Megaprogramming Education

Hisham Haddad and Herbert Tesser
Marshall University
Computer Science Department
Huntington, WV 25755
haddad@acm.org
tesser@marshall.edu

Steven Wartik
Software Productivity Consortium
2214 Rock Hill Road
Herndon, VA 22070
wartik@software.org

Abstract

In the computer science field, educators face several obstacles when attempting to introduce rigorous software engineering concepts and practices into the curriculum. This paper addresses the issue of software engineering education and the role of megaprogramming in introductory courses for high school and college students. We highlight the need for, and the initial effort in megaprogramming education. We provide a brief description of developed materials and a proposed approach to integrate megaprogramming into high school computer science curriculum.

1. Introduction

Since its inception, traditional computer science education has placed heavy emphasis on teaching language syntax and semantics rather than the software engineering process, which is the basis for code development. Educators face several obstacles when attempting to introduce rigorous software engineering concepts and practices into the curriculum. Among them are:

1. the lack of direct experience of the faculty. Too few computer science faculty have had major responsibility of large software system construction, and too few faculty are well versed in software engineering techniques; and

2. many faculty who teach introductory courses are application oriented, rather than tool builders. We suspect that the focus on near term results leaves faculty unconvinced - or even cynical - of the importance of software engineering concepts.

As software development advances, the industrial community has expressed concerns about the quality and readiness of computer science graduates. Their concerns indicate the following weaknesses. Students:

SIGCSE '97 CA, USA

1. lack the knowledge of systematic approach to problems solving; they learn how to write code but not a process by which the code is developed;

2. are not learning the use of latest software engineering techniques and development tools;

3. are not trained to implement solutions for developing large software systems; most classes lack team-oriented projects, and they use simple exercises that do not illustrate modularization, information hiding, and code reuse; and

4. receive software engineering education late in the curriculum, leaving students with the impression that code generation is of paramount importance.

These concerns necessitate changes in computer science education to provide students with the knowledge and skills to effectively and efficiently function in the work place. In addition, there is growing need to accommodate the teaching of software engineering concepts in computer science programs [1,2,3].

The development of large software systems requires the integration of existing systems as components. Megaprogramming is a component-oriented software development approach for building large software systems by integrating existing systems [4,5]. This approach facilitates software engineering environments, software engineering practices, and software reuse technologies and processes [6]. In this paper we address the role of megaprogramming education in preparing future software engineers.

2. The Need for Megaprogramming in High Schools and Colleges

Traditional high school and college introductory computer science courses emphasize a set of topics including:

1. *Programming Languages notation* to help students master the syntax and feel comfortable expressing algorithms using that syntax;

2. *Algorithms* to help students formulate a sequence of steps representing the sequence of activities embodied in the solution of the problem, and use specific algorithms in conjunction with language features and constructs, such as sequencing, conditional execution, and iterations;

3. *Data Structures* to help students learn data structures (such as lists, stacks, queues, and trees) and their possible implementations using records, arrays, and pointers; and

4. *Programming Methodology* to help students learn brief background about specifications, analysis, and design issues. Usually, they learn how to build an algorithm, how to apply top-down decomposition design using abstraction and information hiding, and how to test the program. This practice does not teach students software engineering concepts and practices.

These topics form the basis for introductory courses. They are sound topics. However, the weakness of this traditional approach is that students coming out of the introductory courses have the following perspectives. Students:

1. place great emphasis on the programming language they have learned. As they struggle to learn the syntax, they attach more importance to the syntax than the semantics. By contrast, skilled software developers learn the syntax of a new language in few days;

2. think of a program as algorithm rather than a set of modules;

3. think of software development as coding and testing since less emphasis is placed on other steps of the development cycle; and

4. lack a scientific basis to analyze and design the software they produce. High school and college introductory courses teach little or no science at all. Students may learn how to assess the quality of a program based on some factors, such as execution speed, memory use, user interface friendliness, and reusability. Students learn how to test their programs, but not in a systematic way.

These points should not down play the importance of topics covered in current computer science introductory courses. With these perspectives, students grasp only a small part of what software development is all about, and they require extensive retraining to be successful in the work place. Students who intend to become professional software developers would greatly benefit from a curriculum that teaches more about the industry's software development process. In addition, high school students would also benefit from such curriculum and have better performance when they take college courses. Megaprogramming education is meant to fulfill the need and teach more practical and disciplined approach to software development.

We propose to introduce megaprogramming education to stress and further the teaching of software engineering concepts in introductory computer science courses for both high school and college students. The proposal is motivated by students' need of the knowledge of industrial software development practices and how professional software systems are developed.

3. Objectives of Megaprogramming Education

A frequently heard complaint about computer science education is "We are not preparing students who can recognize and address the software engineering issues facing industry". One reason is that most problems given in introductory courses are toy-like in scope and size. Small projects do not provide adequate preparation simply because large software projects do not scale linearly (either in time or cost) from small projects.

A second, well known, difficulty lies in the fact that most students' software is made for short use - in effect, the only time that the software is executed is when the student conducts testing and when the instructor grades it.

There are several objectives which must be achieved if we are to introduce beginning students to software engineering and reuse concepts:

I. Motivate students to become software engineers who have a better understanding of real issues facing large scale software development.

Software engineers need to understand why megaprogramming concepts have been adopted by major corporations. The importance of these concepts is derived from the requirements of modern software engineering practices. Some laboratory work should focus on software reuse. In addition, students should understand the nature of domain-oriented architectures.

By motivating students to focus on software engineering issues, two important results are achieved: First, we redirect attention from concentrating on language syntax; and second, we prepare students for the work environment of the future where building from reuse will supersede in importance creating new code.

II. Demonstrate that of-the-shelf software can be used to educate students in reuse technology and it needs not be tedious.

Software created with reuse as a design objective will make

use of generic interfaces. These interfaces can be used to automate the process of "connecting" components. Building with software which is capable of adapting to a problem is fun (remember building with Legos).

III. Develop practical curriculum for teaching software processes which reflects the success of existing hands-on high school curricula.

The foundation of a process approach lies in examining how the process spans the business product line. The objective is to teach students that requirements for reuse are not based upon a specific application but must address the central ideas which are the common basis for the product line. In the future, software engineers will design systems much the same way corporations build product lines. We see the introduction of domain engineering as central to the software product line approach (megaprogramming).

4. Megaprogramming Education: The Approach to Software Development

Megaprogramming is a component-oriented approach for developing large software systems. It is built on the process of integrating individual systems (components) that define the functionalities of the overall system. This approach utilizes technologies and processes including software engineering, software reuse, software architectures, domains, and interfacing. With megaprogramming, software engineers build an infrastructure that can be used to plan for, design, develop, manage, and maintain a family of software systems. As a conceptual structure for component integration, megaprogramming can be scaled up to produce complex systems or scaled down to produce simple systems with few components. The overall motivation to megaprogramming is to reduce the cost and time of software development and increase software quality [7,8].

In the megaprogramming approach, reuse plays a central role. That role is the development of software components that are sufficiently general for reuse in a wide range of systems, sharing similar functionalities and architectures. Such components allow a wide range of information, not just source code, to be reused in developing new systems [9]. With this practice, a collection (library) of reusable components based on architectural standards of domain-specific systems may result. Such components increase reusability, improves the quality of developed systems, and the productivity of the development process.

While megaprogramming is the process of building software systems by selecting and adapting reusable components, software engineering is the management, engineering, and communication activities required to develop a software system [10]. Megaprogramming requires studying existing systems that have common features (usually called domain of applications) to identify reusable characteristics, develop reusable software components, and provide guide lines to incorporate reusable components into newly developed systems in the domain.

Traditionally, students think of each program as a self-contained independent program. With megaprogramming education, students learn to think of each program they produce as a component of a bigger system that does not necessarily exist, but may evolve in the future. Thus, giving students a broad view of the practical aspects of the industry's software development process, and establishing team-oriented based thinking and experience.

With megaprogramming education, students also learn that software reuse is not limited to code reuse. It is the process of re-applying the knowledge and experience gained during the development of a system to the development of a new system. Reused knowledge and experience may include domain knowledge, design knowledge, requirement specification, procedures, code, documentation, domains, implementation issues, and maintenance issues.

5. Initial Effort in Megaprogramming Education

In 1993, DoD's Defense Advanced Research Projects Agency (DARPA) became interested in understanding the effects of introducing reuse topics as early as possible in the computing curriculum. DARPA tasked the Software Productivity Consortium (SPC) to develop curricula and supporting materials that introduce megaprogramming in high schools and introductory college computer science courses. SPC developed several courses and a curriculum that form the background for much of the material in this paper. A brief description of the developed products is as follows:

1. A one-to-two week "Overview of Megaprogramming" course [2,11]. This course, targeted to high school students, presents broad concepts of megaprogramming. It challenges students to forget their traditional notions of programming and instead concentrate on:

a) Requirements Elicitation. High school students (and many college students) often naively assume there will always be a teacher to precisely state the problem they are to solve. For many, this course was their first realization that one day they must formulate problems themselves, and it comes as quite a shock;

b) Software Process. Many instructors that teach design principles are frustrated by students who can parrot the concepts but jump immediately to

coding during their assignments. The laboratory in this course forces students to follow a well-defined process by requiring them to provide requirements and design information (and in fact, they perform no coding -- it all happens through reuse); and

c) Quantitative Analysis. The laboratory requires students to evaluate the quality of their work using analytical models. These models are applied during requirements and design as well as during verification.

2. A one-week "Software Engineering Using Ada" course [12]. Whereas the overview course is a broad overview of megaprogramming topics, this course explores in more depth notions of design styles that lead to reuse. The goal in creating the course was to evaluate whether students first encountering software could grasp the importance of such topics as modularization and abstraction, and how they promote reuse.

3. A megaprogramming-based undergraduate curriculum that shows how megaprogramming topics can be integrated into a computing curriculum starting in the very first course [13].

4. A one-semester course introducing software development to non-majors. The course teaches traditional programming topics, but also emphasizes teamwork, project planning, and other important aspects of software development.

The first two courses have been taught at ten high schools and have received favorable reactions from both teachers and students. The one-semester course has become part of James Madison University's Integrated College of Science and Technology curriculum. A key to the success of the course and curriculum development was SPC's involvement with high school and university instructors, who established the constraints and provided expert guidance.

6. Integration of Megaprogramming into High School Computer Science Curriculum

One problem facing undergraduate computer science education is the insufficient preparedness of incoming high school students, and the lack of computer science background. A major contributing factor to this phenomenon is that advances in computer science technology are adding more teaching materials and topics to the field; while high school computer science education has not been advanced accordingly. Thus, creating a gap between current high school educational offerings and the proper preparation for college education.

With megaprogramming education, we propose an approach to advance high school computer science education by propagating introductory college-level computer science topics to the high school level. The theme of this approach is to introduce high school students to the software engineering technology and its practices (including software reuse), and the role of software engineering in todays software development industry.

Proposed approach:

Teach high school students and lower level undergraduates concepts and technologies which address the key issues of modern software engineering. The following course sequence is intended to introduce the concepts of large scale software development into introductory courses, and establish a broad background of what software development is all about.

Proposed Course:

In addition to existing high school computer science courses, we propose the following courses be integrated into the high school computer science curriculum:

Megaprogramming I: First Semester
covers:

1. an overview of the software engineering process and how it is applied to the development of individual systems;
2. an overview of the software development life cycle used in software industry and how professional software systems are developed; and
3. an overview of software reuse and its role in today's software development process.

Megaprogramming II: Second Semester
covers:

1. an introduction to the product-line approach to software development;
2. an overview of the domain concept, domains of applications, and the effect of domains on software development;
3. an overview of domain engineering (domain analysis and domain design) and how it is applied to the entire group of systems in the domain; and
4. an overview of building reusable software components using domain engineering and reuse.

Advantages:

With this approach, high school students will be exposed to details of large scale programming, reuse, and domain engineering along with other appropriate topics.

In general, knowledge of these topics will help improve students' background, provide a smooth transition from high school to college, and better prepare high school graduates to the college environment. However, we envision the following specific advantages. Students:

1. learn to see software development as an exercise in science and engineering, and be able to write better software systems;

2. get the opportunity to work on more complex systems and gain team-oriented experience. With the knowledge of reuse, students can reuse existing software systems that perform complex functions in building more complex systems;

3. learn more about the industry's professional software development process and current technologies and practices used in the industry. Thus, students become better prepared for their careers;

4. will change their perspective toward software development since megaprogramming encompasses all aspects of software development. In general, current teaching methods are lacking a complete approach to provide students full understanding of the importance of covered topics; and

5. will speed up their academic performance and progress during college years.

7. Conclusion

Current computer science education methods place more emphasis on teaching language syntax and semantics rather than software engineering concepts. As software development advances, the academic community is obligated to provide computer science students the proper knowledge of industrial software development methods and practices. Megaprogramming education is design to further the teaching of software engineering concepts and reuse practices, and to provide practical and disciplined approach to software development.

In this paper we highlighted an effort in megaprogramming education, initiated by DARPA and carried out by SPC which developed several courses and a curriculum. The courses were taught in several high schools. The paper also highlighted the importance of teaching software engineering and reuse concepts to beginning students, and proposed an approach to build on and expand DARPA and SPC's effort. We outlined two courses to help eliminate the gap between current high school educational offerings and the proper preparation for college education. The proposed course, Megaprogramming I and II, cover the basic issues of modern software engineering technologies and practices used in todays software development industry. An initial attempt to introduce megaprogramming education in high schools has been conducted by SPC [11].

8. References

1. J. Cohoon and et al. "Software Engineering Beginning in the First Computer Science Course", *Proceedings of the Software Engineering Institute 7th Conference on Software Engineering Education*, NY, (1994).
2. Mary Eward and Steven Wartik, "Introducing Megapro gramming at High School and Undergraduate Levels", *Proceedings of the Software Engineering Institute 7th Conference on Software Engineering Education*, NY, (1994).
3. R. M. Snyder, "Teaching Software Engineering Principals in an Introductory Programming Course", *Journal of Computing in Small Colleges, 10(3)* (Jan. 1995).
4. Barry Boehm and William Scherlis, "Megaprogramming", *Proceedings of the DARPA Software Technology Conference*, Los Angeles, CA, (1992).
5. "On the Way to Megaprogramming", *Technical Report, STARTS Technology Center*, Arlington, Virginia, 1992.
6. Gio Wiederhold, Peter Wegner, and Stefano Ceri. "Toward Megaprogramming", *Communications of the ACM, 35(11)*, (November 1992).
7. Dave Ceely, *Impact of Megaprogramming*, Position Paper for TRIAda'93.
8. Teri Payton, *Megaprogramming - Facilitates a Transition Toward Product-Line Software*, Position Paper for TRIAda'93.
9. Ted Biggerstaff and Alan Perlis, *Software Reusability: Concepts and Models*, ACM Press, Frontier Series, Vol.I, Addison-Wesley, Reading, MA, 1989.
10. Bill Hodges, *Impact of Megaprogramming*, Position Paper for TRIAda'93.
11. "Teacher Notes for Overview of Megaprogramming Course", *SPC-94044-CMC, Version 01.00, Software Productivity Consortium*, (September 1994).
12. "Software Engineering Using Ada Course", *SPC -94094-CMC, Version 01.00, Software Productivity Consortium*, (April 1995).
13. Steven Wartik and et al., "Megaprogramming in the Software Engineering Curriculum", *Proceedings of the 4th Annual Workshop on Software Reuse Education and Training*, Morgantown, WV, (August 1995).