CO7205 Advanced System Design

Credits: 15  Convenor: Prof. J. L. Fiadeiro  Semester: 1st

Prerequisites: none
Assessment: Coursework: 40%  2 hour exam in January: 60%
Lectures: 24 hours
Surgeries: 8 hours
Laboratories: 8 hours
Private Study: 72.5 hours

Subject Knowledge

Aims  In September 1994, an article in the Scientific American alerted to the ‘Software’s chronicle crisis’: The challenge of complexity is not only large but also growing. [...] To keep up with such demand, programmers will have to change the way that they work. “You can’t build skyscrapers using carpenters,” Curtis quips. In May 2003, an article published in The Economist points out that the ‘crisis’ endures: Computing has certainly got faster, smarter and cheaper, but it has also become much more complex. Ever since the orderly days of the mainframe, which allowed tight control of IT, computer systems have become ever more distributed, more heterogeneous and harder to manage. [...] In summary, complexity seems to be a recurrent issue in Software Engineering. The aim of this module is to introduce students to some of the principles, methods and techniques that are available to tackle the complexity of systems, with special emphasis in Software Architectures.

Learning Outcomes  At the end of the course, students should be able to: understand the difference between programming and designing applications, and the techniques that support them; understand the basic concepts and role of software architectures, including the separation between computation and coordination concerns; understand the principles of event-based, synchronous, and service-oriented architectures in particular; be able to model architectural connectors and systems according to these architectural styles.

Methods  Class sessions, tutorials and practical sessions together with course notes, recommended reading, worksheets, printed solutions, and some additional hand-outs.

Assessment  Assessed coursework, traditional written examination.

Skills

Aims  To teach students abstraction and higher-level modelling skills, with special emphasis on architectural views of systems; to develop in the students the ability to separate concerns during system design.

Learning Outcomes  Students will be able to: model an architectural view of a system using both event-based (asynchronous) and action-based (synchronous) interactions; map high-level architectures to design architectures over Java; externalise coordination mechanisms from Java-programmed computations.

Methods  Class sessions together with worksheets.

Explanation of Prerequisites  Experience in Programming in Object Oriented Paradigms (Java) as well as general program design skills will be helpful.

Course Description  The use of an object-oriented language like Java as the first language in computer science courses is by now well established worldwide. Such courses typically cover the basic aspects of object-oriented programming with emphasis on the control structures made available in the language. However, software engineering as a discipline requires another level of maturity to be attained: that of software design, which covers the use of languages like Java for the development of complete and high-quality applications. This level is concerned with the complexity that arises during software development, which comes in two kinds: the complexity that arises from the size of the application, which leads to its decomposition into modules and dependencies among them; and the one that arises from the need to account for a large number of interactions among autonomous entities. This module covers some of the techniques needed to cope with such levels of complexity: architectural
design, and coordination-based modelling.

**Detailed Syllabus**

**Software Engineering** Short history and background. Levels of abstraction: requirements, design and implementation. Complexity in software development: programming in-the-small, in-the-large, and in-the-world.

**Software Architectures** Physiological vs social complexity; architectures of usage vs interaction; components versus connectors; architectural styles; nature and role of architectural description languages.

**Modelling in-the-world** Separation of coordination from computation; externalisation of interactions as connectors; the CCC-approach to architectural modelling; the language CommUnity for action-based (synchronous) architectures; the language SRML for service-oriented architectures; combining Jackson’s Problem Frame approach to requirements engineering with architectural modelling.

**Reading List**


**Resources** Study guide, worksheets, lecture rooms with data projector, computer laboratory access, tutorial rooms with OHP.

**Module Evaluation** Course questionnaires, course review.