NEW YORK UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
COURANT INSTITUTE OF MATHEMATICAL SCIENCES

SOFTWARE ENGINEERING

Spring 2020 - Jean-Claude FRANCHITTI

(CSCI-GA.2440-001 - Mon. 7:10 - 9:00 pm)

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 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.

This course presents modern software engineering techniques and examines the software life-cycle, including software specification, design, implementation, testing and maintenance. The course evaluates past and current trends in software development practices including agile software development methods such as Extreme Programming (XP), Agile Modeling (AM), Scrum, ASD, DSDM, Crystal, Feature Driven Development (FDD), Incremental Funding Method (IFM), DevOps, and Site Reliability Engineering. Agile software processes, DevOps, and SRE are the most recent trends in the software industry and 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. The course gives an overview of methods and techniques used in agile software processes, contrasts agile approaches with traditional software development methods, and discuss the sweet spots of both classes of methodologies. Other non-agile approaches that are widely used in industry such as the Rational Unified Process (RUP) and the Open Process Framework (OPF) will also be covered. Process improvement initiatives such as the Capability Maturity Model (CMM) and Personal Software Process (PSP) will be discussed.

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 focuses on applying modern software engineering techniques and standards to tackle the modeling of complex evolving

requirements, the architecting of secure ubiquitous responsive solutions that can mend the benefits of cloud-based, fog and edge distributed and decentralized architectural styles, the creation of quality solutions, and the management of software projects. 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 various software development methods and understand the context in which each approach might be applicable.

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

3. Expand students' familiarity with mainstream languages used to model and analyze object designs (e.g., UML).

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 Maxim
McGraw-Hill Higher International; ISBN-10: 1259872971; ISBN-13: 978-1259872976, 9th Edition
(09/19)

RECOMMENDED TEXTBOOKS

Software Engineering (10th Edition)
by Ian Sommerville
Pearson; ISBN-10: 0133943038; ISBN-13: 978-0133943030 (04/15)

The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations
by Gene Kin, Patrick Debois, John Willis, Jez Humble, and John Allspaw
IT Revolution Press; ISBN-10: 1942788002; ISBN-13: 978-1942788003 (10/16)

Site Reliability Engineering
by Niall Murphy, Betsy Beyer, Chris Jones, and Jennifer Petoff
O'Reilly Media; ISBN-10: 149192912X, ISBN-13: 978-1491929124 (04/16)

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, developer, or project manager is a plus.

TEAM PROJECT

All assignments (other than the individual assessments) will correspond to milestones in the team project. Teams will pick their own projects, within certain constraints: for instance, all projects should involve multiple distributed sub-systems (e.g., Mobile / Web-based services involving mashups of components and leveraging cloud, fog and edge, decentralized, as well as traditional solution architectures). Students will need to come up to speed on whatever programming languages and/or software technologies they choose for their projects - which will not necessarily be covered in class.

Students will be required to form themselves into "pairs" of exactly two (2) members each; 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(s)) will then assist the pairs in forming "teams", ideally each consisting of two (2) "pairs", possibly three (3) pairs if necessary due to enrollment, but students are encouraged to form their own 2-pair teams in advance. If some students drop the course, any remaining pair or team members may be arbitrarily reassigned to other pairs/teams at the discretion of the instructor (but are strongly encouraged to reform pairs/teams on their own). Students will develop and test their project code together with the other member of their programming pair.

REFERENCES

Related information can be found on the following Web sites:

https://www.ibm.com/cloud/devops
https://landing.google.com/sre
www.agilemanifesto.org
http://www.agilemodeling.com/
http://www.aboutus.org/Adaptivesd.com
www.AgileAlliance.org
http://www.extremeprogramming.org/
https://ronjeffries.com/
http://hillside.net/patterns
http://www.omg.org/mda/
http://www.stevemcconnell.com

Also see references provided on the course website. Also note that abundant additional references will be provided as part of the weekly session material.

OTHER RECOMMENDATIONS

Students are encouraged to review the references provided on the course Web site, and subscribe to online trade magazines such as Application Development Trends, Application Developer Magazine, Information Week, Java Magazine, etc.

COURSE SESSIONS

1. SOFTWARE ENGINEERING FUNDAMENTALS

- Course Logistics
- Software Development Challenges
- Software Scope

- Software Engineering Discipline
- Software Methodologies and Related Process Models
- The Human Side of Software Development
- Introduction to Agile Software Engineering

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

2. SOFTWARE DEVELOPMENT LIFE CYCLES (SDLCs) – Part I

- Process Models and Solution Life Cycle Phases
- Traditional Life Cycle Models
  - Waterfall
  - V
  - Phased
  - Evolutionary
  - Spiral
  - CBSE
- Alternative Techniques
  - UP
  - RAD
  - JAD
  - PSP/TSP
  - Prototyping

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

3. SOFTWARE DEVELOPMENT LIFE CYCLES (SDLCs) – Part II

- Agile Software Engineering Process Models
  - Extreme Programming
  - Agile Software Development
  - DevOps
  - Site Reliability Engineering (SRE)
- Roles and Types of Standards
- ISO 12207: Life Cycle Standard
- IEEE Standards for Software Engineering Processes and Specifications

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

4. SOFTWARE ENGINEERING TOOLS PRIMER

- Requirements Management Tools (e.g., IBM Rational Doors)

- Design Tools (e.g., Sparx Enterprise Architect)
- Development Tools
  - IDEs (e.g., Xcode, Eclipse, IntelliJ IDEA, NetBeans, Microsoft Visual Studio, Atom)
  - Source Control Management (e.g.,GitHub)
  - Release Orchestration (e.g., OpenMake)
  - Collaboration (e.g., Jira, Trello, Slack)
- Operations Management Tools
  - Database Automation (e.g., Datical)
  - Deployment (e.g, ElasticBox)
  - Configuration Management (e.g., Ansible, Chef, Puppet)
  - Continuous Integration (e.g, Jenkins)
  - Container Management (e.g., Docker, Kubernetes)
- Testing Tools and Frameworks
  - Testing Tools (e.g., Junit, Selenium)
  - PaaS (e.g., PythonAnywhere, AWS Code9, Heroku)
- Management and Monitoring Frameworks
  - - AIOps (e.g, Splunk, Logstash)
  - - Analytics (e.g., Dynatrace, ElasticSearch)
  - - Monitoring (e.g., Nagios)
- Security Frameworks (e.g., Snort, BlackDuck)
- Cloud Platforms (e.g., AWS, Azure, GCP, IBM Cloud)
- Project Management (e.g., Scoro, Basecamp, Microsoft Project)
- Selecting Appropriate Tools

5. 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
- Requirements Modeling
- Agile Requirements Engineering

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

6. INTRODUCTION TO SOFTWARE ANALYSIS AND DESIGN

- Roles of Analysis and Design

- Traditional Data and Process Modeling Approaches
- Performing Requirements Analysis
- Object-Oriented Modeling
- User Experience Design
- Design for Mobility
- Selecting and Combining Approaches
- Creating a Data Model

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

7. 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

8. FROM ANALYSIS AND DESIGN TO SOFTWARE ARCHITECTURES

- Building an Object Model using UML
- Architectural and Pattern-Based Design
- Model Driven Architectures
- Business Process Management
- Achieving Optimum-Quality Results
- Selecting Kits and Frameworks (e.g., Bootstrap, .Net)
- Using Open Source, free, freemium, paid, and Enterprise software components

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

9. BUILDING SOFTWARE

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

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

10. SOFTWARE VERIFICATION AND VALIDATION

- Unit Testing
- Integration and System Testing
- Static Confirmation
- Dynamic Testing
- Traceability Matrices
- Automated Testing
- Other Specialized Testing

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

11. SOFTWARE QUALITY AND SECURITY

- Software Quality Concepts
- Software Configuration Management (CM)
- Software Quality Assurance (SQA)
- Software Quality and Agile Methods
  - Automated and Manual Functional Testing
  - Acceptance testing
  - Mock objects
  - User interface testing (HTTPUnit, Canoo)
  - Performance testing
- Software Metrics and Analytics
- Quality and Process Standards and Guidelines
  - ISO 9000
  - SWEBOK
  - ISO 15504
  - SEI's Capability Maturity Model (CMM)
  - CMM Integration (CMMI)
- Software Security Engineering

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

12. RISK MANAGEMENT IN SOFTWARE ENGINEERING PROJECTS

- Project Management Concepts
- Project Planning and Estimation
- Cooperative roles of software engineering and project management

- Developing risk response strategies
- Risk Management in Agile Processes
- Agile Project Planning
- Project Management Metrics
- Software Support Strategies

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

13. Advanced Topics

- Software Process Improvement
- Quantifying Software Specifications Using Formal Methods
  - Using Set Theory and Logic
  - Verifying Requirements Mathematically
- Emerging Trends in Software Engineering
- Data Science for Software Engineers
- Measuring User Satisfaction
- Software Engineering Ethics

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

COURSE READINGS

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

ASSIGNMENTS

- Homework and project assignments completion will be required.
- Quizzes will be administered.
- The final exam will be a 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.

Pair or team members who do not contribute appropriately to an assignment will receive a significantly lower grade for that assignment than the rest of that pair/team, possibly "zero", at the discretion of the instructor. If there is lack of appropriate

contribution on any two or more group assignments, the non-participating student(s) may be asked to withdraw from the course. If this occurs after drop date, "unofficial withdrawal" grades may be given.