Winter 2010

CEG 460/660-01: Introduction to Software Computer Engineering

Prabhaker Mateti
Wright State University - Main Campus, prabhaker.mateti@wright.edu

Follow this and additional works at: http://corescholar.libraries.wright.edu/cecs_syllabi

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation

This Syllabus is brought to you for free and open access by the College of Engineering and Computer Science at CORE Scholar. It has been accepted for inclusion in Computer Science & Engineering Syllabi by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu.
CEG 460/660: Introduction to Software Engineering

Prabhaker Mateti

This course is concerned with the techniques of designing and constructing large programs. Some of the required basic concepts necessarily have to be developed using small programs as examples. To this extent, we also study programming-in-the-small. The overall objectives are to present an overview of issues in the development of software, to discuss terminology, to illustrate via example case studies, and to give sufficiently detailed advice on how to develop quality software.

Catalog Description:

[4 Credit Hours] Concepts of software engineering. Analysis, design, and implementation of software engineering concepts that comprise structured programming and design. Case studies serve as examples illustrating the software life-cycle model. Three hours lecture, two hours lab. 4.0 Credit hours. Prerequisites: Undergraduate level CS 400 Minimum Grade of D

Source Material

We won’t be following any particular book page by page. Any recent software engineering text book will do -- three are cited below. But please do not print hard copies using WSU lab printers.

- Pierre Bourque, Robert Dupuis, Alain Abran and James Moore (Editors), "Guide to the Software Engineering Body of Knowledge", 2004. This is free to read in HTML format only: http://www.computer.org/portal/web/swebok/about.
- Ivan Marsic, Software Engineering, Rutgers, The State University of New Jersey, June 2009, 430 pp. This is a free e-book (pdf) to download: http://www.ece.rutgers.edu/~marsic/books/SE/

Class/Laboratory Schedule

Home Page: http://www.cs.wright.edu/~pmateti/Courses/460/ Bookmark this link and visit it periodically. All course related announcements will be posted on this page.

Each week has two lectures of 75-minutes each. There is no scheduled lab. Students are expected to work in open labs for no less than 2 hours a week.

There is a Discussion Group where students are encouraged to discuss with each other and the instructor, and are advised to check the home page frequently between classes for important updates.

Course Content

The tentative content is as follows. The topics, except software tools and project discussions (equiv to 2 lectures), will be covered roughly in the order they are listed. The numbers in parens at the end of each topic is a rough estimate of the number of (75-minute) lectures I intend to devote. The word "spread" means that precise definitions, lengthier discussion are to be found spread over the entire course.
Introduction


Terms: precision, rigor, formalism, requirements, specs, design, implementation, testing, debugging, maintenance, versions, configurations, life-cycle issues. Prototyping, executability, declarative vs imperative. (spread)

Requirements Analysis

Analyzing the client-given description of the system to be built: precision, ambiguity, consistency and completeness. Describing the results back to the client. Are we describing the right product? (2)

Specification

Functional and other axes of specs. Informal specs written in natural languages, and their problems. 'Declarative' languages for specs. Specs of systems, modules, procedures, and code-segments. Pre-Post conditions (entry/exit assertions), loop and class invariants. Independence of implementation bias. (4)

Design

Step-wise refinement, virtual machines, and classes. Levels of design as we progress towards the given primitives of construction. Hierarchy, modularity, information hiding in each level of design. Cohesion and coupling. Structured design. Object oriented design. UML (4)

Testing and Debugging

Black-box testing, coverage of statements, branches and paths. Data flow coverage. Mutation analysis. Symbolic execution. Unit testing, system testing. Test data selection. Monitoring the state of the running program. (2)

Construction and Tools

Systematic translation of detailed designs into a real programming language. Assembly languages, system implementation languages, and high-level languages. Support for modularity, information hiding, separate compilation, compile-time type checking, run-time watch dogs, exception handling, and concurrency.

The computer-aided design and development of software. make, programmable editors emacs, revision control rcs, svc, git, configuration mgmt, string and pattern languages awk, cross reference generators, documentation tools knuth's WEB. (4)

Social and Ethical Issues

Ethics issues are covered both explicitly and implicitly throughout CEG 460 During Requirements Analysis, after talking about "who is the customer," the "Software Engineering Code of Ethics and Professional Practice" is handed out and we explicitly discuss many of its points. There is a question on the midterm/final explicitly dealing with ethics.

Grading

http://www.cs.wright.edu/~pmateti/Courses/460/Top/460-syllabus.html 1/5/2010
There will be two exams each contributing 30% to the final grade. The midterm is scheduled during the 6th week. The Final is as scheduled by the Registrar during the finals week. Course grades will be based on the total score as follows. A: 90-100, B: 80-89, C: 70-79, D: 60-69, F: below 60. Grades may be further curved if appropriate.

**Project**

There is a project contributing 30%. The project details and due dates will be announced in class and on the home page.

The implementation language must be one of Java, or C++.

There are no oral presentations. Students submit documentation in all phases of their project.

**Homework Assignments**

There are four homework assignments that are to be solved by you *individually*, not as a group. I will grade two out of these four, without pre-announcing which two will be graded. The two graded assignments will contribute 5% each to the final grade.

**Learning Objectives**

The student should have learned the following: 1. Understand the need to apply a structured software engineering discipline. 2. Understand the advantages of following a structured lifecycle approach. 3. Understand the tasks to be accomplished during requirements analysis, design, implementation, testing, and maintenance of software systems. 4. Understand the object-oriented model. 5. Understand the structured analysis and design model. 6. Understand the various testing techniques available to ensure quality. 7. Experience the benefits of applying a structured approach as well as the problems that occur in a group development of a large project.

**Prerequisites by Topic**

1. Mature use of an object-oriented programming language. 2. Use of basic engineering facets: top-down design, narrow interfaces. 3. Abstract data types, objects, encapsulation.

**Course Outcomes**

The student should be able to: 1. Describe the benefit of process and lifecycle models. 2. Do requirements analysis and develop a specification model. 3. Create a design model from a specification. 4. Implement a system in code from an abstract design model. 5. Apply basic test case methods to code modules.

**CEG 660**

Graduate students will do, in addition to all the above, two Readings in Software Engineering that I assign, and write a 2 to 5 page summary.