Introduction

Learning how to program computers is difficult, as backed up by the high drop-out rates in computer science programs around the world. Robbins (2003) This difficulty was first detected early in the development of computers, and has been researched over the ensuing years, with mixed results to date.

This paper will touch on a range of available literature about two different pedagogical approaches to teaching computer science, Problem Based Learning (PBL) and Worked-Out Examples. Given that a significant component of transitioning from a novice programmer to an expert programmer is acquiring problem-solving skills, PBL would seem to be a natural fit. However, at this time, that does not seem to be borne out in the literature. This paper will examine the reasons for this, and report the overall conclusions I reached.

Novice Programmers

Key phrase here is “knowledge and skills”. The definition of a Novice Program is that the student lacks both knowledge and skills. Robins, Rountree and Rountree (2003) identify Novice Programmers as those who “… lack the specific knowledge and skills of experts. Novices are limited to surface knowledge (and organize knowledge based on superficial similarities); lack detailed mental models; fail to apply relevant knowledge; use general problem solving strategies (rather than problem-specific or programming-specific strategies); and approach programming “line by line” rather than at the level of meaningful program “chunks” or structures. In contrast to experts, novices spend very little time planning. They also spend little time testing code, and tend to attempt small “local” fixes rather than significantly reformulating programs. They are frequently poor at tracing/tracking code. Novices can have a poor grasp of the basic sequential nature of program execution.”

Novice versus Expert programmers

In addition to popular press reporting, research has shown that it takes about 10 years for a novice programmer to mature into an expert Robins (2003). The novice starts with no knowledge of programming constructs and no problem-solving strategies. Further, because the novice has not worked, there is also a reasonable possibility that the novice has little domain experience to bring to bear on the problem.
Cognitive Load

The next factor to consider when evaluating PBL is cognitive work load. Cognitive workload is described as the mental effort exerted to first build a mental model or scheme in short term memory and then move the scheme into long term memory. This workload breaks down into three type of loads, i.e. intrinsic, extraneous and germane. Chen (2010)

Intrinsic workload is the scope of the problem, i.e. the factors involved. The higher the number of factors, the higher the workload. In Python programming, the intrinsic load factor is high given that, while the selection of statements in Python can be categorized as sequence, selection and integration, the combination of statements is vast and the number of functions that can be brought to bear is even larger.

As students use their cognitive resources to build their schema, there are extraneous factors. An example of extraneous load in programming is learning how to use the development environment. It makes no difference in executing the program whether the code was compiled or interpreted, yet the student must expend cognitive capacity to figure this out. There is an entire area of study devoted to building software that reduces the extraneous cognitive load for learning how to write computer programs.

Germane cognitive load is the flip side of extraneous cognitive load. The student learns the programming elements necessary to have a program run successfully. One example of a germane load is the effort expended to learn the syntax of the language and then applying it to solve a problem.

While PBL does not completely eschew scaffolding, it does try to minimize it, first by limiting the time it is available (fading) or limiting it to verbal discussions primarily among the students. Haruehansawasin (2017) For this paper, scaffolding is the act of imbuing a student with the minimum knowledge need to build a schema. This schema will enable the student to successfully solve a problem using a programming language.

There are two challenges with scaffolding: 1) Recognition that the level of scaffolding varies from student to student due to prior experience. 2) Identification of the pedagogy used to deliver the scaffolding. Usually scaffolding is part of a given pedagogy such as PBL, but this does not have to be the case.
Transfer

The goal of an education process in an academic setting is to provide the student with the knowledge and skills to solve real world problems. Unfortunately, studies have shown that currently this goal is not yet being reached. Chen (2004) (This could form the basis of an interesting future study.)

Problem-Based Learning

My first observational impression of Problem-Based Learning (PBL) was that it taught problem solving by having the student solve the problem using student-directed effort. As it turns out, this is not actually the case. PBL is defined by Savery (2006) by its characteristics. He states:

“PBL is an instructional (and curricular) learner-centered approach that empowers learners to conduct research, integrate theory and practice, and apply knowledge and skills to develop a viable solution to a defined problem. Critical to the success of the approach is the selection of ill-structured problems (often interdisciplinary) and a tutor who guides the learning process and conducts a thorough debriefing at the conclusion of the learning experience. Several authors have described the characteristics and features required for a successful PBL approach to instruction.”

Further, he summarized the main points as:

- Students must have the responsibility for their own learning. PBL is a learner-centered approach—students engage with the problem with
- The problem simulations used in problem-based learning must be ill-structured and allow for free inquiry. Learning should be integrated from a wide range of disciplines or subjects.
- Collaboration is essential.
- What students learn during their self-directed learning must be applied back to the problem with reanalysis and resolution.
- A closing analysis of what has been learned from work with the problem and a discussion of what concepts and principles have been learned are essential.
- Self and peer assessment should be carried out at the completion of each problem and at the end of every curricular unit.
- The activities carried out in problem-based learning must be those valued in the real world.
- Student examinations must measure student progress towards the goals of problem-based learning.
- Problem-based learning must be the pedagogical base in the curriculum and not part of a didactic curriculum.
Advantages of Problem-based Learning in teaching programming

Group learning is one area in which PBL has an inherent advantage. While the popular view of programming is that of a lone person sitting in front of a monitor and keyboard for hours on end, the reality is that almost all code has to interact with other parts of the program. Therefore, the programmer must be able to work in a group environment. Further problems that require an automated solution are not small – to be handled by “throw a bit of code at it” solutions. Rather, they are large, complex and require a multi-person team to solve.

Another stated advantage of PBL is peer learning. In a standard lecture class, there are only the lecturer and the material as the sources of knowledge. In PBL, students learn from each other, starting with first learning that they are not alone in their lack of understanding. This can lead to increased motivation and to camaraderie, an important factor in group dynamics. Nuutila (2005)

A further advantage of PBL is that it frees the teacher to observe and provide feedback. Not only can the teacher observe and provide guidance on the skills and knowledge being acquired, he or she also can provide feedback on the interpersonal development and adaptation to working-in-groups skills of each individual. These social skills are invaluable as the student advances into the workforce.

PBL changes the pace and, to a large degree, the direction of the curriculum. With PBL, there is no longer Lesson 1, Lesson 2, etc. Rather the ill-structured problem is presented and the work pace and direction is set by the team. The tutor is there to make sure that progress occurs, but not to lead the team. This leads to intangible strengths emerging within the group such as drive, passion, creativity, empathy and resilience. The feeling in the PBL community is that benefit cannot be presented lecture-style, but must be acquired through experience. Markham (2011)

Potential Pitfalls When Using PBL

With PBL, the students are given a complex, ill-structured problem and tasked with solving it. It is up to them to completely determine a strategy of how to figure out the problem and then acquire the knowledge to apply through group discussion. When the student is in a novice program, they possess neither the skills nor the knowledge to successfully solve the problem. Given this overwhelming task, they can succumb to overloading their cognitive abilities. For example, they have to resolve and understand how to manage the overwhelming sources of information available to them. Googling “variable” returns 399 million entries – a huge extraneous cognitive load.
Weak problem strategies will also contribute to cognitive overload. The group will learn better strategies but this is developing concurrently with the knowledge issues described above. Sern (2014)

**Worked-out Examples**

Worked-out examples, as the name implies, provide programming solutions to a problem. Sern (2014) The theory behind the pedagogy is that the novice programmer needs some place to start. Worked-out examples give them a place to start by providing the recurring aspects of a task. Further, the novice can have early success without cognitive overload, i.e. excessive stress.

Once the novice programmer has acquired a baseline set of recurring aspect knowledge to work with and some rudimentary problem solving skills, the worked-out problem pedagogy is enhanced, moving from being product-based to process-based. Product-based worked-out examples consist of only the recurring knowledge such as how to assign a variable. Process-based worked out examples add in the “how” and the “why”, such as the realization that a variable is a location in memory with the value. This starts to improve the novice programmer’s problems solving skills and advances them toward the goal of themselves being able to transfer these skills to high order problems and/or other domains. Van Gog(2004). Figure 1 shows process oriented worked-out examples.

![Figure 1 - 1. Elements of process-oriented worked examples for modeling recurrent non-recurrent aspects of a complex cognitive skill. Note: The darker grey area encloses the elements of product-oriented worked-out examples.](image-url)
Fading

When using worked-out examples, the goal is to transition the novice programmer to work on their own. This is accomplished by fading. There are two approaches to making the novice programmer self-sufficient, forward fading and backwards fading. Forward fading consists of removing the first part of the problem and letting the novice program solve the problem by providing the solution. Backwards fading provides the beginning of the problem and the novice programmer completes the problem. Gog (2004)

Reduced Extraneous Cognitive Load

A primary goal of worked-out examples is to reduce extraneous cognitive load. From research, some guidelines have emerged. Gog (2004) Table 1 highlights three design guidelines. Two of particular interest are: “Avoid Split Attention” and “Use Multiple Modalities”. Both point to the effort needed to design presentation materials that use multiple modalities without splitting attention.

Table 1. Design guidelines for worked-out examples aimed at reducing extraneous load (source: Gog (2004))

<table>
<thead>
<tr>
<th>Design Guideline</th>
<th>Implication</th>
<th>For process-oriented examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid Split Attention</td>
<td>Multiple mutually-referring information sources. (e.g., text and picture, text and text) should be offered in an integrated format</td>
<td>How should the extra information (&quot;why&quot; and/or &quot;how&quot; information) be integrated?</td>
</tr>
<tr>
<td>Avoid Redundancy</td>
<td>If one information source is redundant (e.g., text) because the other is intelligible by itself (e.g., diagram) the redundant source should be left out.</td>
<td>Is the &quot;why&quot; information redundant if the students are able to self-explain the procedure?</td>
</tr>
<tr>
<td>Use Multiple Modalities</td>
<td>In multimedia consisting of pictorial and textual information, the text is presented orally rather than visually.</td>
<td>In what modality should the &quot;why&quot; and/or the &quot;how&quot; information in the text be presented?</td>
</tr>
</tbody>
</table>
I have been teaching at West Point for two and a half years. Before teaching at West Point, I taught Introduction CIS 2010 Intro Comp-based Info Sys at Georgia State University as an adjunct professor. Prior to my time at Georgia State, I had not had professional teaching experience. Sadly, as is true in most large universities, the system at Ga State threw me in the deep end of the pool and hoped I could swim. On the first day, they handed me the lesson materials from Pierson including PowerPoint slides and a textbook and the key to my classroom, then told me the password to the classroom computer and wished me “good luck”. I never saw anyone from administration after that.

From my recent studies at the Georgia Institute of Technology, I had learned about technical education pedagogies from a graduate course on educational technology. In that course, we learned about the overarching concepts of objectivism, behaviorism and constructivism, and the underpinnings of these approaches such as scaffolding, novice learners, expert learners and other pedagogical approaches. Although useful foundationally, this did not quite prepare me to apply these concepts in an academic environment as I was just learning classroom management skills and overcoming technical issues with Blackboard.

Coming to West Point put me on a new track, starting with attending a Faculty Development Workshop (FDW) given to all incoming EECS faculty. This “school to learn how to teach” implicitly taught me how to teach the introductory programming course, IT 105 by reviewing the material and using worked-out examples for a number of the lessons. The worked-out example consisted of senior faculty teaching a lesson, followed by the student instructor employing that model to teach a different lesson.

The fact that I now can describe the Faculty Development Workshop in academic terms is all due to the Master Teacher Program. This is not idle praise, but a dawning realization. The program, through discussion, readings, and, research for this paper has provided me with the foundation and the tools needed to improve my teaching. While I have already used some Master Teacher Program tools such as in-class surveys, I am excited to be applying some of the larger concepts such as Worked-Out Examples on a broader scale in my class for the rest of this semester and with more vigor next fall.

Conclusion

Given that BPL focuses on problem-solving and enthusiastic motivation to tackle the problem, there are very difficult challenges in using it in the novice program. Programming demands both broad-based knowledge of complex interactions and problems-solving skills that can address ill-structured problems. This is a tall order and may be why even universities who can draw from
the best and brightest have difficulty teaching programming. Like in any number of other highly complex skills, it takes about 10 years to become an expert programmer.

In completing this analysis of two methods for teaching a novice computer programmer, I found clear parallels in the novice’s situation to my own experience as a novice instructor in recent years. The recognition that many of the questions and issues that I faced are the same that the novice programmers face was very helpful. My first-hand familiarity with the issues and the methods used complemented the objective analyses conducted by knowledgeable people in the field, primarily including those cited here such as Gog, Haruehansawasin, Robbins, Roundtree & Roundtree, Savery and Sern. I experienced first-hand both the PBL approach ("sink or swim" at Georgia State University) and the Worked-Out Examples approach (West Point Faculty Development Workshop). The much greater effectiveness of the Faculty Development Workshop approach in my experience further validates the higher effectiveness discussed in the literature. Based on the literature review, without reservation I have concluded that the Worked-Out Examples approach to the instruction of novice programmers is the better-suited method. Because of the specific characteristics of computer programming, the optimum approach to teaching novice programmers is to start with worked-out examples that are product-based. Once the novice builds a strong foundation in this way, they can then move to process-based worked-out examples.
Bibliography


