CSE331: Software Design and Implementation
Course Description
Draft of September 5, 2009

Note: This document was written by Dan even though he does not intend to teach the course in the near term and did not participate in all the discussions about it. After no progress by others for several months, Dan designed the course outline below.

Structural place in the curriculum

- 4 credits (3 weekly lectures, 1 weekly section)
- Pre-requisites: 143
- A pre-requisite for 403, recommended for several project-oriented 400-level courses, including software capstones
- Taken by: Required for CS and the software-track of CompE, optional for the hardware track of CompE
- Catalog description: To be determined

Course Overview / Goals

The course goal is to develop students' ability to understand, develop, and reason about modern software development and programs of moderate size (say 1000-5000 lines, which is bigger than they see in CSE142/143). Of course, "modern software development" means far too many things for one course. This course generally favors depth, such as project experience, over breadth. Toward that end, the course primarily builds on students' familiarity with Java while adding some scripting-language experience near the end. While some object-oriented methodology is important, the course is much broader than just the OOP paradigm.

Particular foci of the course include:

- Notions of correctness and establishing it: testing, verification, assertions, invariants, ...
- Notions of program structure: modularity, code reuse, separation of concerns, programming patterns (e.g., inheritance, callbacks, generic interfaces, and design patterns)
- Examining program behavior (debugging, profiling)
- Project experience: working in a team, informal specification, design, documentation, milestones
- Dealing with the external world: some combination of asynchrony, event-driven programming, GUIs, or similar I/O complexity

Variability among offerings:

Of all the new 300-level courses, this course can probably afford the greatest variability among offerings by different instructors since:

- There are many ways to achieve additional maturity with software development.
- Because the material is mostly not covered in our current curriculum (except perhaps two weeks overlap with 303), it will take time to determine what parts of the course are essential for our 400-level courses.

In the course outline below, the material in the first six weeks is presumably stable in that one cannot imagine a version of the course not including almost all of it. The remaining material could more reasonably be replaced by complementary topics, which might depend on the course project.

Description of Possible Homework, Etc.

The course begins with a few weeks of individual weekly homework assignments followed by a four-week project completed in teams of 3–4 students. The homeworks involve writing code, but also writing test cases, invariants, and documentation. Similarly, the project would focus primarily on the code, but not only on the code.

A midterm and a final would both fit perfectly well, but may be worth less than is typical given the substantial project. Not having a final is also possible.

Possible Textbook

To be determined.

Non-Topics and Relation to the Rest of the Curriculum

In the old curriculum, we do not have a course like this. It shares goals with some of the original motivation of 303, but 303 has evolved into a course more about Unix, C, software-development tools, and societal implications. 303 is much closer to CSE333 Systems Programming.

In the new curriculum, this course provides a segue between 143 and other software courses. Because it is not related via pre-requisites with other 300-level courses, students can take 300-level courses in many orders, which is essential for scheduling. The trade-off is that 331 comes neither before nor after:

- 351, which covers "what really happens" when Java is compiled
- 332, which covers classic data structures and algorithms (note 143 covers stacks, queues, and binary search trees)
- 333, which covers C programming (beyond a short introduction in 351)
- 311, which covers propositional and first-order logic, thus precluding a more formal treatment of code preconditions in 331

Also note that with 341 now optional, the number of languages and programming paradigms in our required courses is smaller.
331 is not a replacement for 403 Software Engineering, an optional senior-level course. 403 would be much more about the large-scale software-development process, of which programming is only a piece. In contrast, 331 is very much a programming course that follows our two-quarter introductory sequence while exposing students to the next level of software complexity.

Approximate/Example Course Schedule
Missing from this schedule is a lecture or two that may be necessary to prepare them for the specific project. We should not, "throw them to the wolves" but rather engineer success with a 3–4 person, 3–4 week project.

Week 1: Segue from 143

- Review / slight extension of abstractions and reuse approaches: Subclassing, abstract classes, interfaces, generics, equals vs. ==
- Contrasting benefits of generics and subclassing, comparison to casting to/from Object
- Interface vs. implementation, library vs. client, representation hiding
- Section: Basics of a full-fledged IDE such as Eclipse

Week 2: Testing

- Test suite, expected observable outputs, unit testing, coding for testability
- Coverage metrics, false negatives

- Regression testing
- Section: Pragmatics of a unit-testing infrastructure such as JUnit
- Homework idea: Take a small library and build unit tests for it. Ensure test suite has full branch coverage. Write a short document explaining how: (1) a small code change could cause the test suite to no longer have branch coverage even though the code is still correct and (2) a small code change could cause the code to be wrong even though the test suite still passes and still has full branch coverage.

Week 3: Specification / Verification / Invariants

- Assert statements
- Pre- and post-conditions for methods and code blocks; requires / ensures / modifies
- Loop invariants, object invariants
- Functional correctness vs. performance
- Simple static checks and notion of false positives
- Section: Javadoc, practice with preconditions and invariants

Week 4: Larger-scale program design / patterns

- General concepts: coupling & cohesion, principle of least privilege, how to build abstractions, good use of exceptions
- Higher-order programming: Using call-backs for event-listeners and as an alternative to iterators
- Some larger-scale design patterns, such as model-view-controller and an event loop
- Section: Maybe coding details on anonymous inner classes, why local variables from the enclosing method must be final, etc.

Week 5: Frameworks and GUIs

- Common widgets and OO class hierarchies supporting GUIs
- Java Swing library
- Maybe some GUI design principles
- Perhaps a second framework
- Section: Testing GUI code? Midterm review?

Week 6: Tools and Methodologies for Team Development

- Version control (e.g., Subversion), essential concepts of working copies, source vs. generated files, etc.
- Build managers (e.g., make, ant), essential concept of dependencies
- Automated regression testing
- "Software Engineering Lite": notion of requirements, intermediate milestones, etc.
- Section: Tool pragmatics

Week 7: Debugging

- Methodology: hypothesis validation, stubbing

- Tool: Breakpoint debuggers, common features and limitations
- Performance debugging: profilers, common features and limiations
- Section: Tool pragmatics, project details as necessary

Week 8: Scripting

- Some Python basics, "what it's good for"
- Advantages and disadvantages of lightweight languages
- Static vs. dynamic typing
- Section: Tools for Python (unit-testing, debugging, documenting, etc.)

Week 9: Common Software-Engineering Concerns

- Security: basic concepts, threat model, trusted computing base, string-injection attacks, denial-of-service, fuzzing
- Internationalization
- The outside world: Persistence, serialization, foreign calls, platform portability
- Section: Java and Python calling each other

Week 10: Wrap-up

- "Putting it all together": What a "real" industry or open-source project looks like. A one-lecture "show-and-tell" of a real software system and how it uses the tools and concepts from the course.
- Project post-mortem
- Course review

Project scope: Something in Java with a GUI, graphics manipulation, some simple algorithms, nontrivial testing, and some room for student creativity. One idea is a small photo-editing program where users can load pictures, crop them, rotate them, invert colors, and generate an HTML page with captions. A Python piece could perform testing and/or installation.

Commentary:

Admittedly, week 8 seems "tossed in" but it can be well-integrated into the course to re-enforce all the key ideas. It can demonstrate that *nothing* in the course is Java-specific. It can make the project more textured by requiring a small scripting piece or to use Python for automatic installation or testing. It can emphasize that different languages check different things statically and this dramatically affects testing and assertions.

Clearly if the schedule has too much stuff weeks 8 and 9 are the most ancillary, but the earlier material cannot be delayed to the detriment of the project.