CO2006 Software Engineering and System Development

Credits: *20* **Convenor:** *Dr A. Boronat* **Semester:** *1st*

Prerequisites:	*Essential: CO1003, CO1005, CO1019*		*Desirable: CO1001, CO1012*		
Assessment:	*Coursework: 100%*				
Lectures:	*12*	*hours*	**Class Tests:**	*2*	*hours*
Surgeries:	*10*	*hours*	**Private Study:**	*106*	*hours*
Laboratories:	*20*	*hours*			

Subject Knowledge

Aims According to a report of the British Computer Society, only about 16% of IT projects can be considered truly successful and over 60% of them experience severe problems. The difficulties of software development led to the coining of the phrase "the software crisis" and the birth of software engineering as a discipline. However, in many companies, software is still developed in an ad-hoc way. The purpose of this module is to teach object-oriented methods for analysis, specification, design, implementation, and testing of software systems.

Learning Outcomes At the end of this course, successful students will be able to: explain the main phases in a software development process; elicit and analyse customer requirements following an agile methodology; produce object-oriented system designs, by applying design patterns and architectural styles; use UML for consistent specification of software systems and business processes; incorporate security into specifications and designs by following a flexible security specification process; and use appropriate techniques for software development and testing, including mechanisms for software reuse.

Methods Lectures, lecture notes, surgeries, problem classes, recommended textbooks, worksheets, supervised laboratories, formative feedback and web resources.

Assessment Formative coursework, assessed class tests, take-home assignment and mini project.

Skills

Aims To teach students a range problem-solving skills tailored to SE, including knowledge acquisition and software modelling.

Learning Outcomes At the end of this course, successful students will be able to write short, clear summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen).

Methods Surgeries, problem classes, worksheets, supervised laboratories, formative feedback.

Assessment Formative coursework, assessed class tests, take-home assignment and mini project.

Explanation of Prerequisites

A sound knowledge of basic algorithms, data structures and programming is required. Some knowledge of database systems of basic web application development (HTML, CSS, Javascript) is desirable.

Course Description

This module introduces students to principles and methods used to specify, design, implement and test software systems. In particular, the object-oriented paradigm will be followed, and techniques therein.

Detailed Syllabus

Introduction: Introduction to software engineering; the inherent complexity of software; examples of complex systems; basic notions and techniques for modelling software systems.

Overview of the UML: Use-case diagrams; class diagrams; object diagrams; package diagrams; activity diagrams; sequence diagrams; statechart diagrams.

Requirements elicitation: Project glossary; actors, use cases and scenario-based design; functional and non-functional requirements; business models; requirements analysis documents.

Object-oriented analysis: Functional vs object-oriented modelling; model of an application domain; analysis of use cases; CRC cards; interaction diagrams; user interface specification; dynamic model; requirements validation and verification.

System design: Scope of system design; subsystem decomposition; encapsulation and information hiding; coupling and cohesion; layers and partitions; examples of software architectures; open and closed software architectures; architectural styles; design goals; vertical and horizontal prototypes.

Object design and implementation: system design vs object design; component diagrams; interface design; contracts; OCL; invariants; pre- and post-conditions; software reuse; delegation and inheritance; frameworks; design patterns; mapping object-oriented designs to code; container/collection classes in code.

Testing: fault vs failure; the oracle problem; test model; testing activities; V model; unit testing; white-box and black-box testing; JUnit; integration testing; test harness; system testing; acceptance testing.

Reading List

[A] B. Bruegge and A.H. Dutoit, *OO Software Engineering: Using UML, Patterns, and Java; 3rd edition,* Addison-Wesley, 2009.

[B] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Modeling Language User Guide; 2nd edition,* Addison-Wesley, 2005.

[B] S. L. Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice; 4th edition,* Prentice Hall, 2009.

[C] M. Fowler, *UML Distilled, 3rd edition,* Addison-Wesley, 2003.

[C] R. Pooley and P. Stevens, *Using UML: Software Engineering with Objects and Components; 2nd edition,* Addison-Wesley, 2005.

[C] I. Sommerville, *Software Engineering; 9th edition,* Pearson, 2010.

[C] R. Pressman and D. Ince, *Software Engineering: A Practitioner's Approach; 7th edition,* McGraw Hill, 2009.

Resources Lecture notes, module web page, study guide, worksheets, handouts, lecture rooms with data projector.

Module Evaluation Course questionnaires, course review.