COURSE DESCRIPTION

Dept., Number	CSci 387	Course Title	Software Design and Development
Semester hours	3	Course Coordinator	Charles M. Jenkins, Instructor

Current Catalog Description

Study of techniques for the construction of large, complex software systems, including project management, requirements analysis, specification, design, development, testing, documentation, deployment, and maintenance. Students develop software systems in a group structure that simulates an industrial setting.

Textbook

Stephen R. Schach, *Object–Oriented Software Engineering*, McGraw–Hill Higher Education, ISBN 978–0–07–352333–0, Boston, 2008.

References

Course Outcomes

This course introduces the student to software engineering, with emphasis on modern object–oriented methods. In addition, it exposes the student to the processes and challenges of managing and executing a software project as a member of a team. Specifically, upon successful completion of this course, the students are able to:

1. Participate effectively in the context of a small software development team.
2. Explain the economic and risk drivers behind modern software engineering.
3. Compare and contrast common software life–cycle models (e.g., iterative–and–incremental, waterfall, agile processes, etc.).
4. Explain and apply the workflows and phases of an iterative–and incremental life–cycle model.
5. Document the requirements, analysis, design, and deployment of a software product.
6. Explain and apply the tools of software engineering (e.g., CASE tools, cost–benefit analysis, and configuration control).
7. Explain and apply the techniques of execution– and non–execution–based testing.
8. Explain the properties of software modules (e.g., cohesion, coupling, encapsulation, and information hiding).
9. Explain abstract data types and the object–oriented paradigm.
10. Explain the importance of software reusability and portability.
11. Explain and apply software engineering metrics (e.g., measures of product size, quality,

and complexity).
12. Discuss the ethics of software engineering practice.

Relationship between Course Outcomes and Program Outcomes

The Course Outcomes contribute to the Program Outcomes as shown below:

1. Participate effectively in the context of a small software development team: Outcomes d, f
2. Explain the economic and risk drivers behind modern software engineering: Outcomes b, c, i, j
3. Compare and contrast common software life–cycle models. Outcomes b, c, k
4. Explain and apply the workflows and phases of an iterative–and incremental life–cycle model: Outcomes b, c, k
5. Document the requirements, analysis, design, and deployment of a software product: Outcomes b, c, j, k
6. Explain and apply the tools of software: Outcome i
7. Explain and apply the techniques of execution– and non–execution–based testing: Outcome c
8. Explain the properties of software modules: Outcomes b, c
9. Explain abstract data types and the object–oriented paradigm: Outcomes b, c
10. Explain the importance of software reusability and portability: Outcomes b, c, j, k
11. Explain and apply software engineering metrics: Outcomes b, c, i
12. Discuss the ethics of software engineering practice: Outcome e

Prerequisites by Topic

CSci 211, Computer Science III
CSci 223, Computer Organization and Assembly Language

Major Topics Covered in the Course

1. The scope of software engineering: historical and economic aspect; ethical issues
2. Iterative and incremental software life–cycle models: 2D and other life–cycle models
3. The Unified Process: workflows and artifacts; phases; continuous improvement
4. Developing software in teams: roles and responsibilities; accountability and trust; managing problems within a team
5. Tools and techniques: cost–benefit analysis, software and project metrics, CASE, and the Unified Modeling Language
6. Design concepts: cohesion and coupling; abstract data types, encapsulation, and information hiding; object–orientation, inheritance, polymorphism, and dynamic binding
7. Planning and estimating: overview of techniques (e.g., COCOMO II), the Software Project Management Plan
8. Implementation: choice of environment and language; programming practices and standards; integration and testing
9. Testing: scope; execution– and non–execution–based testing
10. Post–delivery maintenance: skills required; processes; management

Assessment Plan for the Course

A comprehensive, 30-question exam constructed by a faculty committee is administered to each offering of CSci 387. Student performance is analyzed question-by-question to identify needed adjustments in the textbook, lectures, or assignments. Faculty members participate in the evaluation, in the selection of textbooks, and in formulating a response appropriate to the assessment results.

Students complete individual, graded homework assignments and take examinations designed to measure their mastery of the theoretical concepts discussed in class.

Each student works with a three- to four-person team to construct a complete solution for a realistic computing problem. Teams deliver project documents (requirements documents, analysis documents, design documents, finished code, user and systems manuals, and so forth) for grading on a schedule established by the instructor. These deliverables are graded for content, for design quality, for grammatical correctness, and so forth. Each student also submits periodic evaluations of the performance of each member of his or her team.

At the end of the semester, each team orally presents its solution to the class. The instructor and students evaluate the presentations using a rubric designed by the instructor.

How Data in the Course are Used to Assess Program Outcomes (unless adequately covered already in the assessment discussion under Criterion 4)

The standard exam and the intra-team evaluations used in CSci 387 (see the previous item) are included in the curriculum-wide outcome assessment described in Chapters 2, 3, and 4 of the Self-Study.

Estimate Curriculum Category Content (Semester hours)

Area	Core	Advanced	Area	Core	Advanced
Algorithms			Software design	3	
Data structures			Concepts of programming languages		