Computer Science Department
cs.salemstate.edu

CSC 351 Software Engineering II (formerly CSC 301) **4 cr.**

Instructor: TBA **Office:** location **Phone:** (978) 542-extension
email: TBA@salemstate.edu **Office Hours:** days and times

Section	Time	Room	Final Exam
nn	days and times	location	date and time
Lnn	days and times	location	

Catalog description:

This course is an extension of CSC 300 and focuses on the implementation of the software engineering principles covered therein. It will explore state-of-practice and cutting-edge techniques and tools related to the design, implementation and maintenance of software systems. Topics include: design patterns; Model Driven Architecture (MDA); test-driven development; agile development; extreme programming (XP); aspect-oriented design. An ongoing group project will be used to gain practical experience with current software engineering practices and a variety of IDEs and CASE tools. Three lecture hours per week and three hours of scheduled laboratory per week, plus programming work outside of class. Not open to students who have received credit for CSC 301.

Prerequisite: CSC 300; CSC 263 strongly recommended.

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:

- **CG1:** to give students experience with a variety of software engineering techniques and paradigms;
- **CG2:** to expand and integrate students' knowledge and skills in the areas of system analysis and software design, implementation and verification;
- **CG3:** to give students experience in making and critiquing presentations;
- **CG4:** to give students experience in team software development.

Upon completion of the course, a student should have experience with a variety of the activities and techniques necessary to conduct the development of a large system, should be able to select and apply the appropriate tools required to effect the development process, and should have an appreciation of the strengths and weaknesses of the various design and implementation models extant in the field.

Objectives:

Upon successful completion of the course, student will have:

- **CO1:** demonstrated understanding of the software development life cycle and its phases;
- **CO2:** demonstrated knowledge of the major techniques and models used in the implementation of each phase (workflow) of software development;
- **CO3:** gained experience with the tools and techniques of software development;
- **CO4:** demonstrated understanding of modern design paradigms;
- **CO5:** properly utilized modern CASE tool environments, specifically including UML modeling, in the design and implementation of a large-scale project;
- **CO6:** participated in the development and presentation of group projects.

Program Outcome vs. Course Objectives matrix

Program Objective (condensed form)	CO01	CO02	CO03	CO04	CO05	CO06
PO-A: apply knowledge of computing and math	✓	✓	✓	✓	✓	✓
PO-B: analyze a problem and define its computing requirements	✓	✓	✓	✓	✓	✓
PO-C: design, implement and evaluate applications			✓		✓	✓
PO-D: function effectively in teams to accomplish a common goal			✓	✓	✓	✓
PO-E: professional, ethical, and social responsibilities		✓		✓	✓	✓
PO-F: communicate effectively with a range of audiences		✓	✓	✓	✓	✓
PO-G: local and global impact of computing on people and society						
PO-H: need for continuing professional development			✓		✓	
PO-I: use current techniques, skills, and tools	✓	✓	✓	✓	✓	
PO-J: apply theory and principles to model and design systems				✓	✓	
PO-K: apply design and development principles in constructing software	✓	✓	✓	✓	✓	

note - full statements of the Program Outcomes (program objectives) for the Computer Science Major can be found in the document *Computer Science Major Program Educational Objectives and Program Outcomes* on the Assessment page of the Computer Science Major (cs.salemstate.edu)

Topics:

- Requirements — **SE5(3),SE6(0.5),SE7(1),SE8(1)**,SE11(0.5), **SP1(0.5),SP2(0.5),SP4(1),SP5(0.5)**,SP9(0.5)
  - determining user needs
  - distinguishing "needs" and "wants"
  - overview of the requirements workflow
  - defining scope
  - understanding the domain
  - requirement elicitation techniques
    - interviewing, forms collection, use cases, prototyping
  - test workflow in the context of requirements
  - human factors
  - prototypes and reuse
  - metrics for the requirements workflow
    - what to measure, how to evaluate
- Object-Oriented (OO) Analysis — **IM1(1), SE1(1),SE2(0.5),SE7(0.5), SP1(0.5),SP3(0.5),SP5(0.5)**
  - overview of the analysis workflow
  - OO analysis overview
  - recognizing entity (data) classes
  - entity modeling
    - noun extraction
    - CRC cards
  - functional modeling
  - dynamic modeling
  - test workflow in the context of OO analysis
  - interface class extraction
  - control class extraction
  - use cases and dynamic modeling
  - the specification document in OO analysis
  - CASE tools for OO analysis

- metrics for the OO analysis workflow

- Classical ("Structured") Analysis **IM1(2), SE1(0.5),SE7(0.5),SE8(5)**,SE10(1), **SP1(1)**
  - structured analysis overview
  - informal specifications
  - the Specification Document
  - structured systems analysis
    - data flow diagrams, alternative techniques
  - entity-relationship (ER) modeling
  - finite state machines
  - other formal techniques, including petri nets, Z, and Anna
  - test workflow in the context of classical analysis
  - the specification document in OO analysis
  - CASE tools for structured analysis
  - metrics for the structured analysis workflow
- Design **IM3(1), SE1(1),SE3(2),SE6(0.5), SE7(0.5)**,SE11(0.5), **SP3(1),SP4(0.5),SP6(0.5)**
  - design and abstraction
  - overview of the design workflow
  - operation-oriented (function-oriented) design
  - data flow analysis
  - transaction analysis
  - data-oriented design
  - object-oriented design
  - test workflow in the context of design
  - real-time design techniques
  - CASE tools for design
  - metrics for the design workflow
- Design Patterns **PL5(1), SE1(6),SE4(1)**
  - what is a design pattern?
  - design patterns solve *design* problems
  - design by contract / programming to an interface
  - design with change in mind
  - toolkits
  - frameworks
  - foundation creational patterns
    - abstract factory, builder, factory method, prototype, singleton
  - foundation structural patterns
    - adapter, bridge, composite, decorator, facade, flyweight, proxy
  - foundation behavioral patterns
    - chain of responsibility, command, interpreter, iterator, mediator, memento, observer, state, strategy, template method, visitor
  - how to select design patterns
    - understand that design patterns are *abstractions*
    - know each pattern's *intent*
    - know how patterns *interrelate*
      - → non-trivial problems are likely to require multiple patterns
    - know how *similar* patterns *differ*
    - know the causes for *redesign* (refactoring) and consider patterns designed to avoid those causes
  - what to expect from patterns
  - implementing design patterns
  - CASE tools for design patterns
- Implementation **IM2(1),IM3(1)**,IM4(1),IM6(1),IM7(1), **SE2(1),SE3(2),SE5(0.5),SE6(2),SE7(0.5)**,SE11(1)**, SP4(0.5),SP5(0.5),SP6(0.5)**,SP9(0.5)
  - overview of the implementation workflow
  - choosing a programming language / platform

- good programming practices
  - mnemonic names, self-documenting code, formatting, general style rules
- coding standards
- code reuse
  - licensing / intellectual property issues
- unit integration
- test workflow in the context of implementation
  - testing to specifications (black box)
  - testing to code (white box, glass box)
  - theory vs. reality of testing
- black box testing techniques
- glass box testing techniques
- unit testing
- regression testing
- code walkthroughs and code inspections
- potential testing problems
- when to rewrite vs. debug
- integration testing
- product testing
- acceptance testing
- CASE tools for implementation, testing and code/configuration management
- metrics for the implementation workflow

- Maintenance, Post-Delivery **SE3(1),SE5(1),SE6(0.5),SE7(1),SE8(0.5)**,SE11(1), **SP4(0.5)**
  - necessity for post-delivery maintenance
  - post-delivery maintenance skills requirements vs. development skills
  - management of post-delivery maintenance
  - maintenance of OO software vs. classical ("structured") software
  - reverse engineering
  - testing during post-delivery maintenances
    - regression testing revisited
  - CASE tools for post-delivery maintenance

The emphasis of the course is on the proper design, management and implementation of a software system from initial conception to final product maintenance. There will be an ongoing case study presented in depth, paralleled by a semester-long project in which all phases of the creation of a moderate-sized system will be addressed by groups within the class. Extensive laboratory work, group discussion time 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.

All programs must conform to departmental guidelines for program design and implementation, and all lab reports must conform to guidelines announced in class. Regardless of numeric average, a student will not be eligible for a passing grade unless he or she has submitted a lab report for every programming assignment.

The course grade will be determined using the following approximate weights: project reports and deliverables: 25%; presentations: 10%; midterm and final exam: 40%; homework and/or papers: 25%.

Course Objective / Assessment Mechanism matrix

	Exam / Quiz Questions	Homework Problems	Programming Projects	Lab Exercises	Group Projects
CO01	✓	✓	✓	✓	✓
CO02	✓	✓	✓	✓	✓
CO03	✓	✓	✓	✓	✓
CO04	✓	✓	✓	✓	✓
CO05		✓	✓	✓	✓

CO06			✓	✓	✓

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

Bibliography:

Ambler, Scott. **The Object Primer: Agile Model-Driven Development with UML 2.0. Third Edition.** Cambridge University Press, 2004.

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

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.

Coplien, James; Harrison, Neil. **Organizational Patterns of Agile Software Development**. Prentice Hall, 2005.

Dennis, Alan; Wixom, Barabar; Tegarden, David. **Systems Analysis and Design with UML Version 2.0 : An Object-Oriented Approach . Third Edition.** John Wiley & Sons, 2007.

Dikel, David M.; Kane, David; Wilson, James R. **Software Architecture: Organizational Principles and Patterns.** Prentice Hall, 2001.

Fowler, Martin. **Analysis Patterns: Reusable Object Models.** Addison-Wesley, 1997.

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. Sixth Edition.** Prentice Hall, 2010.

Kerievsky, Joshua. **Refactoring to Patterns .** Addison-Wesley Professional, 2004.

Larman, Craig. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.** Pearson Education, 2005.

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

Pressman, Roger S. **Software Engineering: A Practitioner's Approach. Seventh Edition.** McGraw-Hill, 2009.

Sommerville, Ian. **Software Engineering. Ninth Edition.** Addison-Wesley, 2010

Schach, Stephen R. **Classical and Object-Oriented Software Engineering. Eighth Edition.** McGraw-Hill, 2010.

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

Zobel, Justin. **Writing for Computer Science. Second Edition**. Springer, 2004.

Academic Integrity Statement:

"Salem State University assumes that all students come to the University with serious educational intent and expects them to be mature, responsible individuals who will exhibit high standards of honesty and personal conduct in their academic life. All forms of academic dishonesty are considered to be serious offences against the University community. The University will apply sanctions when student conduct interferes with the University primary responsibility of ensuring its educational objectives." Consult the University catalog for further details on Academic Integrity Regulations and, in particular, the University definition of academic dishonesty.

The Academic Integrity Policy and Regulations can be found in the University Catalog and on the University website (http://catalog.salemstate.edu/content.php?catoid=13&navoid=1295#Academic_Integrity). The formal regulations are extensive and detailed - familiarize yourself with them if you have not previously done so. A concise summary of and direct quote from the regulations: "Materials (written or otherwise) submitted to fulfill academic requirements must represent a student's own efforts". *Submission of other's work as one's own without proper attribution is in direct violation of the University's Policy* and will be dealt with according to the University's formal Procedures. *Copying without attribution is considered cheating in an academic environment - simply put,* ***do not do it!***

University-Declared Critical Emergency Statement:

In the event of a university-declared emergency, Salem State University reserves the right to alter this course plan. Students should refer to www.salemstate.edu for further information and updates. The course attendance policy stays in effect until there is a university-declared critical emergency.

In the event of an emergency, please refer to the alternative educational plans for this course, which will be distributed via standing class communication protocols. Students should review the plans and act accordingly. Any required material that may be necessary will have been previously distributed to students electronically or will be made available as needed via email and/or Internet access.

Equal Access Statement:

"Salem State University is committed to providing equal access to the educational experience for all students in compliance with Section 504 of The Rehabilitation Act and The Americans with Disabilities Act and to providing all reasonable academic accommodations, aids and adjustments. **Any student who has a documented disability requiring an accommodation, aid or adjustment should speak with the instructor immediately.** Students with Disabilities who have not previously done so should provide documentation to and schedule an appointment with the Office for Students with Disabilities and obtain appropriate services."

Note: This syllabus represents the intended structure of the course for the semester. If changes are necessary, students will be notified in writing and via email.