wished by managers, software engineers, and users, such as software that is better adapted to their needs, with short time-to-market and low cost. However, the quality of this software depends on an adequate supply of proficient and up-to-date software engineers.

Nevertheless, software engineers are educated in the traditional ways and it has not produced the supply and quality of developers needed to satisfy the growing demand [Shaw 2000]. In general, the universities offer a general software engineering course spanning several issues related to software development such as project management, requirements engineering, processes models, and so on [Lutz and Bagert 2006]. On the other hand, the introductory courses such as the ones related to computer programming are not linked to the software engineering ones making harder to students understand in a practical way important concepts and fundamentals related to software development.

In this sense, we propose a new approach to teach introductory computer programming courses. Our assumption is that students can learn important software engineering fundamentals embedded in this kind of course through practical projects.

2. Software Engineering Fundamentals

The software engineering (SE) area has been investigated for more than four decades and several efforts were developed to define a set of fundamentals [Selby 2007] and body of knowledge for the field [Abran et al. 2004]. In our approach, we considered all of them as important, but for the specific proposal, it is based on:

- **Collaborative work in small teams:** software development is a complex activity to be performed by someone in an isolated way and people with different skills should be considered. Moreover, the idea is to divide the problem into smaller pieces and assign it for tightly proactive focused teams.

- **Iterative development and Feedback:** the idea is to develop software through repeated cycles and in smaller portions at a time (incremental way) producing functional software at each iteration. Moreover, during the iterations, feedback is incorporated to improve its features in next releases.
- **Modularity:** the goal is to decouple design decisions that are likely to change so that they can be changed independently making easier maintenance tasks [Parnas 1979].
- **Configuration Management:** it identifies the configuration of software at distinct points in time to control changes and maintain the integrity and traceability of the configuration throughout the software lifecycle.
- **Documentation:** software documentation can reduce the time and effort to develop new software, increase the ease of porting software to different platforms, and help users to understand software more easily.
- **Testing:** testing activities support quality assurance by executing the software being studied to gather information about the nature of that software [Harrold 2000].
- **Refactoring:** is a disciplined technique for restructuring an existing piece of code, modifying its internal structure without changing its external behavior presenting benefits related to maintainability and extensibility [Fowler et al. 1999].

3. The Approach

3.1. Structure

Our initial challenge was to define an introductory computer programming course (held at the 2^{nd} semester of the 1^{st} year in a computer science course) based on a set of SE principles once, in general, the computer programming courses are just focused on the language itself discussing its features, constructions, and so on. We believe that students should be taught a proper SE mindset from the beginning of their computer programming education, in this way, their perception for the concepts in future SE courses will be faster as well as their future connection with other important fundamentals.

The computer programming course in the C language is offered after an initial course on computer programming introduction using Pascal (1^{st} semester, 2 classes/week, in a total of 68 hours/semester). The course follows the classical language book [Kernighan and Ritchie 1988] and is composed of sixteen classes in the semester (each class means 3 hours/week) as showed in Table 1. During the classes at the Lab conducted by the professor and the tutor[1], the concepts are explained (30-60 minutes) and after that, a class assignment (hands-on session) divided in 3-5 questions is performed to practice the concepts presented.

3.2. Instrumentation

In order to support the proposed approach, we defined a set of tools to be used in the course:

- **Wiki system:** It is used to organize all the instruments (homework, reading list, spreadsheet links), the activities, and also to keep track of all changes that have been made along the course. It is also used to access data from previous semesters in order to understand the changes performed during the entire course editions;
- **Mailing list:** It is used as a communication channel among students, the tutor and the professor to discuss any topic regarding the course;
- **Version Control System:** It is used to manage all the students produced artifacts on the group work activity. Moreover, it allows the data collection process to evaluate the students, their motivation and participation on each iteration;

[1]In each semester, a Ph.D. student was selected to act as a teaching assistant, in this study named *tutor*.

Table 1. Course structure.

CLASS	TOPICS
1	C overview • Expressions • Statements • *Class assignment*
2	Arrays • *Class assignment*
3	Multidimensional arrays • Strings • *Class assignment*
4	Functions • Class assignment
5	Recursive Functions • *Class assignment*
6	*Class assignment* about the previous topics
7	Pointers • *Class assignment*
8	Dynamic Allocations • *Class assignment*
9	*Class assignment* about pointers
10	Exam
11	Structure, Union, Enumerations• User-defined Types • *Class assignment*
12	File I/O • Console I/O • *Class assignment*
13	Modularity • 1^{st} Project phase start
14	Project presentation • Testing • 2^{nd} Project phase start
15	Project presentation • Testing • 3^{rd} Project phase start
16	Final Project presentation

- **Web Testing Spreadsheet:** It is used to organize and classify all the defects identified during each iteration phase. It is important for students and professors understand the quality of the produced software as well as the pending issues to be fixed;
- **Survey tool:** We collect sensitive data using a survey tool to send two forms to students: the prior was applied in order to gather information about their background in programming; and the latter to obtain feedback after running the proposed course structure.

3.3. Approach Steps

In the class 13, the project course starts and the approach based on the principles defined in Section 2 is executed as follows:

1. Initially, the students are divided into teams, each composed of four/five students (collaborative work in small teams). We do not define a formal way to set the teams [Scott and Cross 1995] and in our case, the best students identified during the classes and exam are put in separated teams and work as the project leader. The remaining ones are signed to the groups in a random way. However, it is important to try to balance the other good students in different teams. In addition, the professor and tutor explain the role of a project leader and his importance in a software development project for the students selected for this role.
2. Next, the professor presents the project specification for each team. The specification is part of a small information system chosen by the professor and is composed of functional requirements with operations to create, remove, retrieve and update entities, and present several types of queries.
3. Based on the guidance from the professor and tutor, the students create the repository structure for their team (Configuration Management) and the technical leader starts modularizing the specification defining the `.h` files and functions to be developed (Modularity and Documentation).
4. The students team have one week to work in the specification (Iterative development) and make the project presentation. During the presentation (30-40 minutes), the students should present the specification under their responsibility, explain the source code and decisions taken, and show the running system. In this presentation, the professor and tutor ask questions for each participant in the team to evaluate their participation in the project and present feedback related to the implementation. The students have to incorporate the feedback if applicable.

5. After the presentation, the professor explains some basic notions about unit and system testing [Harrold 2000] and based on this, he shows a web spreadsheet[2] to report the defects found. This is composed of fields to describe the defects summary, steps to reproduce them, severity, owner, and so on. Thus, each team has an assignment to test the system (with a 3-day deadline) developed by another one and register the defects found (Testing). The professor and tutor make also the testing and report the defects for each team. Finally, the students' team has two days to fix the defect reported.
6. In the following class, the professor starts the second iteration for the project. In this iteration, each team has to work with another one on a new system specification, e.g. team 1 works together with team 2 in one specification and teams 3, 4, and 5 in another one. For this iteration, the professor adds new requirements for the system and make them integrated. As an example, let us consider a system under the university domain with entities such as professors, departments, areas (computer science, math, biology, etc.), courses (data structures, databases, etc.) and students. Thus, in this iteration, team 1 and team 2 have to work with professors and departments, team 3, 4, 5 with areas, courses and students. In this sense, each team has to change their previous iteration source code, add new features, improve the implementation and so on (Refactoring).
7. After the implementation, both teams have to present the running system, answer professor's questions, test the different implementations (team 1 and 2 test team 3, 4, 5 and vice-versa) and fix the defects.
8. In the last iteration, the professor defines a new specification for the system connecting all the previous parts and adds new requirements. In this iteration, all the teams have to work together as one team on the specification, test the whole system, and make the final presentation.

3.4. Additional Skills practiced

Besides the software engineering fundamentals, the students may learn several important skills related to software development:

Leadership and Management. During the project, some students have to work as technical leaders and coordinate the activities in their teams. Moreover, they have to manage the deadlines and make sure that everybody is working on the project.

Requirements understanding. Based on the specifications defined by the professor and tutor, the students have to understand them, interact with the customers (professor and tutor) to clarify issues and make the design, implementation, and testing.

Defects assignments and prioritization. In the testing phase in the project, the students have to assign the defects for each member in the group and set the priorities to fix them according to the defined deadlines.

4. A Preliminary Evaluation

We introduced the approach in our university in 2009.2 period and it has being applied since then. In this study, we consider data gathered from semesters 2009.2, 2010.1 and 2010.2[3]. Table 2 shows some quantitative information about the projects developed by the students.

The set of evaluated students indeed does not represent the initial amount of enrolled studies. From the initial amount of enrolled students (103), 33 did not complete the course, since some of them either dropped it or gave up on the classes and hence stopped taking tests. Some students (13) failed to reach the minimum required score to be approved in the course. We decided not to include these in the analysis. Hence, the final set contains 57 students.

[2]The platform provided by Google - http://code.google.com/ - was used as the project bug report system.
[3]http://disciplinas.dcc.ufba.br/MATA57/20092||20101||20102

Table 2. Case studies using the approach.

PERIOD	APPLICATION DEVELOPED	ENROLLED STUDENTS	EVALUATED STUDENTS
2009.2 - class 1	Hospital Management System	17	14
2009.2 - class 2	University Management System	13	
2010.1 - class 1	Bank System	25	20
2010.1 - class 2	Sports Competition System	19	
2010.2 - class 1	Hospital Management System	17	23
2010.2 - class 2	University Management System	12	

This preliminary evaluation was aimed at understanding how the students consider this new educational approach. Data was collected through questionnaires, applied at the end of each semester, as a kind of feedback, in which students had to rate the following statements on a pre-defined scale, as next explained. These are the *independent variables* of our analysis.

- **Course Content Evaluation:** *knowledge_c:* prior students' knowledge on C programming language; *course_expectancy:* the course met students' expectancy; *course_content:* the course content met students' needs; *course_difficulty:* difficulty level of the course; *material_quality:* evaluate if audiovisual and teaching aids were useful; *interaction_tut_prof:* readiness to solve student's problems; *course_length:* how adequate the course length was; *content_schedule:* the predefined schedule suffices regarding the content; *aplic_nec_interest:* applicable to students' needs and interests in the course;
- **Teaching Evaluation:** Both Tutor and Professor were evaluated according to the items: *_punctuality:* punctuality and regularity; *_knowledge:* subject knowledge; *_didactic:* preparation for the class, interest in teaching the class; *_communication:* communication skills, encouraging students to participate; *_doubts:* ability to clear student's doubts; *_schedule:* the predefined schedule was followed accordingly;

The students were surveyed with questions representing the aforementioned items. These were presented using a Likert-like scale, with values ascending ranging from 0 to 10. Data gathered from the questionnaires were analyzed through the SPSS tool[4]. Besides, after filling out the questionnaire, the students had to give an overall grade to the course. This grade represented the *dependent variable*, called **general evaluation of the course**. The same scale was used.

In order to validate the reliability of the gathered data, we verified the *Cronbach Alpha index* for every item of the constructs, and later of the whole construct. This index is a way to measure the scale reliability, through the analysis of the homogeneity of the items that compose the scale, i.e. it verifies the internal consistency of the questionnaire. It works by extracting *mean* and *std. deviation* values from each variable independently, and next it is analyzed the general measure of the construct. The results can be viewed in Table 3. In general, the indexes suggest satisfactory reliability, presenting values higher than 0.8.

Table 3. Measures of Constructs

CONSTRUCT	NUM. OF ITEMS	ALPHA DE CRONBACH	MEAN	STD. DEV.
Course Evaluation	9	0.8055	6.53	2.38
Teaching Evaluation	12	0.8420	8.64	0.61

Next, we proceeded with a factor analysis, performed in order to assess the constructs validity. The purpose of factor analysis is to discover simple patterns in the pattern of relationships among the variables. In particular, it seeks to discover if the observed variables can be

[4]http://www.spss.com/

explained largely or entirely in terms of a much smaller number of variables called factors. Our intention was to identify the analyzed factors that influenced the general grade the students gave to the course. Tables 4 and 5 show the calculations for the rotations performed to the course content evaluation and the teaching evaluation, respectively. Items were grouped according to their importance for each factor, e.g. in Table 4 the factor 1 was constructed emphasizing the items: *course_expectancy, course_content, material_quality, interaction_tut_prof, content_schedule* and *aplic_nec_interest*. The same idea was applied to the model presented in Table 5.

Table 4. Rotated Factor Matrix - Model 1 (Course Content Evaluation)

VARIABLE	FACTOR		
	1	2	3
knowledge_c	-2,973E-02	-,104	,288
course_expectancy	,910	,115	-,107
course_content	,679	,330	2,683E-02
course_difficulty	,215	,975	-3,684E-02
material_quality	,704	,252	-,109
interaction_tut_prof	,692	,108	-8,240E-02
course_length	4,770E-02	,225	,973
content_schedule	,742	,168	8,229E-02
aplic_nec_interest	,391	-,117	5,805E-02
Explained variance	3,02	1,25	1,07
% of explained variance	33,52	13,92	11,90
% of cumulative explained variance	33,52	47,43	59,34

Table 5. Rotated Factor Matrix - Model 2 (Teaching Evaluation)

VARIABLE	FACTOR		
	1	2	3
professor_punctuality	,258	,368	,118
professor_knowledge	,138	5,087E-02	,287
professor_didactic	,189	,864	,129
professor_communication	,123	,987	9,631E-02
professor_clear_doubts	,193	,726	,198
professor_schedule	-8,309E-02	,236	,878
tutor_punctuality	,396	,106	,289
tutor_knowledge	,566	,213	,235
tutor_didactic	,880	,202	-1,063E-02
tutor_communication	,893	,268	-2,796E-02
tutor_clear_doubts	,888	8,506E-02	,123
tutor_schedule	,115	,112	,951
Explained variance	3,031	2,630	1,992
% of explained variance	25,256	21,913	16,601
% of cumulative explained variance	25,256	47,169	63,770

Multiple Regression analysis was performed, taking as dependent variable the value of *general evaluation of the course*, and as independent variables all items prior presented in this section. The goal of this analysis was to verify the relative importance of each of variables that influenced on the general grade they gave to the course. Table 6 shows that regression model is significant, with a $R^2 = 0.547$, we can state that 54.7% of the total variance of the dependent variable is explained by the regression equation. In general, the mean value for the course evaluation was 7.818, as shown in Table 7. This value indicates that most students consider this new approach as relevant and promising.

5. Lessons Learned

After conducting this new approach for 3 semesters, involving a dozens of students, we identified a set of points that need special attention, since we are intended to keep applying such a new approach for the introductory programming course. The main points are:

Collaborative work. When the students have to work integrating their code with a different team, they present several problems: some of them do not engage in the code activities

Table 6. ANOVA calculations

Model	Sum of Squares	df	Mean Square	F	Sig.
Regression	62,236	2	31,118	38,006	,000[b]
Residual	51,583	63	,819		
Total	113,818	65			

R =0.739, R-Square = 0.547, Adjusted R-Square = 0.532, Std. Error of the Estimate = 0.9

Table 7. Coefficients

Model	Unstand. Coeff.*		Stand. Coeff.*	t	Sig.
	B	Std. Error	Beta		
(Constant)	7,818	0,111		70,193	,000
Content Evaluation[b]	,921	,124	,662	7,441	,000
Schedule, Teaching Knowledge	,251	,121	,184	2,072	,042

since some of them lead the activities in the group; they have difficulties to meet with all the participants because of the different schedules. These problems are usually more visible in the second and third iterations;

Commits and Defect reports. In some situations, a student submits a commit or defect for a member of his group and it makes hard to analyze the data and identifies their participation in the project. It might have happened because they could have network problems or other issue. We indeed had the opportunity to gather information on the amount of errors found, etc, but we rather can not make any inferences on such data, regarding individuals' performance. Hence, data collection should be improved;

Last specification. During all semesters, the last iteration is very problematic for the students. They have several difficulties to work on a team with 20 members in one specification and we noticed problems concerning to work assignment for each member, manage the implementation progress, organize meetings with the whole team, code convention, and common understanding and agreement for the final software. In some situations, they developed two different versions for the system because of the difficulty on agreement.

6. Related Work

In [Van Scoy 1990], the authors present an Ada-based introductory computer science course which introduces some basic SE concepts such as abstraction, information hiding, object-oriented software development and reuse. [Prey et al. 1994] discusses an approach to introduce SE concepts in the first computer science course based on code walkthrough and an analysis of large software systems. At the University of California at Santa Cruz (UCSC), the authors [Bevan et al. 2002] introduced pair programming into a freshman programming classes to prepare students for the software development challenges. Our approach is closed to [Van Scoy 1990]. However, we believe that our proposal is based on more solid SE fundamentals and the project setting defined in the approach allows exercising it systematically.

7. Concluding Remarks

SE education is a challenge for educators, mainly, aligning the industrial needs and the material presented for students, as well as integrating programming and SE courses. In this paper, we addressed the second issue proposing a new approach to teach an introductory computer programming course based on a set of SE fundamentals. The approach allows students to practice several concepts and the professor does not need to explain theoretical issues about it. The approach has been used in our evaluation since 2009 and we believe that it can be useful for achieving the defined goals.

As future work, we are considering to combine our approach for the computer programming introduction course with the SE one. The idea is that students from the SE course based on the classes related to project management and requirements engineering can work during the project as the project managers and system analysts, in this way, they could manage the teams, deadlines, assignments, and define the specification for the students in the computer programming introduction course. Besides, evaluations considering not only the students feedback but rather making correlations with their performance in courses held in the following semesters, referring to SE, in order to identify opportunities for improvement.

Acknowledgements

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES[5]), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1.

References

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., and Tripp, L. L. (2004). *Guide to the Software Engineering Body of Knowledge (SWEBOK).* IEEE. ISO/IEC TR 19759.

Bevan, J., Werner, L., and McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. In *Proceedings of the 15th Conference on Software Engineering Education and Training*, Washington, DC, USA.

Boehm, B. (2006). A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 12–29, New York, NY, USA. ACM.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: improving the design of existing code.* Addison-Wesley, Boston, MA, USA.

Harrold, M. J. (2000). Testing: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 61–72. ACM.

Kernighan, B. W. and Ritchie, D. M. (1988). *The C Programming Language, Second Edition.* Prentice-Hall, Englewood Cliffs, New Jersey.

Lutz, M. J. and Bagert, D. (2006). Guest editors' introduction: Software engineering curriculum development. *IEEE Software*, 23:16–18.

Parnas, D. L. (1979). *On the criteria to be used in decomposing systems into modules*, pages 139–150. Yourdon Press, Upper Saddle River, NJ, USA.

Prey, J. C., Cohoon, J. P., and Fife, G. (1994). Software engineering beginning in the first computer science course. In *Proceedings of the 7th SEI CSEE Conference on Software Engineering Education*, pages 359–374, London, UK. Springer-Verlag.

Scott, T. J. and Cross, II, J. H. (1995). Team selection methods for student programming projects. In *Proceedings of the 8th SEI CSEE Conference on Software Engineering Education*, pages 295–303, London, UK. Springer-Verlag.

Selby, R. W. (2007). *Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research (Practitioners).* Wiley Press.

Shaw, M. (2000). Software engineering education: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 371–380. ACM.

Van Scoy, F. L. (1990). Introduction of software engineering concepts in an ada-based introductory computer science course. In *Proceedings of the SEI conference on Software engineering education*, pages 67–76, New York, NY, USA. Springer-Verlag.

[5]http://www.ines.org.br/