CS 4278/5278 Principles of Software Engineering

3 credit hours, 150 minutes/week

Instructor: Jules White

Course description: This course uses hands-on learning to introduce the principles of software engineering. The course covers key topics ranging from requirements to testing to maintenance. Specific emphasis is placed on teaching students how to learn and acquire new software development skills.

Prerequisites: CS 3251
Specific Outcomes of instruction: Upon completion of the course, students will be familiar with state of the art software engineering, the dynamics of working in teams, the importance of managing ethical and social aspects of software engineering in teams, the criticality of rigorous software measurement, build/test/deployment automation, managing planning uncertainty/risk, and presenting software projects to engineering and non-engineering audiences.

Relationship to ABET Outcomes:

a. Ability to apply mathematics, science and engineering principles
    a. Realization in course: Application of mathematical and scientific principles to realize complex group projects.
b. Ability to design and conduct experiments, analyze and interpret data
    a. Realization in course: Identification of hidden hypotheses/assumptions in software design, execution of performance and other experiments, analysis of results to improve software design
c. Ability to design a system, component, or process to meet desired needs
    a. Realization in course: End-to-end design of all components in large-scale group software projects to meet real-world requirements
d. Ability to function on multidisciplinary teams.
    a. Realization in course: All course content is taught in the context of group projects
e. Ability to identify, formulate and solve engineering problems.
    a. Realization in course: Identification and design/implementation of solutions to solve large-scale engineering challenges for group projects, such as scalable/secure grading of 100,000 code bases from unknown sources
f. Understanding of professional and ethical responsibility.

    a. Realization in course: Discussion and exercises to deal with ambiguous requirements, scope creep, and other ethical situations that require engineers to voice concerns, etc.

g. Ability to communicate effectively.
    a. Realization in course: Presentations to technical and general audiences of projects to audiences of 50 or more people.
h. The broad education necessary to understand the impact of engineering solutions in a global and societal context.
    a. Realization in course: Understanding of software engineering issues/problems, presentations relating team challenges to real world software engineering project failures, ongoing discussion of how project design decisions affect broader community and society
i. Recognition of the need for and an ability to engage in life-long learning.
    a. Realization in course: Mandatory learning outside of the classroom of tools/libraries/frameworks/techniques needed to meet project requirements and to discover and share new software engineering tools, techniques, and processes with other students in the course
j. Knowledge of contemporary issues.
    a. Realization in course: Weekly requirements to seek out and share materials on contemporary software engineering challenges/solutions with class
k. Ability to use the techniques, skills and modern engineering tools necessary for engineering practice.
    a. Realization in course: Implementation of complex software projects using sophisticated integrated development environment, cloud platforms, mobile platforms, debuggers, developer operations tools, and automation frameworks

Topics covered include:

- Requirements analysis
- Estimation
- Testing
- Automation
- Client interaction
- Documentation
- Automation
- Planning & risk
- Monitoring & maintenance
- Architecture
- Refactoring
- Team dynamics
- Agile processes
- Coding standards
- Version control
- Presentation skills
- Broader impact analysis
- Microservices
- Containers
- Web applications

Example Project:

> **A Scalable Autograding System for 100,000 MOOC Students:** A second example project from the 2014 class is a scalable automated grading platform for Coursera Massively Open Online Courses (MOOCs). The students will design and develop a secure automated grading platform to build, deploy, and test C++ and Java code for 3 Coursera MOOCs. The automated grading platform will be deployed in May of 2014 and support automated grading for the 100,000+ students that will be enrolled in a series of trans-institutional sequenced MOOCs taught by professors at the University of Maryland and Vanderbilt University. Students will employ virtualization and isolation technologies, such as Vagrant and Docker, to isolate, build, execute, and test code submitted by students. Significant automation around JUnit and C++ Unit will be required to manage the testing processes. Web-based interfaces will be built on top of the Java Spring Framework and hosted in Amazon EC2.

Course Format: The course will be divided into 4 key phases. In the initial phase of the course, students will be responsible for outside reading focused on requirements, agile processes, estimation, risk, ambiguity, and software engineering anti-patterns. Classes will employ hands-on group exercises to explore these topics and illicit discussion on lessons learned. The second phase of the course will emphasize key tools for software engineering, including debuggers, automated build tools, test frameworks, dependency management, dependency injection, configuration injection, logging, and enterprise application frameworks. Students will complete hands-on exercises in class to obtain a deeper understanding of these topics. The third phase of the course will focus on allowing students to exercise their requirement specification, estimation, planning, and knowledge transfer skills by designing assignments that must be completed by the other students. Teams of students will be responsible for ensuring that their assignments are clear, appropriately scoped, and implemented successfully by other students. In the fourth and final part of the class, teams of students will develop the specifications and MVP implementations of research-oriented software projects for mentors from around Vanderbilt. During the first and fourth phases of the course, the class and teams will include non-engineering students registered in the UNIV 3278 course.

Class Schedule:

> 2 classes per week of 75 minutes each

Topics:

1. Course Format and Project Overview
2. Software Processes (Interdisciplinary Classes with UNIV students)
   - Agile Processes
   - Waterfall Processes
   - Software development anti-patterns
   - Scrum
   - YAGNI

- Mythical Man
- No Silver Bullet

3. Requirements & Estimation (Interdisciplinary Classes with UNIV students)
   - Managing requirement ambiguity
   - Estimating development complexity
   - Planning sprints
   - Identifying hidden hypotheses and designing validating experiments
   - User Stories
   - Asking questions
4. Testing & Debugging
   - Debuggers & breakpoints
   - Unit testing
   - Integration testing
   - Performance testing
   - JUnit
   - Mock objects
5. Enterprise Applications & Services
   - Dependency Injection
   - HTTP & REST
   - Object Marshalling
   - Java Annotations
   - Spring Boot
   - Object Persistence
6. Applying Skills & Acquiring Advanced Knowledge
   - Student-driven assignments
   - Student-driven support of assignments
   - Enterprise application generation with JHipster
   - Performance & log monitoring with the ELK stack
   - Containerization with Docker
   - Microservices
   - Spring Netflix Cloud, Zuul, Ribbon, Eureka
   - Docker Compose
   - React
7. University Mentor Project MVPs (Interdisciplinary Classes with UNIV students)
   - Work with Vanderbilt researches to scope large-scale software projects to tackle big research and societal challenges
   - Compete for mentor's projects with pitches and plans
   - Develop specifications for entire project
   - Develop specifications for an initial MVP
   - Develop wireframes
   - Develop MVPs
   - Intermediate checkpoints
   - Final presentations