Integrating Large-Scale Group Projects and Software Engineering Approaches for Early Computer Science Courses

M. Brian Blake, *Senior Member, IEEE*

***Abstract*—The utilization of large-scale group projects in early computer science courses has been readily accepted in academia. In these types of projects, students are given a specific portion of a large programming problem to design and develop. Ultimately, the consolidation of all of the independent student projects integrates to form the solution for the large-scale project. Although many studies report on the experience of executing a semester-long course of this nature, course experience at Georgetown University, Washington, DC, shows the benefits of embedding a large-scale project that comprises just a segment of the course (three to four weeks). The success of these types of courses requires an effective process for creating the specific large-scale project. In this paper, an effective process for large-scale group project course development is applied to the second computer science course at Georgetown University.**

***Index Terms*—Collaboration skills, computer science II, object-oriented design, programming, software engineering education and training.**

I. Introduction

An INTRODUCTION of the most effective combination of training techniques is essential for students who are programming for the first time. This initial stage enables students to learn the foundation of their programming skills and develop individual practices that will remain with them throughout their careers as computer professionals. Because students are pliable at this initial stage, they need to be immersed in skills that expand further than programming syntax. Students who understand the big picture at an early stage tend to adopt advanced programming techniques more seamlessly later in the curriculum [1], [2]. In addition, understanding how software can be modularized encourages the development of more robust and reliable software [3]. Since programming and debugging skills are so important, introducing these higher order software engineering skills should not decrease the amount of programming reinforcement training offered in the early computer science courses. In addition to the work at Georgetown University, Washington, DC, [4], many researchers [5]–[9] promote semester-long, large-scale programming courses. These courses provide students with an understanding of low-level developmental tradeoffs, such as choosing the most efficient data structure or the use of recursion versus iteration. However, such courses may be more effective if offered later in the computer science curriculum. Introducing these types of courses later in the curriculum would prevent problems [3] such as the following:

- students understanding small portions of the big problem;
- advanced students hindering the development of students who perhaps need more programming experience;
- a decrease in the students' exposure to the array of concepts, as programming tends to be more specific to a particular technique in each part of the large-scale project.

These problems may be alleviated if group projects are integrated for no more than one third of the course. Although this type of approach requires additional effort by the instructor to scope the large-scale project appropriately, the remaining two thirds of the course can be used to provide the reinforcement programming training that students need at this stage. This approach can be separated into several phases within the three-to-four-week classwide project that promotes design, development, and utilization of advanced techniques (such as programming techniques with abstract data types). Another important consideration is the effective placement of the special project as it relates to the typical course topics presented in early computer science courses.

In the following section, the computer science curriculum at Georgetown University is described in detail. A discussion of earlier shortcomings in the curriculum plan is presented and goals are set to address them. In Section IV, the general structure of the large-scale project and a procedure for integrating it into the second introductory computer science course is detailed. In Section V, a description of the large-scale project that was integrated into the Spring 2003 computer science course at Georgetown University is given. In the remaining sections, this approach is compared with related projects, and the benefits and drawbacks based on quantitative and anecdotal results are evaluated.

II. Background: The Computer Science Curriculum at Georgetown University

The computer science curriculum at Georgetown University is similar to other curricula in computer science departments

Manuscript received July 30, 2003; revised October 1, 2003. This work was supported in part by The Rational Corporation under a grant of software licenses. The author's affiliation with The MITRE Corporation is provided for identification purposes only and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions, or viewpoints expressed by the author.

The author is with Georgetown University, Washington, DC 20057 USA. He is also with the Center for Advanced Aviation System Development, The MITRE Corporation, McLean, VA 22102 USA (e-mail: blakeb@mitre.org).

Digital Object Identifier 10.1109/TE.2004.832875

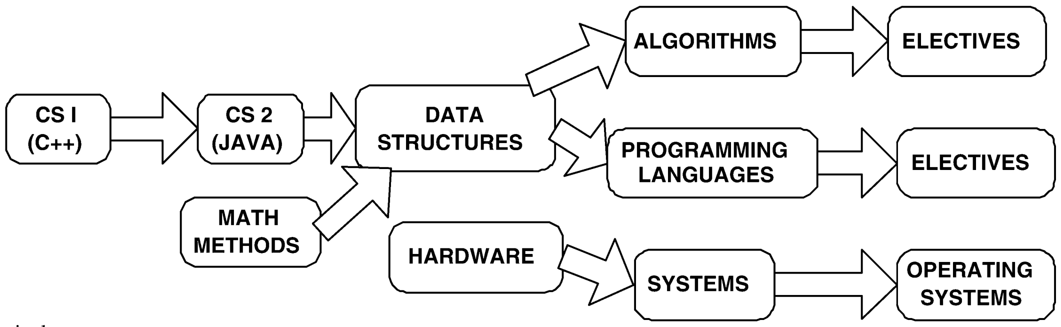

Fig. 1. Initial curriculum.

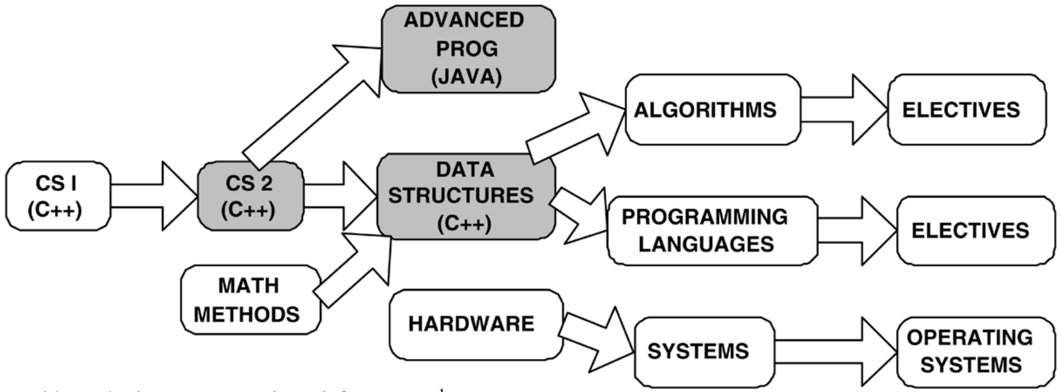

Fig. 2. Curriculum with emphasis on programming reinforcements.[1]

of similar stature. There are some variations in the depth of coverage in some areas; however, most programs cover similar topics. With significance to the Computer Science Department at Georgetown University, no graduate program exists. An emphasis on undergraduate education is a focus of the department and of the college. The required curriculum consists of introductory programming courses, such as Computer Science I and II (CS1 and CS2). The required coursework also includes foundational courses, such as Math Methods, Data Structures, Algorithms, Programming Languages, Hardware, Systems, and Operating Systems. Courses such as Artificial Intelligence, Software Engineering, Networking, Machine Learning, Natural Language Processing, Information Assurance, and Electronic Commerce are offered as a choice of electives (four electives are required). Figs. 1 and 2 illustrate two prerequisite structures for the aforementioned required coursework. The initial structure (Fig. 1) represents the curriculum prior to 2001, and the 2002 curriculum (Fig. 2) shows a new, more focused plan.

In the initial curriculum, CS1 and CS2 were split into two different programming languages (C++ and Java). As a result, significant time was spent in each class introducing each programming language. The two courses lacked cohesiveness, and the introductory nature of the courses limited the depth of the programming training. In addition, the Data Structures course was not taught as a programming course. Moreover, courses such as Algorithms, Programming Languages, and Systems were not intensive programming courses with respect to students solving complex problems by conceptualizing programming solutions. Because of the dynamic nature of computer science technologies, the faculty perceived that the students would be better prepared if more programming training was provided for the two-year period central to the curriculum. This perception was reinforced based on the large amount of programming background required in later programming-intensive courses, such as Operating Systems and a majority of the electives. More reinforcement training in programming would better prepare the students to handle the advanced problems presented in the more focused elective subjects.

The faculty curriculum committee decided to make major changes to the curriculum and the underlying courses to help alleviate these issues and promote reinforcement training in programming (shown in Fig. 2). Several major changes were made that are highlighted in Fig. 2. One change was the use of one language (C++) that would be consistent across CS1, CS2, and Data Structures. Another change was encouraging the instructors teaching the Data Structure course to include more programming-intensive exercises. In addition, an advanced programming course that uses Java was introduced which would give students exposure to the Java programming language while introducing advanced programming approaches to networking, web development, database connectivity, and graphical user interfaces. Finally, the committee encouraged better connection between CS1, CS2, and Data Structures, in addition to an emphasis on problem solving with programming in CS2.

III. The Connection Between CS1, CS2, and Data Structures Courses

The focus of this paper is on changes to CS2. However, the connection of CS1, CS2, and Data Structure courses provides the additional motivation with respect to the importance of these changes to CS2. The first action for making these three courses more cohesive was to align the underlying subjects taught in

[1]Figs. 1 and 2 are reproductions of illustrations developed by the departmental curriculum committee of Georgetown University (the faculty members are listed in the Acknowledgment).

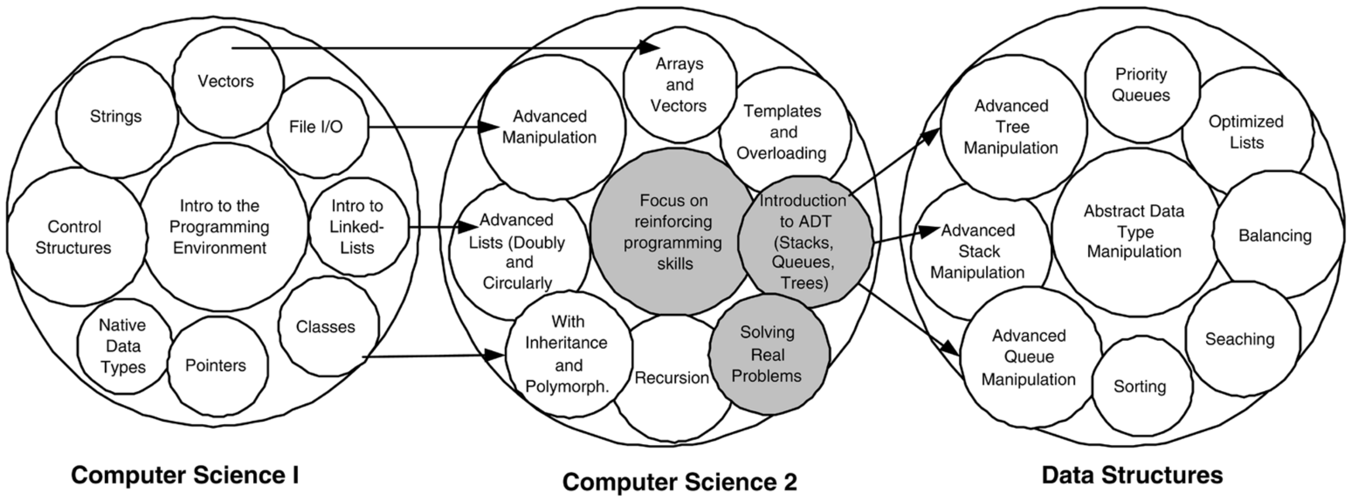
Strings
Vectors
File I/O
Control Structures
Intro to the Programming Environment
Intro to Linked-Lists
Native Data Types
Pointers
Classes
Computer Science I
Arrays and Vectors
Advanced Manipulation
Templates and Overloading
Focus on reinforcing programming skills
Introduction to ADT (Stacks, Queues, Trees)
Advanced Lists (Doubly and Circularly
With Inheritance and Polymorph.
Recursion
Solving Real Problems
Computer Science 2
Priority Queues
Advanced Tree Manipulation
Optimized Lists
Abstract Data Type Manipulation
Advanced Stack Manipulation
Balancing
Advanced Queue Manipulation
Sorting
Seaching
Data Structures

Fig. 3. The connections between CS1, CS2, and Data Structures courses.

each course. The CS1 course serves as an introduction to programming and programming syntax, while the Data Structures course provides training for the advanced manipulation of abstract data types (ADTs). Therefore, the ability to connect the three courses lies in changes that must occur in the CS2 course. Consequently, several new goals for the CS2 course were created, as follows:

1) reinforce and expand on CS1 topics;
2) include an introduction to Data Structures;
3) promote the use of software design to solve problems.

In reinforcing subjects from CS1, the goal in CS2 is to review subjects, such as file input and output routines, vectors, classes, and linked lists, while introducing other advanced approaches (e.g., inheritance, polymorphism, and doubly/circularly linked lists).

Although abstract data types are not the focus of CS2, the introduction to the data types in the context of solving real problems completes the mapping between CS1, CS2, and Data Structures courses. A sampling of relevant subjects in these courses and their mappings are illustrated in Fig. 3.

IV. Developing a Course With a Short-Term Classwide Group Project

In general, instructors of engineering and science courses tend to build the basic skills of students for a significant portion of their courses. The acquisition of these basic skills helps to build a cumulative aptitude in the students. For the computer science domain, basic programming techniques can be used to develop the students' ability to apply the combination of multiple techniques to unique problems. Likewise, in other domains, such as circuit evaluation in electrical engineering courses, students learn to evaluate circuits using techniques such as loop, mesh, and node before being given an advanced problem that relies on a combination of techniques.

A. The General Course Development Approach

The objective in these introductory courses is to develop both training for basic skills and the use of higher level composite skills.

In developing such a course, several steps have been determined.

1) *Create a list of basic competencies for the course*—These basic competencies tend to be consistent with course objectives.
2) *Schedule individual portions of the course to train students in each competency (such as weekly lectures and problem sets)*—This scheduling constitutes the basic training of the students, which should precede later training in higher level skills which utilizes the combination of basic skills.
3) *Design a set of independent modules that students must perform that require the higher level combination of competencies*—These modules should require students to leverage their basic skills. The focus of these modules can vary, but the difficulty should be consistent.
4) *Formulate a large-scale project through the combination of the set of modules (in some cases, modules must be duplicated).*

The final two steps in developing this type of course are the most difficult. Developing the classwide project requires extensive planning by the instructor. The problem space should consist of a number of modules or interrelated *black boxes*. Students should be able to implement each module or black box with a variation, extension, or combination of basic skills. For the CS2 course, such a problem may require the utilization of an abstract data structure or some other advanced programming technique. Once the technique is determined, the instructor should estimate the amount of time needed to implement specific modules. To the greatest extent possible, the modules should be evenly distributed among the class. In classes with a large number of students, one may find necessary the assignment of multiple students to a module. The project is most interesting to students if it is associated with a real-world application.

Alternatively, multiple groups of students could be assigned the same large-scale project. This approach would be optimal when scaling the problem to larger class sizes. One positive effect of having multiple concurrent groups is that a student or subteam on one project team can be used to supplement the shortcomings of the *mirror* student or subteam of the other

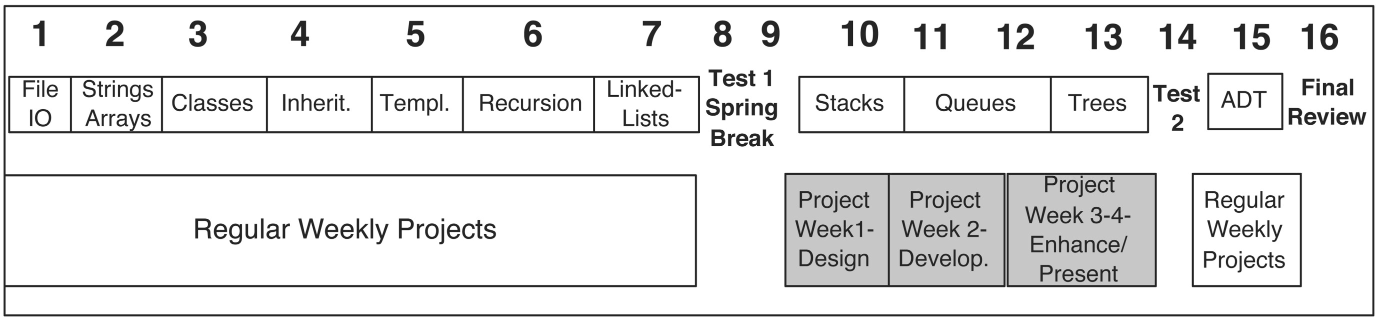

Fig. 4. Placement of the three-to-four-week large-scale project in Computer Science II.

project team. This possibility is particularly important to the instructor who would otherwise personally have to supplement this gap with his or her own solution. This situation also allows the instructor to evaluate multiple solutions for a particular module and to demonstrate to the students the tradeoff in design and implementation.

B. Specifying the Large-Scale Project for the CS2 Course

A major goal of the CS2 course is to introduce new programming techniques continually while encouraging students to think independently in solving unique problems. This approach utilizes weekly programming exercises. A new programming problem is introduced weekly that allows the students to get hands-on knowledge on the techniques presented. However, the brief and well-defined nature of these projects presents problems, as discussed in Section I. In order to see the usefulness of the advanced programming techniques, the students need to conceptualize solutions to problems that are not so well defined. In addition, students should get an introduction to the collaboration necessary in large-group programming environments. Immersion in such an environment greatly benefits their appreciation for interface design and software reliability. As mentioned previously, other academicians promote a semester-long course to incorporate the benefits of the group programming environments. In this context, the semester-long approach would interfere with the reinforcement training needed in CS2. Therefore, a project that represents 30% of the programming assignments is preferable.

A normal semester consists of 16 weeks. There are two weeks without projects based on the first and second exam weeks and one additional nonproject week prior to final exams. Typically, two additional weeks are without projects because of the placement of holidays in the semester. Therefore, an instructor can typically assign 11 to 12 weekly projects per semester. In this approach, large-scale projects are used to substitute three to four weeks of weekly projects. Fig. 4 illustrates the proposed time line for weekly projects. In the first row, there is a list of the total number of weeks for the semester. The second row shows a sample list of subjects covered in the first computer science course. In the first seven weeks and in the last one to two weeks, regular weekly projects are assigned; and in weeks 10 to 13, a large-scale project is incorporated.

The large-scale project should optimally be assigned concurrently with the introduction of data structures. The goal is for students to begin thinking about solutions to the unique projects at the same time that they receive lectures on abstract data-type concepts and usage. The project consists of a week for *design*, a week for *development*, and a week or two for *enhancements.* In the first week, students are presented with the full software problem, which is composed of parts that students can achieve independently or in small groups. The instructor explains each independent part as a *black box* with specific inputs and outputs. Students are required in the first week to provide a design solution for mechanisms internal to their specific black box. One group is assigned the *central* black box that acts as the control for the program, thus integrating the entire software solution. The responsibility of each student who represents a black box is to submit a software design visually and with anticipated function prototypes. In addition, at the end of the first week, the students must present their designs during class. Copies of each of the designs are given to the group responsible for the central black box. At the end of the second week, students must submit their portion of the project with a test driver and stub programs. The group with the central portion of the project integrates the software in the third week, when all students are required to enhance their designs based on the latest lectures of data structures. The responsibility of the instructor is to create modular *components* of the classwide project that are applicable to the use of data structures, such as queues, stacks, and trees.

V. A Project Example: Creating an Agent-Based Workflow Simulation

This approach is best understood when illustrated with a concrete example. A large-scale project was integrated into the Spring 2003 course of Computer Science II at Georgetown University. The project was introduced using a time line similar to the time frame proposed in Section IV-B. The large-scale project was substituted for the sixth, seventh, and eighth projects of a ten-project semester. The project portion of the course was conducted during three 75-minute classes and represented 20% of students' course grade. Students indicated that they spent from six to ten hours per week working on the project, depending on their skill level.

The problem dealt with the creation of a simulation application. The students were asked to model intelligent software robots or *agents*. These software agents would be used to manage Internet-based travel reservations. Thus, intelligent software agents would fulfill the orders for making travel arrangements of users that schedule travel over the Internet. Students had to determine the efficiency of the intelligent agents

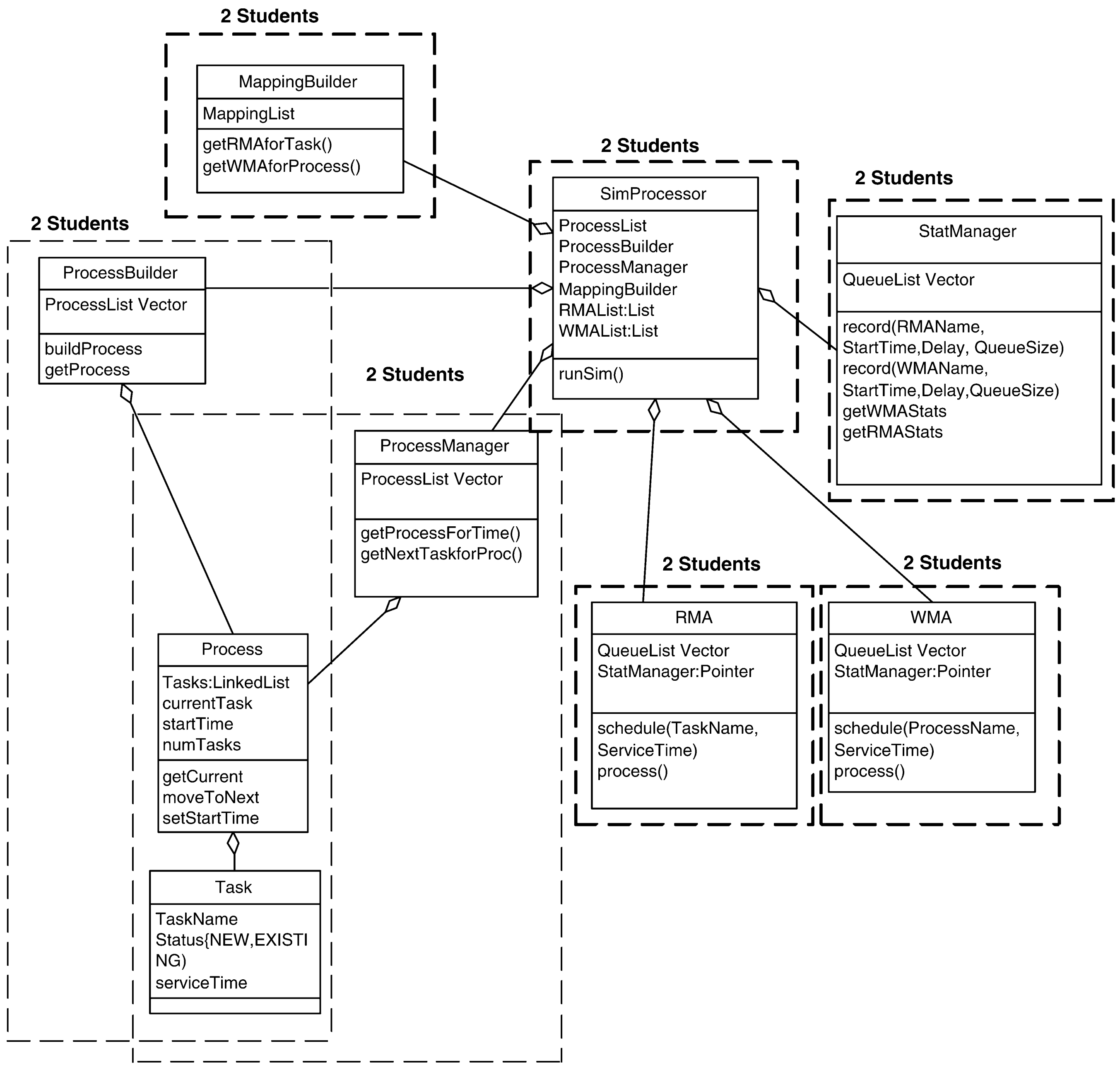

Fig. 5. Agent-based workflow project: an example of a classwide project.

to handle increasing loads of travel requests. The class first had to develop a simulation that would allow the entry of traffic information in the form of text files. This traffic information would be used to task multiple software objects, acting as software agents. These software objects (modeled as agents) would maintain a list of scheduled tasks. Each time a new task was scheduled, the object would record numerical measures of the service time delay and the operational delay based on its current queue size (outstanding tasks).

A. Assigning the Agent-Based Workflow Simulation Project (Week 1)

The instructor decided that the best way to visualize the project was to create a class diagram [10]. By decomposing the class diagram into lower level groups of classes, one class or group of classes could represent an individual black box as described in Section IV-B. In addition, the instructor could specify interface functions or methods as opposed to specifying input and output data. The application was composed of nine classes: ProcessBuilder, ProcessManager, Process, Task, MappingBuilder, SimProcessor, StatManager, Workflow Manager Agent (WMA), and Role Manager Agent (RMA). Participation in the classwide project was on a voluntary basis, but most of the students decided to participate. Therefore, 14 of 16 students in the class participated, and the instructor assigned two students per module. The class diagram for the agent-based workflow simulation is illustrated in Fig. 5.

In Fig. 5, the separation of subprojects (modules) is illustrated by dotted lines, and the functions illustrated are the interface methods and data structures agreed upon by the students and

Start Time, Workflow_Process_Name, {New,Existing}, (TaskName, Expected Service Time, {New, Existing})$_{1\ldots n}$

Fig. 6. Record of business process information.

```
Initialize TempProcessList and TempTaskList

Main Loop (Time)
        {
                Loop through all WMAs and Insert time (Time)
                        Remove processes from WMAs based on TempProcessList
                For Current Time, get List of New Processes (Vector = ProcessMgr.getProcessForTime() )
                        For Each New Process
                                Look for corresponding WMA (MapBuilder.getWMAforProcess)
                                        Schedule New Process in WMA (with Time estimate)
                                        Get First Task from Process
                                                Insert into TempTaskList

                Loop through all RMAs and insert Current Time (Time)
                        If task has completed
                                Add next task to TempTaskList
                                        If last task in process, add Process to TempProcessList

                Loop though TempTaskList
                        Look up pertinent RMA for Task (MappingBuilder)
                        Schedule Task with RMA (with Expected Service Time)
        }
```

Fig. 7. Main algorithm of the simulation application to be performed by the SimProcessor.

instructor after the first project week. The task of the ProcessBuilder module is to collect a list of processes from a text file (traffic file) and to construct, in memory, containers for storing the information. The instructor suggested the use of Process and Task classes to store the information. The information is composed of the start time of the process, the process name, new/existing status, a sequential list of tasks, with service time and new/existing status (Fig. 6). The students involved with this module were tasked to design software that could retrieve the information from a text file in a format of their choice. The module also had to provide methods to access the information once stored in memory.

The requirements of the ProcessManager module were to access the traffic information and cycle through the tasks at simulation time. Thus, the ProcessBuilder and ProcessManager groups were required to work together. In fact, the Process and Task classes were a shared component of both groups. The team that built the MappingBuilder was responsible for reading a text file that contained information about the relation between the software objects and the processes and tasks. This class was used to assure that tasks were distributed to the correct software objects at run time.

The RMA was a software object (agent emulator) representing an individual travel reservation task, such as car rental, hotel reservation, or airline ticket purchases. Tasks from the traffic file were assigned to RMAs using the mappings maintained by the MappingBuilder. The RMA had a scheduling method that accepted new assignments and placed them in a task list.

The WMA object emulated a process-level agent that kept track of the entire travel reservation process (composed of multiple services). Like the RMA, processes from the process file could be assigned to WMAs using the bindings maintained by the MappingBuilder. In addition, the WMA maintained a list of active processes. Both RMAs and WMAs had the requirement to record a delay when new tasks and processes, respectively, were assigned.

A StatManager was passed to the RMAs and WMAs to record statistics. In addition, this class was responsible for showing a display of the statistics in some format. The statistics would be in the form of total delay per operation time for all RMAs, delay per operation time for all WMAs, or total system delay per process at a given time.

Finally, the SimProcessor was the *central black box* that controls the entire simulation. The ambitious students who agreed to be assigned this part had the task of integrating all parts in addition to working on the main algorithmic process. With the initial suggestion of the instructor, the students worked to revise an algorithm that would incorporate all classes to manage the simulation.

The algorithm is shown in Fig. 7. The focus of this paper is not to describe the inner workings of this algorithm but to show a method of describing the operation of the main algorithm using a pseudocode approach. The instructor also introduced sequence diagrams from the Unified Modeling Language as a method to demonstrate the operation, but the students seem to adopt the pseudocode representations more easily.

Admittedly, the resulting application represents a basic approach to simulation. The major shortcoming of the simulation design is that service times and assigned service completion times remain static once the software objects assign them. In a true simulation, service time and processing times are dynamic as the simulation executes. However, despite the static nature of the processing times, this simulation application has been effective in evaluating alternative types of traffic. In fact, this simulation has been useful in evaluating agent interaction protocols on various streams of traffic as recorded in other literature [11].

B. Implementing the Project (Weeks 2 and 3)

During the first week, the students were somewhat overwhelmed with the scope of the total project. In addition, they realized that it would be difficult for one person to complete the entire project in the time allotted. As the students began to understand their individual responsibilities, their enthusiasm for successfully completing the project began to increase. Students were conscientious about appropriately developing their particular piece of the project. During the second week of the project, students presented their project designs and submitted their designs both to the instructor and to the group integrating the entire project.

In the second week, students began to think about what programming techniques could be used to implement their classes. It was the intention of the instructor to allow the students to program the classes in any manner necessary, but then to suggest using data structures in the third week. Some of the names used in Fig. 6 suggest the use of certain data structures for particular modules. Although these were not available during the first two weeks of the class, in this offering of the course, the students began to suggest the use of abstract data structures, particularly in the development of the ProcessManager, WMA, RMA, and MappingBuilder. This outcome was a positive effect of holding this special project concurrently with the lectures on abstract data types. Some groups were savvy enough to use the data structures in the second week and, on the third week, were assigned other extension tasks to enhance their modules. Other students were assigned enhancements to their module using a specific data structure. The fourth week was used for students to present their completed modules and to demonstrate the completed classwide application.

VI. Comparison to Related Work

As previously mentioned, several studies have been made into semester-long courses of this sort. Bareiss [1], Turner [2], Rebelsky and Flynt [3], and Adams [9] endorse the use of large-scale, course-long projects to enhance the students' ability to discover developmental tradeoffs. Other literature by Krone, *et al.* [7], and the earlier work of the author [4] show how the developmental training is greatly enhanced when combined with formal training in coordination and collaboration. In addition, other literature [12]–[14] concentrates on the general benefits of collaborative approaches to teaching in engineering and science education.

The approaches presented in this paper attempt to leverage all three approaches. Students learn to work collaboratively as a large developmental team. The major difference is that this approach utilizes a shortened large-scale project that represents just a portion of the course. There are other departments that adopt similar approaches of short-term, large-scale projects. However, this approach is one of few approaches in publication that presents a process to develop and incorporate this type of course. Decreasing this project to a smaller portion of the semester allows students to have regular individual programming exercises during the semester to reinforce developmental skills while also having the experience of large-scale software development (shown to be valuable in the aforementioned related literature).

In addition, in this approach, the early introduction of object-oriented modeling techniques is used to explain the group projects. This introduction has been helpful in later courses [1] although additional training may be needed before the first-year students can totally understand the modeling techniques. Of all the design models introduced, the students, at this level, best understood the structural diagrams (class diagrams).

This approach also supports less group work at the module level. In the curriculum at Georgetown University, the instructor found that, at this early stage, students should benefit from participating in individual programming projects. However, there are barriers to creating the individual programming components. Even in a class of 16 students, a problem that decomposes easily into 16 parts is difficult to develop. In the case of the course presented in Section V, two students were assigned per module. The goal should be to minimize the number of students in each group when individual work is not possible.

VII. Evaluation of This Approach

In evaluating the impacts of this approach, both quantitative impacts (comparison to an earlier course) and anecdotal responses from the students were considered. In the following sections, the course is evaluated using both methods, including a discussion of the results.

A. Quantitative Evaluation Based on Earlier Courses

This paper describes the first offering (Spring 2003) of this type of course that combines weekly projects with a short-term, large-scale project. However, there was an earlier course using a semester-long project taught by the same instructor in the previous year (Spring 2002). The Spring 2002 course contained a large-scale project spread over the entire semester. Each module during the semester was performed independently by the students and represented a building block to the final large-scale application. Consequently, each student individually developed the large-scale project over the length of the semester.

1) Comparing the Two CS2 Courses: The major difference between the Spring 2002 and Spring 2003 courses was the lower level of collaboration among the students in the instance of Spring 2002. In addition, the Spring 2002 instructor did not develop a problem that the students could develop in its entirety. Instead, each week, students were given assignments to modify pre-existing modules with techniques learned that week. This type of training is common in courses taught at Georgetown University, but developing modules without pre-existing "starter" modules was discovered to be more valuable to the students.

A common thread of both courses was the format of the final exam. Both final exams consisted of programming problems. In one part of the exam, students were given a specific problem and were required to demonstrate a programming solution using a specified data structure. Another portion of the final exam was an abstract problem that allowed the students to showcase their

TABLE I
PERCENTAGE OF STUDENTS EXCELLING AT SPECIFIC COURSE COMPETENCIES

Criteria	Spring 2002	Spring 2003
File I/O	94%	87%
Strings	90	93
Arrays/Vectors	52	80
Syntax Nuances	32	87
Class Design	26	53
Inheritance	62	60
Templates	26	67
Linked-Lists	45	80
Selecting the most efficient ADTs	23	27
Implementing ADTs	39	80

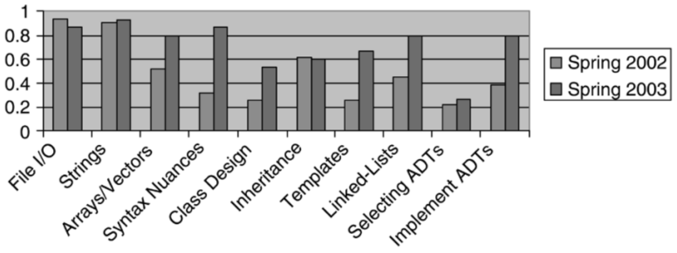

Fig. 8. Graphical display of students' aptitude at specific competencies.

ability to designate the most efficient design to solve a problem and their ability to implement that design in code.

The Spring 2002 course was composed of 31 students while the new Spring 2003 had only 16 students. The Spring 2003 student course evaluation was a quarter of a point higher (.25) than the Spring 2002 student evaluations on a five-point scale. The average of the two most relevant issues (i.e., evaluate how much you learned in this course and the effectiveness of special training techniques) was virtually the same. In addition, the grade point average of the Spring 2003 course was four points higher than the Spring 2002 course on a 100-point scale.

2) Evaluation Approach and Assessment: In evaluating the impact of the new Spring 2003 course, the student course evaluations and grade point average comparisons were somewhat ineffective. The instructor historically has had strong student evaluations, which do not vary enough to be particularly useful. Although the grade point average was higher in the new course, the increase was not significant considering the historical variance in the averages of all the CS2 courses taught by the instructor. Although both the increased score in student evaluations and in the class grade average suggest positive improvement of the students' performances in the latter of the two courses, the decision was reached that evaluating the performance on the cumulative final exams (which were similar in both semesters) was a better indicator of improvement.

Both sets of final exams were evaluated based on ten programming competencies similar to those presented in Fig. 4. In addition, the number of students that achieved an aptitude of 80% or better in that competency was calculated. This number was used to create a percentage of students in the class that excelled in that particular programming competency. The percentage based on the ratios for both classes are shown in Table I and is graphically shown in Fig. 8.

Based on the evaluation, the students were significantly better prepared in Spring 2003 to understand and program the advanced techniques, such as templates, linked lists, and implementing abstract data types. Initial programming techniques, such as file input/output, strings, vectors, and inheritance, were virtually the same for both classes. In addition, the category that considered the nuances in programming syntax significantly increased from Spring 2002 to Spring 2003. These results support the conclusion that students accepted the advanced programming training better in Spring 2003 than in Spring 2002.

However, the evaluation also shows that neither class was particularly effective in training students on identifying the most efficient advancing programming solution. In addition, class and software design (selecting the proper ADT) was below anticipated standards in the overall performance on final exams for *both* courses.

B. Anecdotal Student Responses

Students were encouraged to fill out anonymous evaluation forms or send e-mail to the instructor making suggestions on improving the course. Most students felt comfortable sending comments directly to the instructor. A paraphrased list of anecdotes is discussed below.

- *Parts of the design that were not totally specified were difficult because the communication channels were not established among students.* In the first offering of this approach, the instructor did not set up proper communication channels; therefore, design inconsistencies were more difficult for students to alleviate. In future courses, an online discussion forum should be set up to help students communicate mandatory changes that must deviate from the original design. Students suggested that regular updates during the development week (week 2) would have been helpful.
- *A strong characteristic of the project was the level of freedom given in designing the solution.* This freedom allowed the independent group to make programming choices without having to refer to the instructor beforehand.
- *Although groups were highly interdependent, it was a benefit to give the central module group the global picture.* Allowing one group to have the responsibility to connect all of the projects was better than having totally independent projects.

C. Anecdotal Instructor Responses

- *The instructor should have in-depth knowledge of the project.* Even if the instructor does not have the solution to the special project, he or she should have in-depth knowledge of the domain. Students will introduce new ideas and variations that the instructor may not have envisioned; thus, thorough familiarity will benefit the process.
- *The instructor should expect that students will not understand the project in the first discussion.* Several students needed multiple iterations to digest the specification for their module. Using an entire week for design is helpful in allowing the students to digest their individual problems. Furthermore, additional class time can assist in clarifying issues in the individual projects. Courses of this nature require the

instructor to utilize a significant amount of coaching expertise.
- *The instructor should use interface methods to help explain the initial expectations.* Showing example methods of their individual modules helped students understand what their module is expected to do. This method is the most effective way to describe the inputs and outputs for this approach to black box design.
- *Ambiguity in design has both positive and negative effects.* Some students responded favorably to having freedom of design in their implementations. Other students desired more details. The instructor should make an effort to allow some freedom in development to accommodate for the more ambitious students. However, the same vagueness, although uncomfortable, may cause other students to become more proactive.
- *Students that implement the central module receive the greatest challenge.* The students who agreed to be responsible for the central module received by far the best experience in code integration. The instructor suggests that these students be given freedom to contact other students in the final week to make small changes that assist the integration effort. This contact empowered the students as they gained ownership of the project.

D. Discussion

The most positive result from this new course is that students proactively suggested the use of data structures prior to the third week of the special project. This suggestion exceeded the expectation of the instructor and suggests that the use of the special project was valuable in helping students conceive the importance of data structures to certain real-life applications. There are also positive effects as a result of reinforcing CS1 subjects. At times, there is a disparity between the teaching approaches of multiple instructors that teach the prerequisite CS1 course. Revisiting introductory subjects helps to balance the experience of students that had different instructors for CS1. At the same time, other variations of the subjects were introduced to prevent students from seeing this review as redundant (such as the introduction of arrays as opposed to vectors in the second course). Quantitative studies show that students in the new course performed better in implementing advanced programming techniques.

Several drawbacks were discovered in the offering of this course. One drawback was that the instructor had to invest a substantial amount of time conceptualizing a sufficient problem with classwide scope. Even with the substantial project preparation, the instructor was unable to distribute the work. The main reason is that, as in any class, some students are better equipped to handle certain parts than others. More specifically, a module that may be particularly difficult for one student could be implemented by another student with ease. Additional coaching on the part of the instructor addresses this problem. Finally, this course, as with earlier courses, does not seem to provide strong enough training in software design. In future offerings, there should be additional attempts to focus on software design.

VIII. Conclusion and Future Offerings

In this paper, a method was discussed for inserting a classwide, large-scale project into one of the early computer science courses. This approach was useful in helping students understand advanced programming techniques. In addition, the students found the exercise helpful in building skills to conceptualize solutions for unique problems. The students agreed that collaborating on a solution to a classwide project presented other aspects to software design and development that they did not perceive when performing individual projects. As with earlier literature [3], [13], communication is a major consideration in these types of courses. The communication challenges are relevant to approaches presented in this paper and in semester-long courses. Online discussion forums would largely alleviate communication barriers among the students.

In future course offerings, the intention will be to extend the project to a fixed four weeks ($\sim$35% of the course projects) in an attempt to alleviate some concerns with communication and ambiguity in the project design. In addition, the project will be split into a hierarchy of large-scale projects that work together. Depending on the size of the class, multiple groups may execute several concurrent large-scale projects. This development will allow more students the opportunity to gain experience in software integration, which has a significant impact on students at this level.

Acknowledgment

The discussions of the curriculum committee helped to develop the motivation for the course changes. The curriculum committee was composed of Prof. S. Caraballo, Prof. D. Denning, Prof. B. Kalyanasundaram, Prof. M. Maloof, Prof. R. Squier, and Prof. M. Velauthapillai. J. R. Alston, M. A. Bittmann, J. Devallon, Jr., M. D. Dinkov, S. E. Sardari, J. M. Fier, J. M. Helmink, R. P. Kashyap, P. S. Kumar, K. S. Lee, W. A. Mitchell, C. J. Piro, A. Ngua, and K. E. Quinlan were students who participated in the course and provided useful comments that served as qualitative results. In addition, other students in the Computer Science II course are acknowledged for allowing the special project to run concurrently with the course. This paper was greatly enhanced by the anonymous reviewers who added to both the quality and the completeness of this work. Finally, B. Blake and D. Innerarity are acknowledged for enhancing the clarity of the work.

References

[1] M. B. Blake and T. Cornett, "Teaching an object-oriented software development lifecycle in undergraduate software engineering education," in *Proc. IEEE Conf. Software Engineering Education and Training (CSEET2002)*, Feb. 2002, pp. 234–241.

[2] J. A. Turner and J. L. Zachary, "Using course-long programming projects in CS2," in *Proc. 30th Tech. Symp. Computer Science Education*, 1999, pp. 43–47.

[3] S. A. Rebelsky and C. Flynt, "Real-world program design in CS2: The roles of a large-scale, multi-group class project," in *Proc. 31st Tech. Symp. Computer Science Education (SIGCSE 2000)*, Mar. 2000, pp. 192–196.

[4] M. B. Blake, "A student-enacted simulation approach to software engineering education," *IEEE Trans. Educ.*, vol. 46, no. 1, pp. 124–132, Feb. 2003.

[5] C. C. Bareiss, "A semester project for CS1," in *Proc. 27th Tech. Symp. Computer Science Education (SIGCSE 1996)*, Mar. 1996, pp. 310–314.

[6] M. Godfrey and D. Grossman, "JDuck: Building a software engineering tool in Java as a CS2 project," in *Proc. 30th SIGCSE Tech. Symp. Computer Science Education*, 1999, pp. 48–52.

[7] J. Krone, D. Juedes, and M. Sitharam, "When theory meets practice: Enriching the CS curriculum through industrial case studies," in *Proc. 15th Conf. Software Engineering Education and Training (CSEET 2002)*, 2002, pp. 207–215.

[8] R. J. Daigle, M. V. Doran, and J. H. Pardue, "Integrating collaborative problem solving throughout the curriculum," in *Proc. 27th SIGCSE Tech. Symp. Computer Science Education (SIGCSE 1996)*, Philadelphia, PA, Feb. 1996, pp. 237–241.

[9] E. J. Adams, "A project-intensive software design course," in *Proc. 25th ACM SIGCSE Tech. Symp. Computer Science Education*, Indianapolis, IN, Mar. 1993, pp. 112–116.

[10] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language Guide*. Reading, MA: Addison-Wesley, 1998.

[11] M. B. Blake. Evaluating Agent-to-Agent Workflow Interactions for Service Composition: Third-Party Control or P2P?. Department of Computer Science, Georgetown University, Washington, DC. [Online], Tech. Rep. CSTR-20 030 420-5. Available: http://www.cs.georgetown.edu/techreports/

[12] D. Hanesiane01– and A. J. Perna, "An interdisplinary collaborative approach in teaching freshman engineering design," in *Proc. Int. Conf. Engineering Education*, Oslo, Norway, 2001, pp. 6E5-1–6E5-5.

[13] M. Michael, "Fostering and assessing communication skills in the computer science context," in *Proc. 31st SIGCSE Tech. Symp. Computer Science Education (SIGCSE 2000)*, Austin, TX, Feb.–Mar. 2000, pp. 119–123.

[14] L. Pollock, "Integrating an intensive experience with communications skills development into a computer science course," in *Proc. 32nd SIGCSE Tech. Symp. Computer Science Education (SIGCSE 2001)*, Charlotte, NC, 2001, pp. 287–291.

[15] The Joint Task Force on Computing Curricula: IEEE Computer Society and Association for Computing Machinery, Computing Curricula 2001—Computer Science. [Online]. Available: http://www.computer.org/education/cc2001/final/cc2001.pdf

M. Brian Blake (S'98–M'01–SM'03) received the B.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, the M.S. degree in electrical engineering (minor in software engineering) from Mercer University, Atlanta, GA, and the Ph.D. degree in information technology with a concentration in information and software engineering from George Mason University, Fairfax, VA.

He is currently an Assistant Professor with the Department of Computer Science, Georgetown University, Washington, DC. His current research interests are in the area of systems and software engineering with an emphasis on application of fundamental software engineering techniques to cooperative information systems. He has authored more than 40 refereed journal and conference proceedings in the area of software systems engineering and agent-based information systems. He also maintains an ongoing relationship with The MITRE Corporation, McLean, VA, as Lead Software System Engineer Consultant, where he brings a unique blend of academic and industry experience to the software engineering courses that he teaches both at Georgetown University and for industry.