COURSE DESCRIPTION:

Successful software development depends on an in-depth understanding of how the phases and supporting activities of the software development life cycle work together. Each phase of the life cycle process contributes to a reliable, maintainable product that satisfies user requirements. The application of good engineering practices throughout the cycle dramatically improves the likelihood of delivering a quality software project on time, in scope and within budget. While there are many rigorous methodologies, in fact most approaches and tools have a mixture of strengths and weaknesses. Traditional development approaches result in models that are incomplete and quickly become out-of-sync with the application source code. Many modeling approaches focus on describing software designs, rather than solving business problems. Curiously, the various process models, including traditional SDLCs, Agile methods, DevOps, Design Thinking, Lean Startup, and others do not seem to provide a reliable all-encompassing end-to-end solution to software development. Therefore, it is important to understand how the various methods differ and/or complement each other.

This course presents modern software engineering techniques and examines the software development life-cycle (SDLC), including software specification, design, implementation, testing and maintenance. The course evaluates past and current trends in software development practices including agile software development methods, DevOps, and Site Reliability Engineering (SRE) as described below.

- Non-agile approaches that are still widely used in industry today and are covered in the course include the Rational Unified Process (RUP) and the Open Process Framework (OPF). Process improvement initiatives such as
the Capability Maturity Model (CMM) and Personal Software Process (PSP) will also be discussed.

- Agile methodologies involve continuous iteration of software development and testing as part of the SLDC process. It emphasizes iterative, incremental, and evolutionary development by breaking the product development process into smaller pieces and integrating them for final testing. Agile software engineering may be implemented in many ways, including Scrum, Kanban, Extreme Programming (XP), ASD, DSDM, Crystal, Feature Driven Development (FDD), and the Incremental Funding Method (IFM). Agile software processes promise strong productivity improvements, increased software quality, higher customer satisfaction and reduced developer turnover. Agile development techniques empower teams to overcome time-to-market pressures and volatile requirements. DevOps is a software development method that focuses on communication, integration, and collaboration among IT professionals to enable rapid deployment of products.

- DevOps is a culture that promotes collaboration between the development and operations teams. It allows deploying code to production faster and in an automated way to help increase an organization’s speed to deliver applications and services. DevOps may be defined as an alignment method between the development and IT operations teams.

- SRE and DevOps are not so different from one another. According to Google, the creator of SRE, both DevOps and SRE focus on closing the gap between the development and operations teams. While DevOps is all about the “what” needs to be done to close the gap, SRE focuses more on “how” this can be achieved. SRE expands the “theoretical” part to an efficient workflow, with the right work methods, tools, and so on. SRE is also about sharing the responsibility between everyone and getting everyone in sync with the same goal and vision.

The course gives an overview of methods and techniques used in agile software development and DevOps/SRE processes. It outlines the differences between these more recent approaches and contrasts them with traditional software development methods while discussing the sweet spots of the various methodologies. In particular, while modeling appears to be
fundamentally in contradiction with the goals of agile and DevOps, the course focuses on demonstrating how model-driven engineering (MDE) can contribute to DevOps. Conversely, given the important role of MDE in the conception and development of complex systems, the course demonstrates how DevOps helps in the continuous integration and evolution of modeling artifacts themselves.

This course is designed for anyone interested in learning how to understand requirements, specify solutions for complex systems, and deploy scalable, portable, and robust enterprise applications. The course will present a variety of tools, in the context of team production of publicly releasable software. The goal will be for each student to have had a hand in building complete and useful applications that could be released for real-world use.

This course is a highly interactive course, in which students are expected to fully participate in class-based activities and discussions. Students will be encouraged to bring their own experiences to the discussion, as most of the topics being covered in this course are still considered open research topics. More than 50% of the course will be spend on student presentations, hands-on exercises, and practical software development as part of a team project.

COURSE OBJECTIVES

The objectives of the course are as follows:

1. Describe and compare the various mainstream software development methods and understand the context in which each approach, or portion of such, might be applicable.

2. Develop students’ critical skills to distinguish sound development practices from ad-hoc practices, judge which technique is most appropriate for solving large-scale software problems, and articulate the benefits of applying sound practices.

3. Expand students’ familiarity with mainstream modeling approaches (e.g., agile modeling, design thinking, model-driven engineering) and languages used to model and analyze processes and object designs (e.g., UML and associated notations).
4. Demonstrate the importance of formal and executable specifications of object models, and the ability to verify the correctness and completeness of the solution by executing the models.

5. Explain the scope of the software maintenance problem and demonstrate the use of several tools for reverse engineering software.

6. Develop students’ ability to evaluate the effectiveness of an organization’s software development practices, suggest improvements, and define a process improvement strategy.

7. Introduce state-of-the-art tools and techniques for large-scale software systems development.

8. Implement the major software development methods in practical projects.

REQUIRED TEXTBOOKS

Software Engineering: A Practitioner's Approach
By Roger S. Pressman and Bruce R. Maxim

RECOMMENDED TEXTBOOKS

Software Engineering (10th Edition)
by Ian Sommerville

Agile Software Development Ecosystems
Jim Highsmith
Addison Wesley; ISBN-10: 0201760436; 1st edition (04/02)

Extreme Programming Explained: Embrace Change
Kent Beck

Agile Modeling: Effective Practices for Extreme Programming and the Unified Process
Scott Amber
PREREQUISITES

Students enrolling in this class are expected to have taken CSCI-GA.2110-001 (i.e., Programming Languages) and CSCI-GA.2250-001 (i.e., Design of Operating Systems) and their prerequisites. Knowledge of UML or a specific programming language is not required. For some of the practical aspects of the course, a working knowledge of an object-oriented programming language (e.g., Java) is recommended. Experience as a software development team member in the role of business analyst, (DevOps) architect, developer, or project manager is a plus.

TEAM PROJECT

All assignments (other than the individual assessments) will correspond to milestones in the team project. All team projects will involve multiple distributed sub-systems based on combinations of n-tier, P2P, or decentralized software architectures. Students may need to come up to speed with the programming languages and software tools that are applicable to their projects. Some of these software tools may not be covered in class.

Students will be required to gather into at most two "pairs"; if there is an odd number of students in the class, then one (1) team of three (3) members will be permitted. There may not be any "pairs" of only one member. The instructor (and TA) will then assist students with forming "teams", ideally each consisting of two (2) "pairs", but students are encouraged to form their own teams in advance. Optimally, students will develop and test their project code together with the other member of their programming pair.

WEB SITES

Related information can be found on the following Web sites:
OTHER RECOMMENDATIONS

Students are encouraged to review the references provided on the course Web site, subscribe to Application Development Trends, Code Magazine, and/or other relevant technology magazines and publications.

COURSE SESSIONS

1. SOFTWARE ENGINEERING FUNDAMENTALS

   - Course Logistics
   - Software Development Challenges
   - Software Scope
   - Software Engineering Discipline
   - Human Side of Software Development
   - Mainstream Software Engineering Methods

   **READINGS:** Selected readings assigned in class
                  Handouts posted on the course Web site

2. SOFTWARE DEVELOPMENT LIFECYCLES (SDLs)

   - Life Cycle Processes and Phases
   - Traditional Life Cycle Models
     - Waterfall
     - V
     - Phased
     - Evolutionary
     - Spiral
     - CBSE
     - etc.
   - Alternative Techniques
     - UP
- RAD
- JAD
- PSP/TSP
- Prototyping
- etc.
- Modern Processes and SDLCs
  - Agile Software Development
  - DevOps
  - SRE
  - Lean Startup Method
- Software Engineering Standards
  - Roles and Types of Standards
  - ISO 12207: Life Cycle Standard
  - IEEE SE Processes Standards and Specifications

**READINGS:**
Selected readings assigned in class
Handouts posted on the course Web site

3. **SOFTWARE ENGINEERING TOOLS**

- AIOpsAnalytics (e.g., Splunk, New Relic)
- Artifact/Package Management (e.g., Docker Hub, NPM)
- Cloud (e.g., Azure, AWS, GCP, IBM Cloud, OpenStack)
- Collaboration (e.g., Slack, Microsoft Teams)
- Configuration Automation (e.g., Ansible, Puppet)
- Containers (e.g., Docker, Kubernetes)
- Continuous Integration (e.g., Jenkins, Travis CI)
- Database Management (e.g., Idera, Dbmaestro)
- Deployment (e.g., Azure DevOps Pipelines)
- Enterprise Agile Planning (e.g., Planview, Rally)
- Issue Tracking / ITSM (e.g., Jira, Trello)
- Release Management (e.g., Cloudbees Flow)
- Security (e.g., Snort)
- Serverless/PaaS (e.g., AWS Lambdas, Cloud Foundry)
- Source Control Management (e.g., GitHub, GitLab SCM)
- Testing (e.g., Selenium, Junit)
- Value Stream Management (e.g., GitLab, Plutora)
- Selecting Appropriate Tools

4. **PLANNING AND MANAGING REQUIREMENTS**

- Requirements Development Methodology
- Specifying Requirements
- Eliciting Accurate Requirements
- Documenting Business Requirements
- Defining User Requirements
- Validating Requirements
- Achieving Requirements Traceability
- Managing Changing Requirements
- Reviews, Walkthroughs, and Inspections
- Agile Requirements Engineering

READINGS: Selected readings assigned in class
Handouts posted on the course Web site

5. **INTRODUCTION TO SOFTWARE ANALYSIS AND DESIGN**

- Software Architectures and Domains Modeling
- Roles of Analysis and Design
- Traditional Data and Process Modeling Approaches
- Performing Requirements Analysis
- Object-Oriented Modeling
- Model-Driven Engineering
- Agile Modeling
- Design Thinking
- Selecting and Combining Approaches
- Creating a Data Model

READINGS: Selected readings assigned in class
Handouts posted on the course web site

6. **BUSINESS MODEL ENGINEERING**

- Business Model Capture Tools
- Process Modeling
- Capturing the Organization and Location Aspects
- Developing a Process Model

READINGS: Selected readings assigned in class
Handouts posted on the course web site

7. **FROM ANALYSIS AND DESIGN TO SOFTWARE ARCHITECTURES**

- Building an Object Model using UML
- Design Patterns
- Architectural Patterns
- Model Driven Architectures
- Business Process Management
- Achieving Optimum-Quality Results
- Selecting Kits and Frameworks (e.g., Eclipse, etc.)
- Using Open Source versus Commercial Infrastructures
8. **BUILDING SOFTWARE**

- Language and Platform Issues
- Component Infrastructures
- Pair Programming
- Refactoring
- Test Driven Development (TDD)
- Distributed Development Using Agile Methods

9. **SOFTWARE VERIFICATION AND VALIDATION**

- Integration and System Testing
- Static Confirmation
- Dynamic Testing
- Traceability Matrices
- Automated Testing

10. **SOFTWARE QUALITY**

- Configuration Management (CM)
- Software Quality Assurance (SQA)
- Project Measurements
- Software Quality and Agile Methods
  - Automated and Manual Functional Testing
  - Acceptance testing
  - Mock objects
  - User interface testing (HTTPUnit, Canoo)
  - Performance testing
- Project Management Metrics
- Quality and Process Standards and Guidelines
  - ISO 9000
  - SWEBOK
  - ISO 15504
  - SEI’s Capability Maturity Model (CMM)
  - CMM Integration (CMMI)
11. RISK MANAGEMENT IN SOFTWARE ENGINEERING PROJECTS

- Project Planning and Estimation
- Cooperative roles of software engineering and project management
- Developing risk response strategies
- Risk Management in Agile Processes
- Agile Project Planning

READINGS: Expert One-on-One: Introduction (cont.)
Handouts posted on the course web site

12. QUANTIFYING SPECIFICATIONS USING FORMAL METHODS

- Using Set Theory and Logic
- Verifying Requirements Mathematically

READINGS: Selected readings assigned in class
Handouts posted on the course web site

13. OTHER PROJECT CONSIDERATIONS

- Evaluating Agile Software Engineering
- Software Engineering Ethics
- Web Engineering Techniques
- Measuring User Satisfaction

READINGS: Selected readings assigned in class
Handouts posted on the course web site

READINGS

Assigned readings for the course will be from the textbooks, various Software Engineering-related websites, trade magazines, and recommended books listed on the course Web site.

ASSIGNMENTS

Homework and project assignments.
Quizzes as applicable.
Final (take-home) exam.

GRADING POLICY
25% Assignments  
35% Projects  
30% Final Exam  
10% Attendance and Participation  
Extra credit will be granted periodically for particularly clever or creative solutions.

Team members who do not contribute appropriately to an assignment will receive a significantly lower grade for that assignment than the rest of the team.