An Inter-Disciplinary Software Engineering Track Emphasizing Component Engineering [†]

David Cordes, Allen Parrish, Brandon Dixon, Richard Borie,
Jeff Jackson, David Hale, Joanne Hale, Shane Sharpe
University of Alabama
Tuscaloosa, Alabama 35487

Abstract: *This paper describes the establishment of an integrated track in software engineering for three distinct academic disciplines at the University of Alabama: Computer Science, Computer Engineering, and Management Information Systems. This integrated track focuses on component engineering, and is being developed by a team of faculty from all three programs.*

Introduction. The University of Alabama is developing a new track in its computing curriculum, available to students majoring in any field related to computing (Computer Science, Computer Engineering, and Management Information Systems). The track focuses on what is commonly known as *software component engineering*.

The basic concept behind this curricular revision involves the integration of current research results into the student's classroom experience. Our concentration involves placing students from all three program who have interests in software engineering into a common set of courses, with students forming interdisciplinary teams to address realistic issues associated with software component engineering.

Software Component Engineering Track. The faculty have been working on the development of this course sequence over the past year. During this time we have identified three distinct phases within the software engineering track. These include the study of components at both the micro and macro level [1], as well as the integration of this information into a realistic application that demands inter-disciplinary participation and cooperation. Each of these three phases is discussed in greater detail below.

Phase I: Micro-Architecture Issues. When looking at software component engineering from a micro viewpoint, one focuses on techniques that facilitate the process of making fundamentally sound designs at the lowest (component) level of the system. This, in turn, should facilitate the eventual integration of these components into a working system. Micro-architecture also addresses the issues associated with the validation and verification of individual components. The table below identifies some of the basic issues associated with the micro-architecture of component engineering.

Micro-Architecture Issues

1. Determining the appropriate programming language mechanisms to use. This includes inheritance, polymorphism, static vs. dynamic, layering issues
2. The precise functionality of the component
3. Low-level design issues
4. Specification techniques
5. Implementation and validation techniques

To implement these issues within the curriculum, we have identified two junior-level courses (taken by all Computer Science and Computer Engineering majors) that will now be taken in parallel, and whose focus will shift towards the construction of software components. The first course addresses component packaging issues, while the second addresses component content issues.

As with any other curriculum modification, care was taken to integrate this software component engineering track with existing courses within the curriculum on campus. Working closely with the faculty in Computer Science, two courses – object-oriented (OO) programming and data structures – were identified for these modifications. The OO course is a natural for component introduction, and the instruction of data structures is now presented in a context of the component framework introduced in the OO course. Course outlines for each of these courses are shown below.

Micro-Architecture Course #1

- Component specification techniques (3 weeks)
- Advanced language constructs in C++: derived classes, virtual functions (3 weeks)
- Component design: Determining a set of standard operators; General operator overloading issues; Provision of copying and data movement operators; Use of static vs. dynamic memory (4 weeks)
- Composition of components: layering vs. inheritance (2 weeks)
- Component implementation and testing techniques (3 weeks)

[†] Research for this project supported by the National Science Foundation under grant CDA-9712929.

Micro-Architecture Course #2

- Principles of performance analysis, Big-O notation (2 weeks)
- Review of basic data structures: Array, record, set, union, linked list, Stack, queue, deque (3 weeks)
- Coverage of new standard data structures: Priority queue, heap; Dictionary, hash table; Graph, directed graph, adjacency matrix, adjacency lists; Tree, binary tree, binary search tree; Red-black tree, 2-3 tree, B-tree, AVL tree (10 weeks)

These two courses will be taught as co-requisites in the curriculum. Using this model, it is possible to weave a common thread throughout both courses. The lectures will be tightly coupled, and the courses will have coordinated outside assignments for the students. Assignments in the first course focus primarily on the proper implementation [2] of components using the data structures introduced concurrently in the second course. Assignments in the second course involve the mathematical analysis of performance properties for the various data structures implementations. Most of the dialog in this course is at the abstract level, either in terms of specification languages [3] or high-level pseudo-code. Implementation details are left to the first course.

We feel that, together, micro course #1 and micro course #2 constitute a unified exposure to the notion of *micro-architecture* as a discipline. We then follow this pair of courses with a single course that presents a unified view of *macro-architecture*. This course is described in the following section.

Phase II: Macro-Architecture Issues. The goals of the macroarchitecture course are to expose the student to a number of different architectural idioms [4], and then to provide an opportunity to perform architectural design in a relatively sanitized setting. A course outline for the macroarchitecture course is given below.

Macro-Architecture Issues

1. Architectural idioms (batch sequential, pipe-and-filter, OO systems, communicating processes, rule-based systems, databases, etc.)
2. Formal approaches to software architecture
3. Pragmatic tools for architectural design, including the Unified Method
4. Software systems integration

The assignments associated with this course are sanitized, yet realistic, medium-scale projects. The idea is to select a task that requires relatively little domain knowledge, given the heterogeneous mix of students in this program. Examples of such projects might include various types of text processing applications, such as text editors or spell checkers. Students will work in small teams on these projects.

Phase III: Capstone Project Experience. The capstone sequence involves two major components: the Integrated Computing Applications Laboratory (ICAL) and a senior-level, two-course sequence required of all students in the software engineering concentration. ICAL is the umbrella under which the large-scale project is completed. Projects are solicited from industry partners, and have scopes that require each of the disciplines to contribute to the development of the product. An example of a specific project requiring this type of interdisciplinary collaboration is an application to support the entire business enterprise for a photography studio. Such an application has an image processing component (where photographs are taken and immediately displayed on a video screen for the customer to inspect), as well as the necessary business-specific interfaces. For example, the customer could select proofs for finishing by navigating through the program and selecting the desired proofs. Pricing alternatives would be integrated in the software so that different size/proof combinations result in different pricing plans. A bill is then sent to the customer and an order invoice (along with an image stream for the selected proofs) is sent to the photo-finishing lab. Such a project involves hardware/system software components for CE students, application-layer software components for CS students, and business integration layer components for MIS students.

The senior-level, two-course project sequence provides a base of students to support the technical activities of ICAL. Each class functions as a team-in-the-large, having total responsibility for completion of the assigned project. For the fall semester of the course, much of the activity will be foundational [5]. In particular, MIS team members are required to assess the business requirements and application features. In parallel, the CS students are seeding the component repositories with modules that appear to be needed, and will be responsible for developing the overall architectural design for the system.. The CE team members are investigating and benchmarking possible technologies to be used, developing potential interfaces and, together with the CS students, developing necessary device drivers. Obviously, considerable inter-group coordination is needed for a successful outcome.

The Spring semester of this course sequence involves assembling the components into a working system in order to achieve the industrial partner's goal. The project is judged as complete only after the system has been developed and tested. By design, the projects require iterative development. This forces numerous interactions by team members from all three disciplines. CS and CE students are responsible for component integration at the various levels of the system. (Generally, CE students continue to be responsible for lower-level components, while CS students are responsible for much of the application layer integration.) MIS student members are required to perform acceptance testing of system features with the industrial partner. As system features are identified that require additional modifications to be deemed acceptable, a cascade of tasks are likely to be generated (including modifications to hardware interfaces and program modules) that demand the entire team's effort to implement.

Summary. As noted in the introduction, our proposed curriculum involves the development of a single software engineering concentration for The University of Alabama. In particular, students from computer science, computer engineering and management information systems are eligible to participate in this concentration. The concentration permits each group of students to complete their respective major degree programs while also meeting the specific concentration requirements. We feel that the integration of these three areas is a major benefit of this proposal. It is rare that faculty from business and engineering colleges are able to successfully collaborate on curriculum issues. More often than not, the various academic computing programs at universities (particularly when multiple colleges are involved) are competitively battling for students and resources. Given that the majority of students from all three programs are interested in jobs that involve elements of software engineering, the provision of a single academic curriculum to provide students with common fundamentals (and to utilize the diversity of student backgrounds on a realistic development project) should provide great benefit.

References

[1] Hollingsworth, J.E. and B.W. Weide, "One architecture does not fit all: Micro-architecture is as important as macro-architecture," *Proceedings of WISR7: Workshop on Institutionalizing Software Reuse*, 1996.

[2] Coplien, J., *Advanced C++ Programming Styles and Idioms*, Addison-Wesley, 1992.

[3] Ogden, W., M. Sitaraman, B. Weide, and S. Zweben, "The RESOLVE framework and discipline: A research synopsis," *Software Engineering Notes*, vol 19, no 4, pp. 23-28.

[4] Shaw, M. and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[5] Lakos, J, *Large-Scale C++ Software Design*, Addison-Wesley, 1996.