CSE 682 Software Engineering
Syllabus

Instructor: Edmund Yu, PhD
Associate Professor
Email: esyu@syr.edu
Phone: (315)443-3110
Office: CST 4-206L

Textbooks

- Sommerville, Ian. *Software Engineering*. 10th ed. Addison-Wesley(Pearson), 2015. ISBN 0133943038

Parts of the following books will be referenced in this course, but they are not required:

- Booch, Grady, et al. *Unified Modeling Language User Guide.* 2nd ed. Addison-Wesley, 2005.
- Gamma, Erich, et al. (The Gang of Four). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- McConnell, Steve. *Code Complete,* 2nd ed. Microsoft Press, 2004. ISBN 0735619670

Course Description

When building a significant software system, the first thing a software engineer should do is make sure he or she is building the right one. This process is commonly known as the requirements engineering process. In this course, students learn and familiarize themselves with the entire requirements engineering process, starting from feasibility study and requirements gathering, through requirements analysis and specification, and down to requirements validation and management.

The logical next step in building a significant software system entails making several, sometimes hard, design decisions about the architectural, structural, and behavioral properties of the system. In this course, techniques and notations, such as the Unified Modeling Language (UML) and design patterns, are introduced for specifying these properties of software systems. Students learn to use those techniques and notations to analyze, design, and document software systems. Finally, students learn how to implement their software designs.

As their term projects for this course, students will also work in teams to undertake the preparation of a software requirements specification (SRS) document for a substantial piece of software, explore high-level design options for implementation, and carry out its detailed design and implementation, meeting (insofar as possible) the requirements set forth in a previously generated SRS. The resulting software should be well organized, well documented, and thoroughly tested. Students will present their SRS to the class at midterm, and their final systems at the end of this course.

Schedule

Week 1: Introduction to Software Engineering and Software Processes
This week's objective is to introduce software engineering and the idea of a software process, which is a coherent set of activities for software production. Specific topics include:

- What is software engineering?
- Why is software engineering important?
- Different types of software systems, which may require different software engineering techniques
- Software processes and three general software process models
- The fundamental process activities
- How to cope with changes in the software requirements and design

Week 2: Agile Software Development
This week's objective is to introduce agile software development methods. Specific topics include:

- The rise of agile methods
- The differences between agile and plan-driven development
- Important agile development practices such as user stories, refactoring, pair programming, and test-first development, as exemplified in extreme programming (XP)
- The scrum approach to agile project management
- The issues of scaling agile development methods and combining agile approaches with plan-driven approaches in the development of large software systems

Week 3: Requirements Engineering
This week's objective is to introduce software requirements and to explain the processes involved in discovering and documenting these requirements. Specific topics include:

- The concepts of user and system requirements and why these requirements should be written in different ways
- The differences between functional and nonfunctional requirements
- The main requirements engineering activities, including elicitation, analysis, and validation, and the relationships between these activities

Weeks 4 and 5: System Modeling
These weeks' objective is to introduce system models that may be developed as part of requirements engineering and system design processes. Specific topics include:

- The rise of Unified Modeling Language (UML)
- How graphical models, such as UML, can be used to represent software systems
- The principal diagram types in UML, including:
    - Class diagrams
    - Use case diagrams
    - Sequence diagrams
    - Activity diagrams
    - State diagrams

- The fundamental system-modeling perspectives of context, interaction, structure, and behavior
- How to draw UML diagrams using Microsoft Visio

Weeks 6 and 7: Architectural Design
These weeks' objective is to introduce the concepts of software architecture and architectural design. Specific topics include:

- Why is the architectural design of software important?
- The decisions you have make about the software architecture during the architectural design process
- Common types of systems and their architectures
    - Data-processing systems
    - Transaction-processing systems
    - Language-processing systems
    - Event-processing systems
- Distributed systems
- The concept of architectural patterns, which are well-tried ways of organizing software architectures that can be reused in system designs
- Model-driven architecture, where an executable system is automatically generated from structural and behavioral models.

Week 8 and 9: Design and Implementation
These weeks' objective is to introduce the object-oriented design process using the UML and to highlight some important implementation issues. Specific topics include:

- The object-oriented design process
    - Define the context and modes of use of the system
    - Design the system architecture
    - Identify the principal system objects
    - Develop design models
    - Specify object interfaces
- Design patterns, which are a way of reusing design knowledge and experience
- Configuration management
- Host-target development
- Open source development
- The benefits and problems of software reuse
- The software reuse landscape, including the application framework as a set of objects that can be used in application development; software product lines, which are made up of a common core architecture and reusable components; and configurable off-the-shelf application software systems
- How to produce high-quality code

Weeks 10: Software Testing and Evolution
This week's objective is to introduce software testing, software testing processes, and software evolution processes. Specific topics include:

- The external and internal characteristics of software quality

- The stages of software testing from testing during development to acceptance testing by system customers, including unit testing, component testing, system testing, release testing, and user testing.
- Effectiveness of quality assurance (QA) techniques
- Software quality assurance tools
- Software change
- Legacy systems
- Software maintenance

Week 11: Team Project Presentations and the Final Exam

Grading:

Weekly essay questions: 30 percent (3 percent each week, 10 weeks)
Midterm exam (week 6): 15 percent
Midterm project presentation: 10 percent
Final exam (week 11): 15 percent
Final project presentation: 10 percent
Term project deliverables (programs + documentation): 20 percent