Replacement of Quarter Courses CSE 221, 222, and 321 by Semester Courses CSE 2221 and 2231

Autumn 2011 Update

(Weide as CSE 2221 and 2231 course coordinator, with help from Bair, Bucci, and Heym)

This brief document summarizes the reasons behind the most significant changes to be made as we replace the current year-long introductory course sequence for CSE, CIS, and ECE (computer option) students with a new semester sequence.

A key attribute of the existing course sequence that will be retained is the early focus on component-based software engineering, including design-by-contract and its formal expression using basic concepts and notations of mathematics. This will remain a distinguishing feature that sets the sequence apart from nearly every other introductory CS sequence anywhere, as it has for the past 15+ years. In other respects, however, there will be major changes.

Change: Programming language will be Java.

Rationale: Students at the undergraduate forum and on exit surveys have made clear their lack of enthusiasm for C++ as the language introduced in these courses. Even many of those who accept C++ as being tolerable would prefer to learn a language like Java that is more widely used by employers looking for interns, not to mention for permanent employees. The problem goes well beyond C++, though. Some students mistakenly believe they are not learning C++ at all but rather a different programming language they call either "Resolve/C++" or "Resolve". The point made multiple times in the intro course sequence—that Resolve/C++ is not a language but rather a discipline for component-based software design and development to be used with C++, as evidenced by the fact that we use an ordinary C++ compiler—has been lost on these students. No amount of explaining that the programming language is C++ seems to have made a dent in the number of students who insist otherwise. Some have even complained that when they tell prospective employers the language they learned in these courses is "Resolve", employers tend to respond in disbelief (no surprise). We accept the blame for this misconception by some students and recognize that it turns out to have been a terrible idea to give the discipline a name at all, and an even worse idea to have used C++ preprocessor macros to try to make C++ look more like a modern object-oriented language by introducing meaningful keywords like "extends" and "implements" as synonyms for "virtual public", etc. In 1995 that might have seemed like a decent idea: C++ was the only real game in town yet it already seemed a bit dated. But in 2011, why not simply use a more modern object-oriented language, e.g., Java? And why not jettison a specific name for the discipline and simply call it by the generic name "best practices"? Having witnessed the success of Paul Sivilotti's CSE 421—in which he has done just that with students who already have been introduced to the concepts in the intro sequence—we decided to switch languages to Java at the time of semester conversion and never to mention a "Resolve discipline" again in these courses.

Risks: In C++, it is possible to design software components that have "clean semantics" by using the Resolve/C++ discipline. Of course, typical software components in C++ are not designed to have clean semantics but rather to have the usual reference semantics employed in all similar object-oriented languages. However, that one *can* do better by careful design discipline, and

reap the modular reasoning rewards of doing so, is in some sense the main technical point of the current course sequence. Unfortunately, the same is technically just not possible with Java. However, by following best practices, including some that are loudly advocated now even by those outside our research group (e.g., by Joshua Bloch of Google in *Effective Java*), one can come pretty close to obtaining the advantages of clean semantics. The story is not perfect but it is much better than it was when we selected C++ for our intro course sequence. Java has evolved substantially since the mid-1990s, e.g., adding generics; and so have OOP best practices in industry and the tools to support them, e.g., Eclipse and standard plug-ins for it including Checkstyle and FindBugs.

Change: Programming environment will be Eclipse.

Rationale: Though, perhaps surprisingly, there have been relatively few complaints at the undergraduate forum and on exit surveys from students about using emacs as the programming environment, some students clearly do consider this to indicate that the course material is not "modern". The decision to switch to Eclipse is a no-brainer when the language is Java.

Risks: Students will not learn how to use Unix (which they do now) in the new intro courses. It will be up to the systems courses to make up the deficiency to the extent it is considered important. In addition, there will be a 1-cr-hr "Introduction to Unix" course available for those students who wish to learn more about using Unix (or Linux).

Change: Learning outcomes will be broader.

Rationale: No students have complained about the learning outcomes. The reason we are making changes here is that the instructors for the existing intro courses have noticed that some students have interpreted the specificity of the existing learning outcomes with respect to certain language features as an indication that the programming language is the primary focus of these courses. We have therefore generalized and broadened the software engineering outcomes to the first course, Software I (CSE 2221), e.g.:

- Be familiar with the reasons it is important that software be "correct", i.e., why "good enough" is not good enough when it comes to software quality
- Be familiar with the reasons for designing software to minimize the impact of change, and why it is difficult to achieve this

Similarly, for Software II (CSE 2231):

- Be competent with using design-by-contract principles and related best practices, including separation of abstract state from concrete representation
- Be familiar with using many industry-standard "best practices" for Java design and development

We believe these changes and others in the learning outcomes will help make it clearer to students that there are four main themes running through these courses: concepts, language, tools such as IDEs, and professional best practices. Presently, the intro courses appear from their outcomes to have three main themes: language, tools such as editors and operating system commands, and a conglomeration of concepts and design discipline that is viewed as being local to OSU and essentially unrelated to the real world.

Risks: None that we can see. We are already emphasizing these concepts in the courses, but we're not saying it so explicitly right now.