COMP 3350 – Software Engineering 1

Course Description

Calendar entry

Introduction to software engineering. Software life cycle models, system and software requirements analysis, specifications, software design, testing and maintenance, software quality. Prerequisites: COMP 2150 or ECE 3740.

General Course Description

Earlier courses in the program focus mostly on individual students writing small, isolated programs for assignments. This course introduces software development as an activity for a team of developers building a substantial project through the entire term. Each team builds a complete product that solves a real-world problem.

To accomplish this goal, we learn important techniques of software design, development strategies, and just enough project management skills at each stage to build and refine the final product.

Detailed Prerequisites

Before entering this course, a student should be able to:

- Design, implement, and build software requiring multiple modules with well-defined interfaces.
- Make use of object-oriented concepts like abstract classes, polymorphism, encapsulation and overriding.
- Collect objects into data structures.
- Recognize the benefits, in terms of code readability and maintainability, of compartmentalizing code into different objects and hierarchies of classes.
- Write code that makes identifying and fixing problems easier.
- Write thorough unit tests for isolated code components.

Additionally, there is some expectation that students should demonstrate programming maturity and be able to contribute to their team by offering **at least one** of the following skills:

- Designing, prototyping, and evaluating a user interface.
- Designing and implementing a relational database system.
- Writing effectively in the context of the computing profession.

These skills, or equivalently advanced skills, are typically demonstrated by having completed at least two other 3000 level Computer Science courses before registering in Software Engineering 1.

Course Goals

By the end of this course students will:

- Compare agile and rigorous software development.
- Integrate testing with implementation to ensure thorough test coverage.
- Develop software to evolving requirements.
- Refine software as it being developed.
- Maintain an existing software system.
- Develop code in a team and share code safely amongst the members.
- Use an agile approach to facilitate incremental development.
- As part of a team, design and build a complete software product to solve a real-world problem.

Learning Outcomes

Models of Software Development

Students should be able to:

1. Identify the stages of the Software Development Life Cycle.
2. Describe how the SDLC stages can be incorporated into a software development process.
3. Identify the characteristics of “lightweight” and “heavyweight” processes.
4. Identify the priorities of an agile software development process.
5. Apply a specific agile process to develop a product as a team over the course of a term.

Process

Students should be able to:

1. Demonstrate analysis of a problem by communicating a vision that describes its solution.
2. Apply an iterative development strategy to build and refine a product.
3. Use conversational techniques to produce non-technical user stories that motivate software features.
4. Estimate and prioritize user stories to generate an appropriate set of developer tasks for each iteration.

Development Techniques and Practices

Students should be able to:

1. Work on a shared code repository as a team using a version control system.
2. Identify and use basic tools of a version control system, such as branching and conflict resolution.
3. Use an IDE to manage the build and dependencies of a complex software project.
4. Describe technical debt and identify strategies for avoiding it.
5. Use practices such as code review, collective ownership, and pair programming to facilitate development in a team.
6. Continually refactor a complex software project to maintain code quality.
7. Identify and apply strategies for modifying legacy code.

Testing

Students should be able to:

1. Identify the reasons for and general strategies of software testing.
2. Write a thorough and well-organized set of unit tests with complete coverage of a complex software system.
3. Apply strategies such as TDD to ensure good test coverage and quality.
4. Use test doubles such as stubs and mocks to replace dependencies for testing.
5. Implement integration tests to verify the seams between system components.
6. Define and implement higher level tests such as end-to-end and acceptance tests.

Software Design

Students should be able to:

1. Measure software design quality using metrics such as coupling and cohesion.
2. Apply design principles such as DRY, Separation of Concerns, Principle of Least Knowledge, and SOLID Principles to produce good design solutions.
3. Apply dependency injection to replace services decoupled from clients.
4. Identify common design smells and anti-patterns that produce poor code.
5. Implement cross-cutting concerns with minimal dependencies, with a particular emphasis on error-handling strategies.
6. Describe the drawbacks of inheritance, and how to replace it with composition.
7. Identify and apply design patterns to solve common software design problems.
8. Compare and use basic architectural design patterns such as layered and MVC.