Toward an Approach to Programming Education to Produce Qualified Software Developers

Jaime F. Castillo, Carlos Montes de Oca, Efraín Salomón Flores and Perla Velasco Elizondo
Centre for Mathematical Research, CIMAT. Guanajuato, Guanajuato, 36240, México.
castillo,moca,esalomon,pvelasco@cimat.mx

Abstract

There is a common complaint that undergraduate programs in computing in many developing countries have not been preparing students sufficiently to become truly effective in the software industry. Although some programs in computing are justified in paying little attention to programming, it is important to teach it properly to those wishing to develop software professionally. In order to produce qualified software developers, the focus should be not only on determining which programming-related subjects have to be taught, but also on training academicians to teach these subjects efficiently. In this paper we introduce an approach which tackles the former aspects.

1. Introduction

The *Software Industry* has been recognised as one of the key opportunity areas to achieve modernisation and development in many countries [2], [8]. Multiple experiences demonstrate that one critical factor to achieve these benefits is *people*. In India, Israel and Ireland –which are in the top tier countries for software exports, the skills, expertise and size of their local labour pools have been factors in determining their current state in the Software Industry [2], [11].

In many developing countries, however, software companies are still struggling to grow in part due to the lack of a large pool of competent and talented *software developers*. Their experiences show that most of the graduates do not have the essential knowledge and skills to join the Software Industry and achieve the level of productivity expected [7], [10], [4], [3].

Although some Undergraduate Programs in Computing (UPC) are justified in paying little attention to programming, it is important to teach it properly to those interested in a career in software developer. However, we also consider that for producing qualified software developers, the focus should be not only on determining which programming-related subjects have to be taught, but also on training academicians to teach these subjects efficiently, and on promoting this manner of education in other institutions. Considering all the former, in this paper we introduce an educational approach which we believe tackles the former aspects efficiently.

This paper is organised as follows. In Section 2, we introduce the proposed approach and describe its main elements. Section 3, we discuss and evaluate it against related work. In Section 4, we describe the current state of practice of approach and describe our future plans. Finally in Section 5, we present the conclusions.

2. The Proposed Approach

The approach presented in this paper tackles two main aspects: (*i*) it considers the technical and pedagogic issues around teaching programming to prospective software developers via a *Pillars Model* and (*ii*) it defines the mechanisms to improve and to promote this manner of programming education across UPC in other institutions via an *Implementation Framework*. In the following sections, we describe both the *Pillars Model* and the *Implementation Framework*.

2.1. The Pillars Model

When preparing software developers at early stages of UPC, the transfer of *theoretical* issues is not all that is needed. *Pedagogy* (and didacticism) is also very important because it stresses the process through which knowledge is constructed. Our approach considers these issues in a *Pillars Model* whose pillars define a hierarchy of topics resulting from analysing (*a*) several relevant works on teaching programming, (*b*) the teaching experience of the involved parts in this project and (*c*) the current needs of software companies. Fig. 1 shows the pillars and their hierarchies' top-level content, which is briefly discussed next.

978-0-7695-3539-5/09 $25.00 © 2009 IEEE
DOI 10.1109/CSEET.2009.21

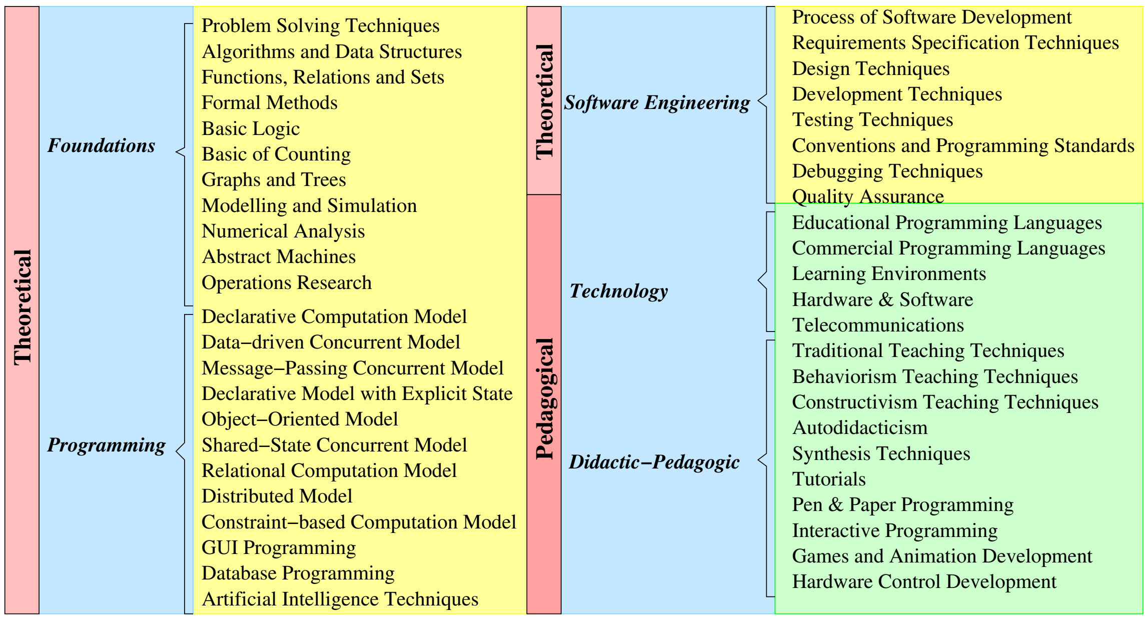
Theoretical
Foundations
Problem Solving Techniques
Algorithms and Data Structures
Functions, Relations and Sets
Formal Methods
Basic Logic
Basic of Counting
Graphs and Trees
Modelling and Simulation
Numerical Analysis
Abstract Machines
Operations Research
Programming
Declarative Computation Model
Data−driven Concurrent Model
Message−Passing Concurrent Model
Declarative Model with Explicit State
Object−Oriented Model
Shared−State Concurrent Model
Relational Computation Model
Distributed Model
Constraint−based Computation Model
GUI Programming
Database Programming
Artificial Intelligence Techniques
Theoretical
Software Engineering
Process of Software Development
Requirements Specification Techniques
Design Techniques
Development Techniques
Testing Techniques
Conventions and Programming Standards
Debugging Techniques
Quality Assurance
Pedagogical
Technology
Educational Programming Languages
Commercial Programming Languages
Learning Environments
Hardware & Software
Telecommunications
Didactic−Pedagogic
Traditional Teaching Techniques
Behaviorism Teaching Techniques
Constructivism Teaching Techniques
Autodidacticism
Synthesis Techniques
Tutorials
Pen & Paper Programming
Interactive Programming
Games and Animation Development
Hardware Control Development

Figure 1: The Pillars Model

2.1.1. Foundations Pillar. The teaching of formal aspects to prospective software engineers is basic in developing their skills in *modelling* and *problem solving*. Constructing a non-real artefact, as software is, requires abstract reasoning and mathematical modelling. However, it seems that the lesser use of formal techniques to software development has contributed to the lack of these skills in current UPC graduates. Our *Foundations Pillar* considers a set of formal subjects which we consider basic to developing these abilities so that the teaching of programming can be oriented to model *computation models*[1] via different formalism.

2.1.2. Programming. Current experiences demonstrate that for students who have a weak background in programming it is harder to make a successful career as a software developer because advancing their abilities takes longer. The study of programming languages as a means of teaching programming gives little insight into programming as a theoretical process. Instead, our *Programming Pillar* includes a set of computation models by which students can focus on understanding various ways for implementing computation rather than on the particular syntax of a programming language.

2.1.3. Software Engineering Pillar. It has been said that "programming without Software Engineering (SE) is like sculpting with a saw". Despite that, in many UPC SE is offered as either an optional or a late course. Introducing SE topics at early stages of UPC allows students to not only understand the necessity of constructing software using a disciplined and systematic way, but also to initiate their adherence to SE practices. Our *Software Engineering Pillar* includes the set of topics we consider relevant in this context.

1. In [9] a computation model is defined as a formal system that states how computations are done.

2.1.4. Technology Pillar. We are convinced that the "technology revolution" demands that institutions include the technology in their teaching activities. However, we consider that the first priority must be teaching and learning and the technology should only be regarded as a tool of efficiency to teach and learn better. By analysing various works, we have identified several technology guidelines that should be considered for the teaching of the topics in the *Foundations*, *Programming* and *Software Engineering Pillars*.

2.1.5. Didactic-Pedagogic Pillar. The use of adequate tools to support the teaching process is not all that is needed to develop an efficient teaching environment. The role of the teacher and his skills are also important. We agree with the adage "a teacher teaches the way he was taught". Thus, in our model the *Didactic-Pedagogic Pillar* provides the guidance to help teachers improve their teaching skills so that all the issues considered in the model can be efficiently taught.

2.2. The Implementation Framework

The *Implementation Framework* defines the process to prepare teachers to teach according to the *Pillars Model* and to promote what is taught to other institutions via a public body designated the *Board*. Besides the *Board*, there are other roles in our framework: the *Instructor*, the *Faculty* and the *Students*, see Fig. 2. For clarity, we will explain the process' flow assuming there is no *Board*.

First in our process we prepare the course (Prepare Course) –according to the *Pillars Model*, and select the faculty members to be trained (Select Faculty); both activities are carried out by the *Instructor*. An assessment of the candidate faculty members is required to achieve the success of the course as not all of them may have the required level with regard to the approach to implement. Then, with a group of faculty members assigned, the teaching-learning activities can be carried out (Teach Course and Take Course). After that, the members of the *Faculty* are evaluated by the *Instructor* (Evaluate Faculty), then faculty members can start teaching to the *Students* at their respective institutions according to the received training.

As can be seen in Fig. 2, the Evaluate Faculty activity leads to the creation of the *Board* (Create Board), which is a body of experts in teaching programming that works as a medium to spread the acquired knowledge. The faculty members with the best grades in the evaluation activity are the potential candidates to comprise the *Board*. The monitoring of the activities made by the *Board* (Monitoring Board), as well as the ones made by the *Faculty* at their respective universities (Monitoring Faculty), are carried out by the *Instructor* to ensure that they comply with their requirements and generate the expected results. When a *Board* exists, activities such as Select Faculty, Teach Course and Evaluate Faculty are then carried out by the *Board* instead of the *Instructor* (see the decision point in Fig. 2).

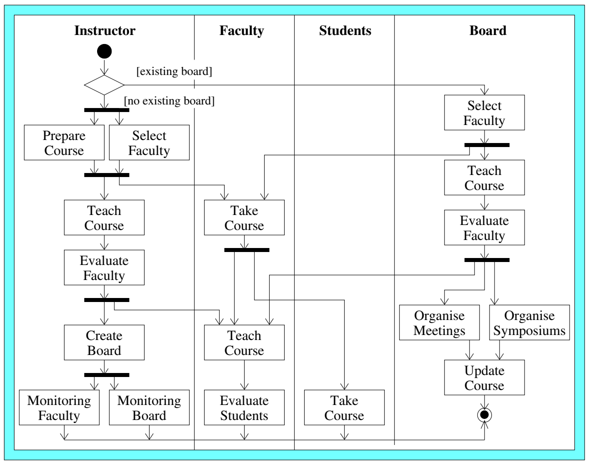

Figure 2: The *Implementation Framework*

Finally, the Organise Meetings and Symposiums activities in our framework work as the sources to direct all decision making management (Update Course) with regard to the establishment of a solid community of programming educators.

Now we have explained our approach, in the following section, we compare it to related work.

3. Related Work

Many approaches and tools to improve the teaching of programming have been proposed. For example, animation/simulation/visualisation educational languages have often been utilised to help students to better understand programming constructs. The idea behind these languages is mapping abstract ideas to graphical representations, which can be better understood by students.[2] However, it is not always clear how these languages should be taught efficiently.

A wider context in the teaching of programming has been considered by [12] and [5]. The former work develops a cognitive approach for learning programming based on learning patterns. The later work introduces a repository for creating and delivering learning objects for a programming course. In our view, these approaches are hardly oriented to the pedagogical aspects of teaching as no consideration is given to the topics that have to be taught to deal with the goal of producing qualified software developers.

A group of lecturers at the Otago Polytechnic developed a sequence of introductory programming courses meant to provide students with good core programming skills and competence in both the Procedural and Object-Oriented paradigms [6]. In contrast to previous works, this work seems to be strongly oriented to the teaching of programming, and some pedagogical principles are utilised to select the topics to be taught. However, it does not consider current demands of software development companies nor the issue of how these courses can be spread to other institutions.

Thus, we consider our approach unique because it considers the technical, pedagogic and organisational of around producing qualified software developers. It considers a set of justified programming-related topics to be taught to prospective software developers at early stages. That is, the *what to teach?* and the *why to teach it?* Secondly, it also considers the form in which the topics have to be taught efficiently. That is, the *how to teach it?* Finally, it also considers a scheme to improve this way of teaching and to promote it across other institutions. That is, the *how to improve it?* and the *how to spread it?* As far as we know, there is no work that considers all these aspects.

2. An example of such languages is JAWAA [1].

Although our work lacks of full evidence of its effectiveness due to it is still ongoing research, we considered we have the conditions to obtain such evidence in the coming months, as we discuss next.

4. Current State and Future Actions

The implementation of any educational approach is unlikely to happen without an outside stimulus. Fortunately, we are actively involved in an IT Industry Development Program in our country, which has as one of its main efforts the improvement of UPC to better prepare professionals for the Software Industry. We are now working with seven academic institutions of one state in our country. All of them are interested in preparing software developers as there is a potential market in which their graduates can be employed.

We have already designed a set of programming courses according to the *Pillars Model*, as well as trained the faculty members of the participant institutions according to the *Implementation Framework*. The grades and feedback obtained from this exercise are positive. However, these data is only part of the exercise to evaluate our approach in full. In the following months, we will select one person from the set of faculty members trained. We will monitor and evaluate the teaching process of this person at his academic institution. Once the teaching process is completed, a software company will test the trained students. In this test they will evaluate the skills they consider most important for "entry-level" software developers. We have already selected a company for such purposes. The selected company has a large data record of "entry-level" examinations given to prospective employees in the last five years, which will give us useful evaluation data.

5. Conclusions

Although some UPC are justified in paying little attention to programming, it is important for UPC to provide the required foundations to those wishing to develop software professionally. In this paper we propose an approach to improve the teaching of programming at early stages in UPC. The approach has two main elements: (*i*) *a Pillars Model* –which defines the core issues within the teaching of programming and (*ii*) *an Implementation Framework* –which defines how the core issues are transmitted to faculty members and promoted to other institutions. As far as the work known so far, there is no approach considering the two elements. Although our work is still in progress, we have the conditions to objectively evaluate it in the short term and further build our understanding of programming education.

Acknowledgment

This work was supported by CONCyTEG under grant 08-02-K662-119.

References

[1] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S.H. Rodger. JAWAA: Easy web-based animation from CS 0 to advanced CS courses. In *Proc. of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 162-166, 2003. ACM.

[2] D. Breznitz. Innovation-based industrial policy in emerging economies? the case of Israel's IT industry. *The Journal of Systems and Software*, 8(3):1–38, 2006.

[3] J.J. Cappel. Entry-level IS job skills: A survey of employers. *Journal of Computer Information Systems*, 42(2):76–82, 2002.

[4] X. Fang, S. XiangKoh, and S. Lee. Transition of knowledge/skills requirement for entry-level is professionals: An exploratory study based on recruiters' perception. *Journal of Computer Information Systems*, 46(1):58–70, October 2005.

[5] A. Gunawardena and V. Adamchik. A customized learning objects approach to teaching programming. *SIGCSE Bulletin*, 35(3):264–264, 2003.

[6] P. Haden and Dr S. Mann. The trouble with teaching programming. In *Proc. of the 16th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ)*, pages 63–70, Palmerston North, New Zealand., 2003.

[7] M.E. McMurtrey, J.P. Downey, S.M. Zeltmann, and W.H. Friedman. Critical skill sets of entry-level it professionals: An empirical examination of perceptions from field personnel. *Journal of Information Technology Education*, 7(2008):102–120.

[8] B. Meyer. The unspoken revolution in software engineering. *IEEE Computer*, 39(1):124–123, 2006.

[9] P. Van Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, Cambridge, MA, USA, 2004.

[10] M. Shaheen and Z.U. Rehman. Critical skills for computer academicians course proposal. In *Proc. of the Informing Science and IT Education Conference (InSITE)*, pages 405–421, 2008.

[11] M. Teubal. The Indian software industry from an Israeli perspective: A microeconomic and policy analysis. *Science Technology & Society*, 7(1):151–186, 2002.

[12] D. Traynor and J.P. Gibson. Towards the development of a cognitive model of programming. A software engineering proposal. In *Proc. of the 14th Workshop of Psychology of Programming Interest Group*, 2004.