Using the Software Development Life Cycle as a Curriculum Design Tool in the Development of a "Companion Course" for Beginning Programmers

Ronald J. Harkins
Miami University
1601 University Blvd.
Hamilton, OH 45011
513-785-3137
harkinrj@muohio.edu

ABSTRACT

The software development lifecycle method has been used widely by software engineers to produce reliable, efficient, and user-friendly software. The lifecycle process solves problems utilizing technology in six distinct steps…Problem Specification, Problem Analysis, Solution Design, Solution Implementation (coding), Solution Testing, and Solution Maintenance. Computer science educators, likewise, have used the lifecycle methodology to promote logical, efficient problem solving, and disciplined programming behaviors in their students. This same six step lifecycle process can be used effectively in solving curricular problems encountered by computer science departments. Specifically, this paper will detail how the lifecycle method was used in solving the problem of helping frustrated, anxious, and unsuccessful students in the early weeks of a first course in computer programming by developing a short, targeted, programming concepts "companion course" for these students. The ensuing content and pedagogical details of this "companion course" will also be reported.

Keywords: CS0, Pre-Programming, Concepts-First Curriculum, Course Development Models

1. INTRODUCTION

Computer science educators have long found the value in having students apply a methodology in writing computer programs to solve problems. The software development lifecycle model is widely popular, both in industry, as well as in the computer programming classroom. This software development lifecycle method involves six phases: Problem Specification, Problem Analysis, Solution Design, Solution Implementation (coding), Program Testing, and Program Maintenance (Koffman, 2002; Wu, 2004). Using this methodology provides a framework in which computer programming students can write software without the stress, time wasting, desperation, and dissatisfaction of experimental or "trial and error" programming (Beck, 2001). Some educators use a problem solving plan related to the software development lifecycle that requires programming students to develop lab reports detailing activities for each step of the plan (and lifecycle). These reports accompany each programming project, and require the students to be more disciplined in their problem solving efforts (Hyde, 1979).

Today's computer science educators need to be dynamic curriculum developers to devise new courses and curricula to meet the rapidly changing needs of both industry and computer science students. Because the

window for this dynamic, responsive curriculum development can be short, such development might also be done in an experimental or "trial and error" style. Consequently, this can result in longer development time, additional curricular revisions, or an inappropriate redesign of the course. This can leave both students and faculty feeling frustrated, overwhelmed and dissatisfied with the process and/or its results. Applying a methodology to course/curriculum development can make the process more efficient, enjoyable and productive. This methodology can involve setting objectives, choosing a context, establishing a feedback process, defining the course infrastructure, and defining the course components (Guzdial, 2005). Likewise, the same software development lifecycle method that is utilized in industry and by computer programming students to solve problems, can also be used as a course/curriculum development model in 'solving' a curriculum problem or issue, and developing a new course within the university structure.

2. METHODOLOGY

This paper will detail how the software development lifecycle method was used to solve a curricular problem whose solution involved the development of a "companion course" for a "first course in computer programming" at the university level. Each step of the life cycle method (Problem Specification, Analysis, Design, Implementation, Testing, and Maintenance) in the development of this new course will be discussed.

Problem Specification

Students enrolled in a "first course in computer programming" at our university were having difficulty very early in these courses, regardless of the programming language used in the course (Java, C++, Visual Basic). Because of significant course content, and the required pace to cover all of the required course topics, students became anxious, dissatisfied, and disinterested. Early withdrawal from these courses became commonplace. Furthermore, the passive nature of some introductory programming courses can also fail to motivate students, turning them away from both the course, as well as the computer science discipline (Thomas, 2002). Indeed, "comfort level," as evidenced by class participation, anxiety while working on assignments, or perceived difficulty completing assignments, was found to be the best predictor of success in a computer science course, followed by mathematics preparedness of the student (Wilson, 2001). The problem of students being unsuccessful, unmotivated, and dissatisfied in the early weeks of their first programming course, and the corresponding enrollment retention problem in these course, required both investigation and a curricular solution.

Problem Analysis

In analyzing the problem of student performance, anxiety and the associated enrollment decrease in the first month of a semester-long "first programming course" at our university, a number of issues and factors were identified. Meetings and conversations with faculty teaching "first courses" in computer programming (C++, Java, and VisualBasic) helped analyze the problem in more detail. Students needed more instruction and practice in problem solving, and associated algorithm development. More mathematical practice was needed. Related data typing and storage topics needed further discussion. These, and other programming-related concepts, such as program translation/ execution, selection and repetition logic, and documentation guidelines, were confusing and somewhat overwhelming for students in their first programming course. Furthermore, instructors of these courses were frustrated in their inability to address these issues significantly for fear of not completing all the required topics in the curriculum for these courses.

The curricular "solution" to this problem that we proposed included the development of a new, one-credit hour, "companion course" to be taken concurrently with a student's first programming course. This new "computer programming concepts" course could also be taken the semester immediately preceding the students' "first programming course," if their schedule prohibited concurrent enrollment in both courses. This new course would not be the "flowcharting course" of 30 years ago that typically accompanied the

first programming course, but would focus on the topics identified above, emphasizing problem solving and algorithm development (Mitchell, 2001). Some institutions incorporate these topics into the first course in programming (perhaps by adding a credit hour) or restructuring a 3-credit introductory "computer programming concepts" course as 2 hours of lecture/discussion and 1 hour of online lab activity (McFarland, 2004). We, however, chose to "factor out" the common curricular problems found in each of the first courses in programming (C++, Java, and VisualBasic) into this new one-hour, 8-week course that would overlap the first 8 weeks of the students' first programming course. Individual computer programming course instructors agreed that adding to the already overwhelming curriculum of their courses was not the preferred solution to this problem. Additionally, exposing students to essential programming-related concepts before introducing them to the intricacies of a high-level programming language can improve the comfort level of the students (DuHadway, 2002), and hopefully decrease their anxiety and increase their satisfaction with computer programming.

An important distinction must be made between this proposed stand-alone "companion course" and the traditional CS0 course taught at many universities (including ours). CS0 courses were intended to provide an overview of the computer science profession, while focusing on programming and applications for both CS majors and non-majors (Cook, 1997). At our university, the CS0 course is a 3-credit hour course in problem solving with VisualBasic.NET. Students majoring in computer science or business enroll in CS1 with Java as their first language, while engineering students use C++ in their first programming course. Consequently, this new one-credit hour "companion course" would have to be "language independent" (utilizing pseudocode throughout), since it would be populated by students using either Java, C++, or Visual Basic in their "first computer programming" course. Thus, as a stand-alone course, not language specific, not covering the computer science profession, and without an online/hands-on computer delivery infrastructure, this course might resemble the "programming concepts" component of a traditional CS0 course, but the complete proposed "companion course" would differ in many respects.

A similar CS0-related course, offered at another university which was non-programming language specific, covered the concepts of functions, procedures, modular program design, abstract data types, and an introduction to object oriented design...all without the "clutter" and "attention" of language syntax (Dierbach, 2005). A study, conducted at this university, found that a "non-specific" programming language approach to their CS0-type course had the potential to better prepare students than an approach involving a preparatory course that used a specific programming language. In a related study, it was found that a programming course used as a first exposure to computer science resulted in a number of overwhelmed, discouraged students, a low rate of successful course completion, and poor retention in successor courses to CS1 (Allan, 1997). This study also found that the CS1 students who first enrolled in their CS0 course performed at a level of a "half-grade" higher (3.2 vs. 2.6) when compared to their counterparts who did not take their CS0 course prior to CS1. Finally, this study found that CS1 students benefited more from a CS0- type "problem solving course" than from a previous, additional stand-alone programming course.

Consequently, we decided to develop a stand-alone, "companion course" for students concurrently enrolled in a first course in computer programming. This new course would fill a knowledge and skill void (especially in problem solving, algorithm development, and program design) that computer programming students seemed to exhibit in the early weeks of their first programming course.

Solution Design

The third step in the software development lifecycle is solution design. Here, it involved designing the content and delivery components for this "companion course" to be taken by students concurrently with (or prior to) their first computer programming course at our university. The new course, entitled "Fundamentals of Computer Program Design" would be a language-independent course emphasizing problem

solving, algorithm development and program design. A set of 9 course objectives was developed, and an accompanying course topic list was written. Course topics included the stored program concept, computer capabilities and limitations, machine cycles, program translation with compilers and interpreters, variables, constants, data typing/conversion, arithmetic/relational/ logical operators, problem solving strategies, design tools (pseudocode, hierarchy charts, etc), program style/documentation, logic associated with sequence, selection, and repetition structures, object oriented vs. procedural paradigms, event-driven environments, debugging strategies, and decoding program error messages.

Although many of these topics are covered in a first programming course, coverage may be limited, inadequate, or seem "rushed" to first-time programmers (especially in the areas of problem solving strategies, design tools, and algorithm development). Indeed, algorithm development, programming style, program debugging and documentation techniques were reported among the ten principles to be incorporated into an introductory programming course (Schneider, 1978). Others found problem solving and computer science principles (data types, operators, logic, algorithms, and control structures) to be invaluable to students in a CS0-type "problem solving course" taken prior to a CS1 course in computer programming (Allan, 1997 ; Cook, 1996).

In the "Solution Design" stage, an Instructor's Guide was developed, to assist faculty in teaching this course. This document included pragmatic, pedagogical suggestions for meeting each of the 9 objectives of the course. Anticipated student questions and problematic areas (with suggested resolutions) were also addressed in this document. A possible textbook (Venit, 2004), was identified for use in the course. However, since this text was not a "perfect match" to our course's objectives and topical content list, an extensive student notepack was written, consisting of a number of "incomplete" pages (problems, algorithms, design tools, etc.), that required the student to complete them during the class session. A pre-programming concepts problem solving course offered by another university used readings, demonstrations, and pencil/paper exercises to successfully meet its course objectives, with positive student learning results (Allan, 1997). Five homework assignments were also developed for our new course. As with another similar course (Goldman, 2004), these assignments consisted of textbook readings and short written exercises. Generic pseudocode (rather than specific programming language syntax) was used in all instructional and student materials for our course because, as stated earlier, students enrolling in this course would be using any of a number of programming languages (C++, Java or Visual Basic) in their complimentary "first computer programming" course. Finally, a set of instructional lecture slides (written in a way to invite student questioning and discussion) were prepared to reflect the course's objectives and topical content list. The instructor's slides, student notepacks, and assignments incorporated textbook references to encourage students to read the textbook as the course progressed.

Solution Implementation

"Fundamentals of Computer Program Design" was offered to a very small number of students during it's first semester. The small number involved might have been a result of inadequate publicity for the course, or students questioning the value of the course in improving their programming capabilities. The 8-week, one-credit hour course was delivered by the course developer (and author of this paper) in a traditional lecture/discussion format. Student participation was encouraged by a number of in-class activities, problem solving exercises, and open ended questioning by the instructor. Real life situations, sometimes using pseudocode, were used to explain programming concepts and structures. For example, when discussing important looping concepts (entry, exit, updating/testing conditions, infinite iterations), situations such as "playing baseball until it is dark" or "playing baseball while it is light" were used, and extended into a discussion of maintaining baseball statistics for a number of innings (to introduce counting loops). This was similar to DuHadway's (2002) example using the situation of "taking bites until your plate is empty or until you are full" in discussing

repetition structures in a pre-programming computer concepts course. Other real-life examples were used to make discussions of selection structures, problem solving, algorithm development, and object oriented concepts more meaningful and relevant to the students. Two examinations and five written assignments comprised the evaluation for the course.

Solution Testing

The course was tested (evaluated) by the instructor in multiple ways. An analysis of the student evaluations for the course was done. Meetings with instructors of the various "first programming" courses were conducted to discuss the performance of their students who had enrolled in the new "Fundamentals of Computer Program Design" course. Self-reflection by the instructor of this new course also contributed to this course evaluation process. Because of the very small enrollment in the "Fundamentals of Computer Program Design" course, any formal statistical analysis of student evaluation data would be unreliable and questionable. Although the pace of the class was manageable, and class attendance was good, final grades for the course were mostly C's. This was due to a number of factors. Students seem to lack the commitment and discipline, probably because they saw this as only a one-credit hour course. Some lacked the logical reasoning abilities so vital to algorithm development and problem solving. Furthermore, some students submitted incomplete assignments that reflected inadequate effort and time, even though students were given a full week to complete them. Nonetheless, students commented on their course evaluations how the course had helped them in their corresponding "first computer programming" course, in which they were also enrolled. Instructors of these courses also confirmed these students' comments, noting that they wished more of their students had enrolled in "Fundamentals of Computer Program Design." Finally, some students questioned the textbook used in the course, noting that it was only used minimally in classroom activities, and its content was somewhat incompatible with the objectives of this course. All of the evaluative feedback would prove invaluable in the modification and maintenance efforts for this new course.

Solution Maintenance

While the potential value of the "Fundamentals of Computer Program Design" companion course in helping "first course" programmers in the early weeks of their programming studies appeared to be evident, some changes to this "solution" were identified to improve it for subsequent offerings. To improve enrollment, each of the "first courses in programming" (C++, Java, and Visual Basic), had notes in the course schedule, advising students to concurrently enroll in the "Fundamentals of Computer Program Design" companion course. Likewise, a note in the schedule for this new "companion course" informed students that "this course should be taken prior to, or concurrent with a first course in computer programming in C++, Java, or Visual Basic." Additionally, faculty in each section of the "first courses" in computer programming described the content and value of the new "companion course" in their first class meeting of the programming courses. As a result, enrollment had improved, but not significantly. Another modification that might increase enrollment in this new course, will occur during the 2005-06 academic year, when this 8-week companion course will delay it's start until 2 weeks into the regular semester. This will allow students in the programming courses, who experience difficulty with the pace or content (especially the early topics of algorithm development, problem solving, and design methods) in the first few weeks, to enroll in the companion course by its new "delayed" start date. To encourage better, and more thoughtful assignment submissions in this companion course, the instructor will discuss and question students about the assignment in the class meeting prior to the assignment due date. Short quizzes might be used to help both students and the instructor to identify content problems in a more timely manner. A different textbook (Messinger, 2005) will be adopted, and better integrated into course lectures, discussion, quizzes, and examination. More object oriented content would be included in the course to meet the needs of students concurrently enrolled in "first courses" in Java and VisualBasic.Net.

Finally, and probably most important, we will continue to monitor "early withdrawal" rates in the first courses in computer programming, and in particular, the performance of students in these courses who were also enrolled in the "Fundamentals of Computer Program Design" companion course.

3. CONCLUSION

The 6-step software development lifecycle methodology is not only a valuable problem solving procedure for software engineers, but also a beneficial process to guide "curricular problem solving" in higher education. The lifecycle method was used to solve the problem of unsuccessful, unsatisfied students (and their associated withdrawal patterns) during the first few weeks of the students' first course in computer programming. The "solution" involved the development and delivery of a "companion course" focused on important computer programming concepts, problem solving and algorithm development. These important topics, covered insufficiently in a "first course" in computer programming, were essential to the students' performance, understanding, and satisfaction in their computer programming efforts. The software development lifecycle paradigm provided a progressive template for devising, implementing, and maintaining a solution to the problem of early withdrawal and undesirable student performance during the early weeks of their first course in programming. The lifecycle approach to curriculum/course problem solving in higher education provides a programmatic, thorough, and reflective technique for curriculum development, especially in dynamic disciplines like computer science, where new technologies present new challenges and new "curricular problems" that need to be solved quickly and efficiently to meet the ever-changing needs of today's students and tomorrow's technological workplace.

REFERENCES

Allan,V.H. and M.V. Kollesar (1997) "Teaching Computer Science: A Problem Solving Approach That Works," ACM SIGCUE Outlook, January. Vol. 25, pp. 2-9.

Buck, D. and D.J. Stucki (2001) "JkarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum," Proceedings of SIGCSE Symposium, pp. 16-20.

Cook, C.R. (1997) "CS0 : Computer Science Orientation Course," Proceedings of SIGCSE Symposium, March, Vol. 29, pp. 87-91.

Cook,C.R. (1996) "A Computer Science Freshman Orientation Course," Proceedings of SIGCSE Symposium, June, Vol. 28, pp. 49-55.

Dierbach, C., B. Taylor , H. Zhou , and I. Zimand (2005) "Experiences With a CS0 Course Targeted For CS1 Success," Proceedings of SIGCSE Symposium, February, Vol. 37, pp. 317-320.

DuHadway L., S. Clyde,. and M. Recker (2002) "A Concepts-First Approach for an Introductory Computer Science Course," Journal of Computing Sciences in Colleges, December, Vol. 18, pp. 6-16.

Goldman, K. (2004) "A Concepts-First Introduction to Computer Science," Proceedings of SIGCSE Symposium, March, Vol. 36, pp. 432-436.

Guzdial, M. and A. Forte (2005) "Design Process for a Non-Majors Computing Course," Proceedings of SIGCSE Symposium, February, Vol. 1, pp. 361-365.

Hyde, D., B. Gay, and D. Utter, (1979) "The Integration of a Problem Solving Process in the First Course," Proceedings of SIGCSE Symposium, January, Vol. 11, pp. 54-59.

Koffman, E. and U. Wolz (2002) "Problem Solving with Java," 2nd ed., Addison Wesley, Boston, MA, pp. 21-24.

McFarland, R. (2004) "Development of a CS0 Course at Western New Mexico University," Journal of Computing Sciences in Colleges, October, Vol. 20, pp. 308-313.

Messinger, L. (2005) "Logic and Design of Computer Programs," Scott Jones, El Granada, CA,.

Mitchell, W. (2001) "Another Look at CS0," Journal of Computing Sciences in Colleges, October, Vol. 17, pp. 194-205.

Schneider, G. (1978) "The Introductory Programming Course in Computer Science—Ten Principles," Proceedings of

SIGCSE Symposium, February, Vol 10., pp. 107-114.

Thomas, L., M. Ratcliffe, J. Woodbury, and E. Jarman, (2002) "Learning Styles and Performance in the Introductory Programming Sequence," Proceedings of SIGCSE Symposium, February, pp. 33-37.

Venit, S. (2004) "Concise Prelude to Programming: Concepts and Design," 2nd ed., Scott Jones, Los Angeles, CA.

Wilson, B.C., and S. Shrock (2001) "Contributing to Success in an Introductory Computer Science Course : A Study of Twelve Factors," Proceedings of SIGCSE Symposium, February, Vol. 33, pp. 184-188.

Wu,C. (2004) "An Introduction to Object-Oriented Programming with Java," 3rd ed., McGraw Hill, New York, NY, pp. 25-26.