The Challenges of Teaching and Learning Software Programming to Novice Students

Seyed Reza Shahamiri
Manukau Institute of Technology, New Zealand

INTRODUCTION

Software Engineering is an engineering discipline that involves with all aspect of software development that applies engineering approaches in order to deliver high quality software products (Pressman, 2005). One of the important stages of software production is coding in which software blueprints are realized via a programming language; a programming language is a language that is understandable for computers (Sommervile, 2007). Table 1 shows the general phases of software development. Coding and programming language skills are required from phase four but knowledge and understanding of programming language is very helpful in previous phases in order to successfully complete a software project.

There has been a dramatic demand increase for software applications that promises a rewarding carrier for those who poses the required skills. With the fast advances in technology and emerging ones like driverless cars, Internet of Things (IoT) (Pandya & Champaneria, 2015), Big Data (Sharma & Mangat, 2015), Software Defined Networks (Bizanis & Kuipers, 2016) etc. it is expected that more software programmers will be required in the future. As the result of such demands, various online and offline courses to introductory programming have been provided.

While knowing software coding is a very useful skill, it is difficult to learn programming especially at the beginning level since acquisition of complex new knowledge, associated strategies, and practical skills are required (Robins, Rountree, & Rountree, 2003). Software development courses are generally among difficult subjects and have low pass rates; according to (Dehnadi & Bornat, 2006) the fail rate of first programming papers in university computer science programmes can be up to 60 percent.

However, what are the reasons that make learning and teaching programming difficult? Why do students find it so challenging? And, why the success rates of programming classes are amongst the lowest in computer science papers? The next section tries to identify the reasons and issues

Table 1. Generic software development life cycle phases

No	Name	Description
1	*Planning*	The planning phase is the fundamental process of understanding why an information system should be built and determining how the project team will go about building it.
2	*Analysis*	The analysis phase answers the questions of who will use the system, what the system will do, and where and when it will be used.
3	*Design*	Based on the user requirements, planning and the detailed analysis, the new system must be designed i.e. a blueprint of the system is created by designing the technical architecture.
4	*Implementation*	Actually implementing the designed system; writing software programs using software languages.
5	*Testing*	Checking whether the implemented software works according to specified requirements; fixing bugs/errors.
6	*Maintenance*	To ensure that the implemented system is properly functioning as per the requirements.

DOI: 10.4018/978-1-5225-2255-3.ch643

that make learning introductory programming challenging. Next, some teaching and learning guidelines are provided to facilitate some of the identified challenges. Finally, recommendations for future studies are provided. The guidelines provided in this chapter are based on the literature and the author's extensive experience in teaching software programming.

BACKGROUND

This section provides a background of the problem and explains the issues in software programming teaching and learning.

Since software is intangible and cannot be seen or touched, creating software products can be a very complex task in comparison to most of the other engineering products; a software project can fail easily and lead to poor quality and unreliable products (Ammann & Offutt, 2008). On the other hand, software applications play very important and critical roles in modern life because they control vital operations that require attributes such as security, reliability, performance, etc., qualities that are hard to achieve (Spillner, Linz, & Schaefer, 2007).

Because of the intangible nature of software, and since students cannot directly sense what they have created, it can become very complex to them to successfully implement, debug and verify the product. In addition, it is more challenging to teach and learn the introduction of coding since this is where the students understand the very basics of software nature and expose to entities and concepts that are generally new to them. They also need to design algorithms and implement them via programming language tools and commands in order to perform the required software functionalities.

Although teaching and learning a programming language may be seen as teaching human languages such as English, it is much more complicated. Teaching a human language is mainly concerned about the language syntax, learning the vocabularies, and how to put them in to use. This part is similar to teaching programming language as there are syntax and vocabularies (i.e. commands) with specific meanings in that particular language. The main difference is, in teaching human languages we show the students the tools to express their taught and feelings in order to communicate, but we do not teach them what to say. To put it differently, we all know what we want to say, typically, but we may not know how to do it in a particular language. On the other hand, with programming languages we need to teach our students what to ask a computer together with how to ask it, and we need to be very specific while following the language syntax and steps of the algorithms thoroughly as computers have zero tolerance in regards to syntax errors. In other words, computer programs do not run even with a single syntax error.

Furthermore, there is a difference between coding knowledge and strategies. The former provides a declarative nature of programming, such as being able to state how an algorithm works, but the last deals with the practical applications of the knowledge, such as how and where to apply the algorithm (Davies, 1993). Programmers need to think outside the box and be creative in applying and creating knowledge in order to formulate a software solution.

Essentially learning programming requires the learners to develop their problem solving skills. As mentioned earlier, the first step of software development is to understand the problem domain, analyze it, and then to propose a solution including design concepts and algorithms to perform the required functionalities. The next step is to relaying commands to computer systems using a programing language in order to implement the designs and algorithms. The challenge is, regardless of how experience the programmer is, there will be various errors either due to syntax mismatch or incorrect semantic, and they need to deal with unexpected program behaviors frequently. Although modern Integrated Development Environments (IDEs) perform analysis on the code to highlight most of the syntax errors and some obvious compile