Computer Science Department
http://cs.salemstate.edu

CSC 300 Software Engineering I — 4 cr. W-II

Catalog description:

This course will explore classic and modern software engineering principles and paradigms used to optimize the software development process for large software systems. Topics include software life cycle models, tools and techniques for software engineering and software development, the software development life cycle, historic and modern process paradigms, testing/evaluation techniques, and evaluation metrics. Group and individual design projects will be used to gain understanding of course topics and experience with development tools and team dynamics; writing experiences will be used to develop skills in analysis and rhetoric. Three lecture hours and three hours of scheduled laboratory per week, plus additional work outside of class.

Prerequisites: CSC 260 and a W-I course.

Course Narrative:

"Software engineering" is a widely used but widely misunderstood term, often conflated with "software development", "programming", and "drowning in documentation". A few definitions can help to clarify and differentiate amongst these terms:

- Computer *programming* is the process of writing and maintaining source code. The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solving a given problem. The process of programming thus often requires expertise in multiple subject areas, including knowledge of the application domain, specialized algorithms, formal logic, and programming language syntax.
- Software *development* encompasses all of the activities involved in creating and maintaining a software product. The term "software development" is commonly used to refer to the activity of computer programming (see above), but in the professional computing field, usage of the term includes all of the activities and artifacts that are involved from the initial proposal for the desired software through to the final completed product. Therefore, software development may include research, analysis, multiple forms of writing (formal and informal proposals, presentation of findings, client interaction), design, creation of new software code, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products. Computer programming is a significant *component* of software development.
- Software *engineering* is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering principles to software development. In less formal terms, it is the act of using patterns and insights to conceive, model and appropriately scale a solution to a problem.

This course presents the fundamental issues underlying the process of developing software in general and in particular the process of developing and maintaining large-scale software products. The history and evolution of software engineering is discussed as a way of providing the background necessary to understand the current state of the field; current state-of-practice paradigms and formal patterns are then presented as mechanisms that can be employed to implement client requirements in such a way as to produce robust, efficient, scalable, and cost-effective software.

While this course does introduce a considerable amount of software technology, specifically including test automation, CASE tools, and artifact management, that technology is not an "end unto itself" but is rather a means to gain understanding and appreciation of the underlying principles that the technology largely automates. The focus throughout the course is consistently on the *processes* that underlay the effort to engineer software, with the technology that is introduced intended to be exemplars of the tools available for, and indeed necessary for, the creation of large-scale software products.

Student activities relating to Written Communication - Level II criteria are found throughout the course and are intimately integrated into the learning process. All projects, whether software development, proposal design, or formal writing exercises, are accompanied by specific objectives and assessment rubrics; students are given the opportunity to make multiple submissions for most exercises and all formal writing assignments, with each submission receiving feedback from the instructor; group projects and selected individual assignments will incorporate peer review. Lab sessions regularly include the opportunity for students, working together and with the instructor, to review work and to discuss the principles underlying their writing efforts. Class assignments include a wide assortment of activities designed to assist students in understanding software *engineering* and how it relates to software *development* and to *programming*. Well over 50% of the final grade for the course is based on writing as it is commonly practiced within the field of computer science in general and software engineering in particular.

Course Goals:

The purpose of this course is to develop students' understanding of modern methodologies, processes and techniques encountered in the development of large-scale software systems. The goals of this course are:

CG01: to develop an appreciation for the process of large-scale software development;
CG02: to develop the skills and knowledge necessary to analyze, design, verify and document large software systems;
CG03: to give students experience in making and critiquing presentations;
CG04: to give students experience in team software development activities;
CG05: to develop students' writing skills in the context of all aspects of the software engineering process;
CG06: use written assignments and class discussion to teach students to write effectively for various purposes and audiences;
CG07: to have students experience writing as a process.

Upon completion of the course, a student should:

- know the activities and techniques necessary to conduct the development of a large system;
- be able to select and apply the appropriate tools required to effectively implement the development process;
- be able to appropriately convey information about the results of the development process to all stakeholders in the development effort (specifically including clients and users).

Course Outcomes (Objectives):

Upon successful completion of the course, student will have:

CO01: demonstrated knowledge of the software development life cycle and its aspects and phases;
CO02: demonstrated knowledge of the major models used in the development of large-scale software;
CO03: demonstrated an appreciation of the factors effecting team selection and performance;
CO04: demonstrated knowledge of the tools and techniques of software development, specifically including UML;
CO05: demonstrated knowledge of modern process paradigms, specifically including agile practices and the Unified Process;
CO06: properly utilized a modern CASE tool environment, specifically including round-trip code engineering and UML modeling;
CO07: developed and executed a plan for product testing and evaluation;
CO08: participated in the development and presentation of group projects;
CO09: demonstrated the ability to critically analyze materials ranging from project proposals to technical specifications to scholarly research and to express this analysis clearly in both spoken and written form for a variety of appropriate audiences;
CO10: demonstrated recognition of the need for ongoing professional development through research into future trends in the area of software engineering;
CO11: demonstrated an understanding of writing as a process by giving and responding to feedback and reflecting on his/her own writing processes.

Student Outcome vs. Course Objectives matrix

Student Outcome (condensed form)	**CO01**	**CO02**	**CO03**	**CO04**	**CO05**	**CO06**	**CO07**	**CO08**	**CO09**	**CO10**	**CO11**
SO-1 Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SO-2 Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
SO-3 Communicate effectively in a variety of professional contexts.	✓	✓	✓	✓	✓	✓		✓	✓		✓
SO-4 Recognize professional responsibilities and make informed judgements in computing practice based on legal and ethical principles.			✓				✓	✓	✓	✓	
SO-5 Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SO-6 Apply computer science theory and software development fundamentals to produce computing-based solutions.	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓

Note:

SO-1 Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.
SO-2 Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.
SO-3 Communicate effectively in a variety of professional contexts.
SO-4 Recognize professional responsibilities and make informed judgements in computing practice based on legal and ethical principles.
SO-5 Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
SO-6 Apply computer science theory and software development fundamentals to produce computing-based solutions.

Topics:

Software Engineering

- Scope of Software Engineering **IM1(2), PF7(0.5), SE5(0.5)**
  - historical and economic aspects of Software Engineering
  - "workflow" vs. "phase" - definitions, uncoupling of tasks from when or where the tasks occur
- social and professional context of software development: ACM/IEEE Code of Ethics and Professional Practice – codified professional responsibilities **SP1(0.5),SP2(1.0),SP4(2.0),SP5(0.5),SP6(1.5),SP7(0.5)**,SP9(0.5)
  - legal, ethical, and social issues and professional responsibilities
  -
  - impact of computing on individuals, organizations, and society, at local and global levels
- Software Life Cycle Models **HC1(1), SE4(3),SE8(1)**
  - Historical models in context - code and fix, waterfall, rapid prototyping, iteration and incrementation,
  - synchronize and stabilize, spiralagile processes
  - Unified Process
  - specialized model: synchronize and stabilize, spiral
  - iteration and incrementation
- Software Development Process **PF6(1.5), PF8(0.5), HC1(1), IM1(1),IM2(0.5),IM3(3), SE1(0.5),SE3(1),SE5(0.5),SE7(0.5),SE8(1), SP3(0.5)**
  - "maintenance": historical (temporal) vs. current (functional) definitions
    - Rationale behind historical and current definitions
  - model vs. process
    -
  -
  - Moving Target problem
  - No Silver Bullet – Essence and Accident in Software Engineering
  - workflows: activities / tasks that *may* take place *in parallel* (but ideally don't)
    - Requirements, Analysis, Design, Implementation, Testing
  - phases: activities / tasks that are expected to take place *in sequence*:
    - Inception, Elaboration, Construction, Transition
  - one vs. two dimensional life cycle models
  - process maturity models: Capability Maturity Models (CMM) and ISO standards
- Software Development Teams **SE1(0.5),SE2(0.5),SE8(0.5)**,SE9(0.5)
  - team organization paradigms: chief programmer, democratic, agile, others
  - choosing an appropriate team organization
  - social and ethical issues
- Tools of the trade **IM1(0.5),IM2(2),IM3(1.5), PF8(0.5) SE2(0.5),SE3(1.5),SE6(1),SE7(0.5)**,SE11(0.5), **SP3(1.0)**
  - conceptual tools
    - Magic Number 7, +/- 2
    - stepwise refinement, design by contract, single responsibility, cost-benefit analysis, software metrics
  - CASE tools
    - taxonomy, scope
  - version control and configuration control systems
  - the role of CASE technology in software development

    - social and ethical issues
- Object-Oriented paradigm vs. Structured Programming paradigm
    - effects on requirements, analysis, and design workflows
    - effects on implementation and testing workflows
- Requirements (need to move BoK times from above)
    - Objectives
    - Strategies and Techniques
    - Expected outcomes
- Analysis (need to move BoK times from above)
    - Objectives
    - Strategies and Techniques
        - How OO influences design activities and outcomes *during analysis*
    - Expected outcomes
- Design (need to move BoK times from above)
    - Objectives
    - Strategies and Techniques
        - How OO influences design activities and outcomes
        - How OO influences implementation activities and outcomes *during design*
    - Expected outcomes
- Implementation
    - Objectives
    - Expected outcomes
- Testing **PF7 (0.5), SE2(0.5),SE3(0.5),SE5(0.5),SE6(1)**,SE11(0.5), **SP5(0.5)**
    - software quality assurance
    - quality issues
        - legal definition of quality
        - client definition of quality
    - non-execution-based testing
        - walkthroughs, inspections
    - execution-based testing
        - black-box, white-box (clear-box)
        - automated testing
        - regression testing
    - focus of testing: correctness, utility, reliability, robustness, performance
    - role of quality assurance in testing
    - social and ethical issues
- Modules and Class/Object Design **PF7(0.5), PL5(0.5), SE1(0.5)**
    - definition of a module
    - cohesion
    - coupling
    - encapsulation of data
    - Abstract Data Types (ADTs)
    - information hiding
    - object-oriented vs. non-object-oriented design
- Reusability **SE7(0.5), SP3(0.5),SP4(0.5),SP5(0.5),SP7(0.5)**,SP10(0.5)
    - reuse concepts and degrees
    - impediments to reuse - what makes code hard to re-use
    - classes/objects in the context of reuse
    - reuse during design and implementation
        - application frameworks, toolkits, formal and informal design patterns, software architecture, component-based software engineering
    - social and ethical issues
- Portability **SE2(0.5)**,SE9(0.5),SE12(0.5)
    - hardware and operating system incompatibilities, techniques for addressing portability needs
- Planning and Estimating **SE8(1.5)**
    - estimating time and cost
        - metrics for size; techniques for cost estimation; COCOMO, COCOMO II; tracking estimates
    - project management plan components
        - IEEE Software Project Management Plan

- planning for testing
- establishing documentation standards
- CASE tools for Planning and Estimating

Writing

- Writing as a process and the importance of reflection and revision
  - Feedback mechanisms
- Use of Rubrics
  - Traits (aspects of writing effectively)
  - Ratings within traits
  - Rubrics as guidelines
- Writing for specific audiences
  - Clients (focus: functionality, non-technical descriptions)
  - Users (focus: form of functionality, relationship to everyday tasks)
  - Designers and Developers (focus: functionality, non-functional constraints, technical precision)
- Typical genres
  - Business communication with clients and users
  - Technical communication with designers and developers
  - Formal specification of project functionality, non-functional constraints
  - Storyline evolution
  - Prototyping
  - Code Review sessions
  - Project post mortem (retrospective analysis of project process and results)

Student Experiences:

The emphasis of the course is on state-of-practice methodologies and patterns, the application of which will result in cost-effective and efficient development of software systems that meet client needs and expectations and are scalable and maintainable.

Extensive laboratory exercises, group discussion time (specifically including time focused on writing) and group presentations conducted as part of the scheduled laboratory sessions are an integral component of the course, serving to reinforce the concepts and techniques presented in lecture. Presentations will be assessed based on the *Presentation* rubric; at least one group project / presentation will focus primarily on the professional, ethical, and social responsibilities of professionals in the field of software engineering / computing.

All programming projects must conform to departmental guidelines for program design and implementation (see the *Analysis* and *Project Implementation* rubrics for guidelines), and all lab reports must conform to guidelines announced in class. Multiple projects will be group projects. Regardless of numeric average, a student will not be eligible for a passing grade unless a lab report that complies with specified standards has been submitted for every programming project.

Regular writing assignments based on assigned articles and internet research will serve to broaden students' exposure to recent developments in the field and to professional, social, legal, and ethical aspects of software engineering. Writing activities will include but will not be limited to:

- review of technical articles;
- presentation of research findings;
- project proposals;
- design and creation of use case scenarios, storyboards, and test cases;
- at least one case study to be analyzed and discussed from the perspective of professional responsibilities and potential local and global effects
- analysis and evaluation of methodologies.

Specific requirements for each assignment will be stated and discussed when the assignment is distributed; all submissions will be graded against the *Writing* rubrics. At least three writing assignments will require the submission of a preliminary draft, which will be reviewed by the course instructor and feedback supplied to provide guidance on improving subsequent submissions.

The course grade will be determined using the following approximate weights: project reports and deliverables (source code, solution designs, test results, etc.): 15%; presentations: 10%; quizzes, midterm exam, and final exam: 45%; homework: 10%; writing assignments: 20%.

Student Experiences by Course Outcome (Objective) matrix:

student outcome / experience	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11
exam questions	✓	✓	✓	✓	✓		✓		✓	✓	✓
homework problems from text	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
project reports and deliverables	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
lab exercises	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
quizzes	✓	✓	✓	✓	✓		✓				
writing assignments	✓	✓		✓	✓				✓	✓	✓
group projects	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
presentations	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Sample Bibliography:

Web Resources:

Agile Modeling (AM) Home Page: Effective Practices for Modeling and Documentation. http://www.agilemodeling.com/

Association for Computing Machinery (ACM). http://www.acm.org/

The Institute of Electrical and Electronics Engineers (IEEE). http://www.ieee.org/portal/site

The Research Center on Computing and Society. http://www.southernct.edu/organizations/rccs

Bibliography

Beck, Kent; Andres, Cynthia. **Extreme Programming Explained: Embrace Change. Second Edition.** Addison-Wesley Professional, 2004.

Beck, Kent. **Implementation Patterns. Second Edition.** Addison-Wesley Professional, 2007.

Booch, Grady; Rumbaugh, James; Jacobson, Ivar. **The Unified Modeling Language User Guide. Second Edition.** Addison-Wesley, 2005.

Bruegge, Bernd; Duto1t, Allen. **Object-Oriented Software Engineering: Using UML, Patterns and Java. Third Edition.** Prentice-Hall, 2009.

Bynum, Terrell W. (editor); Rogerson, Simon. **Computer Ethics and Professional Responsibility: Introductory Text and Readings**. Blackwell,2008.

Freeman, Freeman, Bates & Sierra. **Head First Design Patterns.** O'Reilly Media, 2004.

Fowler, Martin, with Kenneth Scott. **UML Distilled: A Brief Guide to the Standard Object Modeling Language. Third Edition.** Addison-Wesley, 2003.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995.

Hoffer, et. al. **Modern Systems Analysis & Design. Seventh Edition.** Prentice-Hall,2013.

Jacobson, Lawson, and Ng. The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons! ACM Books, 2019.

Pfleeger, Shari Lawrence; Atlee, Joanne. **Software Engineering: Theory and Practice. Fourth Edition.** Prentice Hall, 2009.

Pressman, Roger S. **Software Engineering: A Practitioner's Approach. Eighth Edition.** McGraw-Hill, 2019.

Quinn, Michael. **Ethics for the Information Age**. **Seventh Edition.**Pearson,2016.

Schach, Stephen R. **Classical and Object-Oriented Software Engineering. Eighth Edition.** McGraw-Hill, 2011.

Shalloway, Alan; Trott, James. **Design Patterns Explained: A New Perspective on Object-Oriented Design. Second Edition.** Addison-Wesley, 2004.

Sommerville, Ian. **Software Engineering. Tenth Edition.** Addison-Wesley,2018.

Tavani, Herman T. editor. **Ethics, Computing and Genomics**. Jones and Bartlett, 2006.

West, David. **Head Object-Oriented Analysis and Design.** O'Rielly Media, 2011.

Zobel, Justin. **Writing for Computer Science. Third Edition**. Springer,2015.