Session Number: 3420

Using Introductory Computer Science as a Tool for Teaching General Problem Solving

By Major Timothy G. Nix

Affiliation: United States Military Academy, West Point, NY

Abstract

The primary purpose of the U.S. Military Academy at West Point is to produce leaders for the U.S. Army. Thus, the curriculum is tailored with this goal in mind. One of the selling points of the computer science program is its emphasis on problem solving. The premise is that the problem solving skills that are instilled through computer science can be extrapolated to problem solving in general and easily applied to problem solving in the U. S. Army. All cadets are required to take two or more courses in Information Technology and/or Computer Science. At a minimum, a cadet can take two classes in Information Technology which teach basic problem solving as part of their curriculum. Additionally, some students select a three course engineering sequence in computer science which further develops their problem solving skills. Finally, those cadets who major in computer science are exposed to advanced methodologies such as an object-oriented approach to problem solving. This paper examines the approach to teaching problem solving within the introductory core information technology course (IT105) and the first course of the Computer Science major (CS301). First, this paper will address the introductory techniques taught to all cadets within the first course. Next, it will discuss how these techniques are enhanced and extended in the second. Finally, it will describe how the tools and techniques taught within these two courses extrapolate to problem solving in general.

Introduction

First year college students cover the full spectrum in their ability to solve problems. However, very few have an instilled methodology, or systematic approach to problem solving. Usually, students see the ability to solve problems as a skill you either have or you don’t, not something that can be learned. Software Engineering, on the other hand, is an area in which a good methodology for problem solving is critical for the development of complex, high-end computer programs. A computer programmer may be able to sit down at a keyboard and hack out smaller programs on the fly, but if the same approach is taken on larger projects, the results will be a poorly designed, hard to maintain, buggy system.

Students generally are able to correlate the concept of abstracting a specific methodology for problem solving to a more general methodology. However, they do need to be taught the specifics of a methodology within the context in which it is used. Then they need to be shown how to modify the methodology outside of that context to a more general case.

Why is Problem Solving Important?

The United States Military Academy is dedicated to producing leaders for the U.S. Army and the United States. A leader in any capacity must be able to solve problems. An Army Officer faces challenges on a daily basis. Both the Chief of Staff of the Army down to the second lieutenant maneuvering troops in Iraq must be able to develop quality solutions to often unique problems. Problem solving is not inherent; it is learned. Thus, a critical goal of the Electrical Engineering and Computer Science Department at West Point is to provide students with a skill base to perform their job, independent of whether they are dealing with technology or not.

IT105 Problem Solving

At the West Pont, all cadets are required to take an introductory course in information technology (IT105) as freshmen (Plebes). A major portion of the course is dedicated to problem solving in the context of computer programming. The goal of the course is not to produce a student body (known as the Corps of Cadets) full of expert computer programmers, but rather to produce students who are comfortable with technology and have internalized and practiced a basic problem solving methodology.

All problem solving methodologies tend to tie process to products [1]. At each step in the process, one or more products are developed to help bridge the gap between the problem space and the solution space. Our methodology is no different. We begin with the standard lifecycle process for a system (see Figure 1) [2]. Each step in the process has an associated product. During the analysis phase, the students are taught to develop a simplified Problem Specification. During the design phase, the students develop an algorithm using either pseudocode or a flowchart that attempts to answer the Problem Spec. At this point, the test plan is also developed. Implementing the algorithm consists of converting the pseudocode or flowchart into a computer program – in our case the language of choice is Java. Finally, the test plan is executed to check the design and implementation of the solution against the Problem Specification and the Problem Statement.

The purpose of the Problem Specification is to help map out the problem space in order to gain an initial understanding of the problem to be solved. The Problem Specification consists of the problem objective (specified and implied tasks); the expected output (in appropriate units); the input(s) to the problem (in appropriate units);

assumptions; constraints; and relevant equations. Though common to many methodologies, this format is particularly chosen because it introduces some terms and concepts that are a part of the methodology used by military staff for mission planning (namely specified and implied tasks, assumptions, and constraints). These concepts tend to give students the most difficulty, whether it is correctly identifying implied tasks, assuming away vital parts of the problem, or creating self-imposed constraints that blind the student towards potential solutions.

The design of the solution is achieved either through flowcharts or a simple pseudocode known as PDL (Program Design Language) [3]. Though flowcharts have fallen into disuse because they are not effective for large projects, they are ideal for the scale and scope of problems in IT105. The use of PDL is new to the course. In the past, we used an outline format to develop algorithms. The benefit was that is was simple and straightforward. The problem with this approach was that an algorithm developed in this

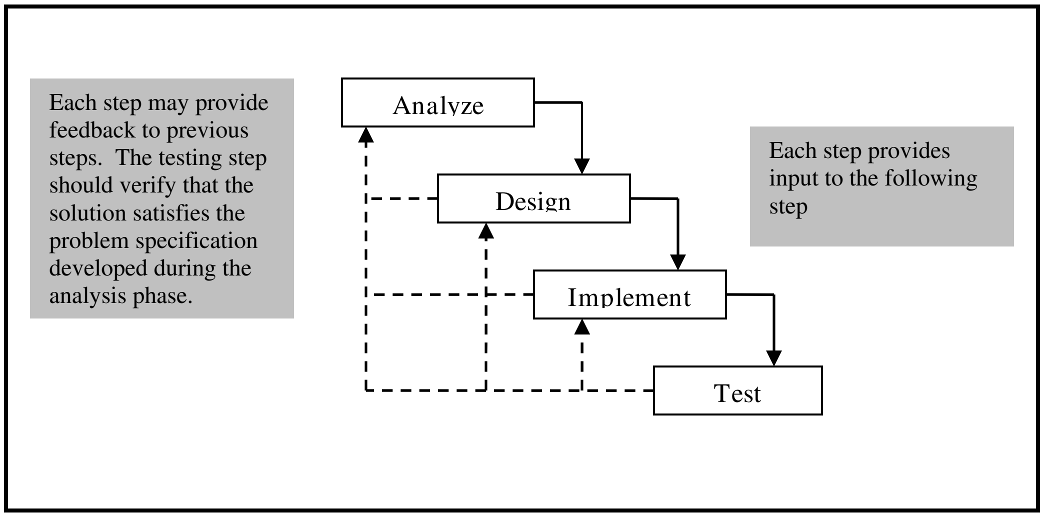
Each step may provide feedback to previous steps. The testing step should verify that the solution satisfies the problem specification developed during the analysis phase.
Analyze
Design
Implement
Test
Each step provides input to the following step

Figure 1: The four-step problem solving methodology.

format was not easily translated into an implementation - as it did not match the structure and flow of the programming language. Therefore, we opted to use PDL which is still very English-based, but has a pseudocode structure for selection and iteration. The cost of using PDL is that it requires a little more time to teach, however we find that we later save time when selection and iteration structures of Java are formally taught. The focus of both of these tools is not only algorithm development, but also achieving the appropriate level of abstraction.

If the flowchart or PDL document is adequate, then implementation is fairly simple. Implementation is done in Java, but with many of the object-oriented aspects removed. I/O has been simplified through a provided library of static methods. All students develop their programming code within a single class file and all shared variables are global. Also, the programming environment is within a text editor specially developed for the course. Students are taught the absolute basics of Java programming, because programming is only the medium for teaching problem solving.

Testing is done after implementation, but the test plan is written before implementation. Developing the test plan prior to implementation provides another tool for helping the student fully understand the solution space. Every path within the execution flow should be tested at least once. The branches and sequences within the program occur at boundary conditions and these boundary conditions should be tested thoroughly. Additionally, depending on the problem to be solved, input validation tends to be an important focus for test cases.

The test plan is a table consisting of, for each test case: a reason for the test; inputs to the program; expected results; and actual results. All but the last column should be developed prior to implementation. The test plan helps to validate the design of the solution. Main areas of concern when teaching test plans are how thorough is the test plan and how specific and well thought out is the reason for each specific test case. Once the implementation is complete, the test plan is executed and the results are annotated. At this point, the implemented solution either works or the iterative nature of the problem solving process provides the student the opportunity to go back to the appropriate stage within the process and fix the errors.

CS300/301 Problem Solving

Long term retention of important concepts comes only through reinforcement. Thus, problem solving is again covered in CS301. CS301 is a course in computer programming designed as the introductory course for majors typically taken during the second semester of their sophomore (Yearling) year. Within the course, the implementation language of choice is Ada 95 and the emphasis is on structured programming.

Since the course is a programming course, the students spend most of the semester learning the specifics of Ada. This also provides the opportunity for the students to solve more complex and complicated problems. Thus, a good methodology to problem solving becomes even more critical. Most of the products used in IT105 problem solving process are reinforced, but there are some minor differences and some additions, driven mainly by the changes is the scope of the projects. Thus, flowcharts are not a requirement within the course (though the students are free to use them), functional decomposition is formalized and the problem specification is tied more to the functional decomposition.

The projects within CS301 are specifically designed to be complex enough that the student will not be able to "hack" out a solution. Analysis and design then become critical. Functional decomposition allows a more regimented top-down design (or bottom-up). It takes a problem and helps break it up into sub-problems. These sub-problems can then be broken down further until the problem space is mapped out into a set of manageable sub-problems. Then, the student formulates a problem specification for each sub-problem. Each sub-problem at the lowest level usually maps to a procedure or function and the problem specification for that sub-problem maps to a comment block for that procedure or function. A flowcharts or pseudo-code is then written for the sub-problem which can map to inline comments for the sub-program. The solution space can be mapped out into skeleton code consisting of only sub-program declarations (including formal parameter declarations and data structures) and comments. Then, if the analysis and design were well thought out, the implementation of the code itself becomes almost trivial.

The problems that the students face within the course are complex enough that a good application of the problem solving methodology is critical. Those students who try to solve the problems without a solid methodology will soon find themselves lost in a sea of poorly designed code. Developing solutions to complex problems require a logical approach to manage all facets of the problem space. The products learned within the course may not be directly applicable to every context. But, each product fills a specific role and serves as stepping stones to completely model the problem space and then transition to the successful development of a model that maps the solution space. As such, each tool can be tailored to fulfill the same role with an entirely new context.

Generalized Problem Solving

The methodology presented within these two courses provides the students with a basis for problem solving that extends beyond computer science. The main objective of teaching problem solving in this manner is that it provides an approach to problem solving and a way of looking at problems; breaking down problems into manageable pieces; and then solving each piece logically, and at the appropriate level of detail. The products are a means to an ends, and products such as a PDL representation of an algorithm will not have much use outside of the context of software design. However, these products model the problem space and the solution space and the relationship between the products help the student to walk through the problem solving process. These products can be tailored to some degree to fit whatever problem the student is trying to solve.

The Problem Specification can be tailored for all problems that can be viewed in the context of black-boxes. Functional decomposition is extremely useful for most any type of complex problem. Flowcharts and PDL algorithms are useful in terms of training the thought process in terms of logical flow and abstraction levels. Implementation is strictly within the context of the problem though knowledge of a programming language can help reinforce the logical thought process of sequential execution.

References

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, Fifth ed. Boston, MA: McGraw-Hill, 2001.

[2] J. G. Brookshear, *Computer Science: An Overview*, Seventh ed. Boston, Massachusetts: Addison-Wesley, 2003.

[3] S. McConnell, *Code Complete: A Pratical Handbook of Software Construction*. Redmond, Washington: Microsoft Press, 1993.

Biography

MAJOR TIMOTHY G. NIX is an officer in the United States Army and an instructor at the United States Military Academy. He has held numerous leadership positions including platoon leader, SFODA Commander, and Company Commander. He has a BS in Computer Science from the United States Air Force Academy and an MCS from Texas A&M University.