Master Syllabus for TCSS 360
Version: September 2009

The **primary goals** of the course are:

1. Understanding the software development lifecycle.
2. Understanding the role of software requirements, and practice with a use-case driven approach to requirements gathering.
3. Practice in generating a software design for a medium-sized software project (~20 classes, 5000 SLOC), through the specification of the component parts, the responsibilities for each part, and the interactions between the parts.
4. Practice with UML as a modeling notation for static and dynamic aspects of software systems.
5. Exposure to disciplined software development practices for increased software quality, including Design by Contract, automated unit testing, integration and regression testing, and design reviews.
6. Effective group practices to produce high-quality software within a fixed time frame.
7. Direct hands-on exposure to the specification, design, and development processes through an extended group project that involves requirements interpretation and specification, design, and implementation of a working solution.

Prerequisites: TCSS 342.

Topics covered

1. The Software Lifecycle and process models such as Waterfall, Spiral, Agile Methods
2. Effective groupwork practices
3. Groupwork technology such as version control and tracking systems
4. Use cases and requirements gathering
5. Design generation, guidelines, and notations
6. Design by contract
7. Design patterns
8. Code conventions
9. Dynamic verification: unit, integration & regression testing
10. Static verification: reviews, walk-throughs & inspections

General Student Postconditions

1. Recognition of the importance of understanding requirements prior to writing code.
2. Understanding the importance of maintaining a structured process in large, complex projects.
3. Understanding how design effort provides large-scalestructure to non-trivial programming projects.
4. Understanding the importance of clear specification of class interfaces, especially within a team-oriented software development project.
5. Ability to read UML sequence diagrams to understand how classes collaborate to carry out scenarios.

6. Ability to generate class diagrams to document the class structure of a software project.
7. Knowledge of black- and white-box unit testing and how to develop automated tests or test plans that ensure thorough unit testing.
8. Facility with groupwork practices and technology that lead to individual accountability and commitment to group goals.