TCSS 360 Master Syllabus
Version: April 2011
(Approved: 27 May 2011)

Catalog Description

How to build quality software using standard development practices and representations. Writing and using requirements, designing and representing computational units, rigorous program testing, reviews and inspections, working effectively in teams. Prerequisite: a minimum grade of 2.0 in TCSS 342; 10 credits of writing coursework.

Preconditions

- Design, implement, document and debug a medium complexity program with an object hierarchy that includes interfaces and/or abstract classes and a graphical user interface, given some guidance on the design.
- Design and implement unit tests for a medium complexity program with an object hierarchy that includes interfaces and/or abstract classes.
- Utilize modern software engineering tools (e.g., IDEs, static checkers, unit testing frameworks, revision control systems) during the implementation of a medium complexity program.
- Correctly employ programming language features by reading and interpreting the associated published API documentation.
- Instantiate and apply the API's of at least 4 data abstractions (or their correlates) in the canonical set as provided in the standard generic library of a modern programming language.
- Select data abstractions, structures, and implementations that a developer would use in solving larger problems and defend the appropriateness of these choices.

Student Learning Goals (to be added to syllabus handed out to students)

- Write and validate client requirements incorporating standard forms of representation for projects with approximately 3 users carrying out several tasks each.
- Construct and verify a design based on client requirements composed of multiple computational units, represented with standard forms.
- Code and test a subset of client requirements consistent with a design.
- Participate effectively in a team to carry out the above tasks using appropriate tools for facilitating teamwork.

CSS Degree Student Learning Outcomes that this course contributes to (to be added to syllabus handed out to students)

a. an ability to apply knowledge of computing and mathematics appropriate to the discipline;

b. an ability to analyze a problem, identify and define the computing requirements appropriate to its solution;

c. an ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs;

d. an ability to function effectively on teams to accomplish a common goal;

e. an understanding of professional, ethical and social responsibilities;

f. an ability to communicate effectively with a range of audiences;

g. an ability to analyze the impact of computing on individuals, organizations and society, including ethical, legal, security and global policy issues;

h. recognition of the need for, and an ability to engage in, continuing professional development;

i. an ability to use current techniques, skills, and tools necessary for computing practice.

UWT Student Learning Goals that this course contributes to (to be added to syllabus handed out to students)

- *Inquiry and Critical Thinking*: Students will acquire skills and familiarity with modes of inquiry and examination from diverse disciplinary perspectives, enabling them to access, interpret, analyze, quantitatively reason, and synthesize information critically.
- *Civic Engagement*: Students will define their roles and responsibilities as members of a broader community and develop an understanding of how they can contribute to that community for the greater good.
- *Communication/Self-Expression*: Students will gain experience with oral, written, symbolic and artistic forms of communication and the ability to communicate with diverse audiences. They will also have the opportunity to increase their understanding of communication through collaboration with others to solve problems or advance knowledge.

Topics covered

1. Software lifecycle and process models such as waterfall, spiral, and agile methods.
2. Effective groupwork practices, such as mutual accountability and public commitments.
3. Groupwork tools such as version control and tracking systems.
4. Requirements elicitation, and standard representations for requirements, such as use cases and interaction prototypes.
5. Design generation, design representations, and heuristics for good design.
6. Dynamic software verification: unit, integration & regression testing
7. Static software verification: reviews, walk-throughs & inspections