University of Leicester
Department of Computer Science

Module Forms

27th January 2016
Contents

Our Modules ii

CO1001 — Logic and Problem Solving 1
CO1003 — Program Design 3
CO1005 — Data Structures and Development Environments 5
CO1007 — Study Skills and Professional Practice 7
CO1012 — Discrete Structures 9
CO1016 — Computer Systems 11
CO1019 — Databases and Web Applications 13
CO1094 — Computers and Society 15
CO1097 — Internet Computing 17
CO1098 — Information Management 19
CO2001 — User Interfaces and HCI 20
CO2002 — Business and Financial Computing 22
CO2006 — Software Engineering and System Development 24
CO2008 — Functional Programming 26
CO2011 — Automata, Languages and Computation 28
CO2012 — Software Project Management and Professionalism 30
CO2015 — Software Engineering Project 32
CO2016 — Multimedia and Computer Graphics 34
CO2017 — Operating Systems, Networks and Distributed Systems 36
CO3002 — Analysis and Design of Algorithms 38
CO3007 — Communication and Concurrency 40
CO3014 — Mathematics and Computer Science Project 42
CO3015 — Computer Science Project 44
CO3016 — Computing Project 46
CO3090 — Distributed Systems and Applications 48
CO3094 — System Modelling 50
Our Modules

This booklet provides the following for each module:

**Summary Information** Details of module lecturer, prerequisites, number of teaching sessions (such as lectures etc), and form of assessment.

**Subject Knowledge**

- **Aims** A short description of the module subject knowledge.
- **Learning Outcomes** A description of the knowledge that students should have on completion of the module, and what they should be able to do with this knowledge. This should cross-match with the appropriate Programme Specification. (It must be possible for the module convenor to objectively measure, by some method of assessment, if the learning outcomes have been met.)
- **Methods** A list of the methods which support teaching and learning, and thus ensure that the subject knowledge outcomes are achieved.
- **Assessment** A list of the methods which are used to measure if the subject knowledge outcomes have been met by the students.

**Skills**

- **Aims** A short description of the skills that students will acquire.
- **Learning Outcomes** A description of the skills that students should have on completion of the module. This should cross-match with the appropriate Programme Specification and Skills Matrix. (It must be possible for the module convenor to objectively measure, by some method of assessment, if the learning outcomes have been met.)
- **Methods** A list of the methods which support teaching and learning, and which ensure that the skills outcomes are achieved.
- **Assessment** A list of the methods which are used to measure if the skills outcomes have been met by the students.

**Explanation of Prerequisites** A short description of the preliminary knowledge which students need to successfully complete the module.

**Course Description** An elementary, descriptive account of the topics covered by the module, which should be understandable to any student who has the required prerequisites.

**Detailed Syllabus** A full list of topics which will be taught. This should be reasonably detailed.

**Reading List** Books, articles and other information sources to support learning:

- Reading list [A] books are essential.
- Reading list [B] books are recommended.
- Reading list [C] books are background.

**Resources** List all resources which are required for teaching the module.

**Module Evaluation** List the methods through which the module will be evaluated.

*Please note that some versions of the booklet do not provide all of the above.*
Subject Knowledge

Aims This module teaches the very basic principles of logic in computer science and gives the student an understanding of the principles of logic programming and how these are applied to standard problems in Artificial Intelligence (AI).

Learning Outcomes Students should be able to demonstrate understanding of

- propositional and predicate logics, and how they can help to solve problems and to write correct programs;
- the principles and techniques of logic programming, and how these can be applied in practice, for example in AI;
- how Prolog relates to the predicate calculus;
- the execution mechanism of Prolog in terms of unification, resolution and SLD-trees.

Methods Lectures, surgeries, problem classes, laboratories, class tests, web-support.

Assessment Marked problem-based worksheets, class tests, traditional written problem-based examination.

Skills

Aims To teach students the use of logic for computing in general and that of Prolog in particular.

Learning Outcomes Students will be able

- to formalise problems in a formal logical language;
- to apply the tableau-method to check satisfiability and validity of propositional formulae;
- to use predicate logic to query a database;
- to use the concepts of invariant and pre/post-condition;
- to solve standard problems in AI;
- to write programs in Prolog using a mixture of recursion, arithmetic, lists/trees.

Methods Class sessions together with worksheets.

Assessment Marked problem-based worksheets, class tests, traditional written examination.
Course Description  Logic provides foundations, methods, and skills to Computer Science. To give only a few examples:

- foundations: the very hardware of computers is based on Propositional Logic;
- methods: logic helps in understanding (and specifying and verifying) programs;
- skills: understanding crucial programming concepts such as the scope of an identifier or analysing a given problem in logical terms.

The first part of the course introduces the two most important logics, Propositional Logic and Predicate Logic. Various examples of their importance to computing will be discussed. The second part of the course will show that a fragment of Predicate Logic can actually be used as a programming language (PROLOG). It will be used to exemplify the use of logic to solve problems.

Detailed Syllabus  Propositional Logic, Predicate Logic, Logic and Programming (Invariants, Pre/Post Conditions), Logic Programming (predicate logic as a programming language, programming with lists/trees and recursion, execution mechanism (unification, SLD-trees).

Reading List


Resources  Course notes, web page, study guide, worksheets, hand-outs, lecture rooms.

Module Evaluation  Course questionnaires, course review.
CO1003 Program Design

Credits: 20 Convenor: Dr. M. Hoffmann Semester: 1st

Prerequisites: none
Assessment: Coursework: 40% Three hour exam in January: 60%
Lectures: 29 hours
Surgeries: 10 hours
Laboratories: 20 hours
Class Tests: 1 hours
Private Study: 90 hours

Subject Knowledge

Aims This module teaches the basic principles of object-oriented programming and design.

Learning Outcomes Students should be able to demonstrate understanding of: the basic components of an object-oriented program including methods and attributes, the distinction between classes and instances, the structures required to write basic algorithms, and the components of simple text and graphics based interfaces. They should be able to undertake design using basic object-oriented design notation and demonstrate the applicability and effectiveness of various basic software testing techniques.

Methods Class sessions, recommended textbook, worksheets, automated feedback and extensive web support.

Assessment Marked coursework, written examination.

Skills

Aims Produce written work in a number of different formats; analyse problems, formulate strategies to solve them, design a plan, carry out the required research, implement and evaluate the solution; recognise the need for information, and then locate and access that information.

Learning Outcomes Students will be able to develop object-oriented programs to satisfy simple problems. This will involve analysis of the problem and the development and implementation of suitable solution strategies. Students will also be able to produce simple design diagrams for the code they produce.

Methods Class sessions, worksheets with automated feedback system.

Assessment Marked coursework (with automated feedback system), written examination.

Explanation of Prerequisites Strongly motivated students will have no problems even if they have no previous experience of programming. The module assumes knowledge of mathematics up to GCSE level.

Course Description Programming and programming skills build part of the foundation of every computer science degree. Many of the ideas and concepts are shared between different programming languages. In this module we focus on Java 1.6. The programming language Java is an Object-Oriented language. The Object-Oriented concepts build the backbone of this module. Starting with the fundamental ideas of classes, attributes and methods we develop many practical examples. Many ideas introduced in this module will be extended on in the 2nd semester module CO1005. Throughout the module we not only focus on the implementation but also on the design and design techniques of larger software products. Step by step we use the UML notations and diagrams to represent and reason about different design options.

Detailed Syllabus

The Department of Computer Science
1. Basic Java concepts: Java virtual machine, byte-code; applications and applets; source, editors, compilers, development environments.

2. Fundamentals of Java programming: types; classes; objects; packages; assignment.

3. Structured programming: methods and parameters; for-loops, while-loops, do-loops.

4. Interactive input.

5. Selection with if-else; the switch statement.

6. Introduction to exception handling.


8. Strings and string handling, formatting.

9. Overview of design and development concepts; requirements analysis; basic notions of specification.


12. Testing.

Reading List


Resources Departmental web page, text book web site, automated feedback and assessment tool (coursemaster), study guide, worksheets, handouts, lecture rooms computer projection facilities and OHPs, past examination papers.

Module Evaluation Course questionnaires, course review.
CO1005 Data Structures and Development Environments

Credits: 20  Convenor: tbc  Semester: 2nd

Prerequisites: Essential: CO1003

Assessment: Coursework: 40%  Three Hour Exam in May/June: 60%
Lectures: 30 hours  Class Tests: 2 hours
Surgeries: 10 hours  Private Study: 90 hours
Laboratories: 20 hours

Subject Knowledge

Aims This module teaches advanced features of the Java language which require sophisticated design techniques and algorithms. Suitable program development environments are also taught.

Learning Outcomes Students should be able to demonstrate an understanding of the fundamental types of structured and dynamic data structure, their specification as abstract data types, and their implementation in Java. Students should be able to demonstrate an understanding of some of the main algorithms for processing dynamic datatypes, and to be able to write Java programs using these algorithms. Students should be able to analyse the behaviour of Java programs with the help of exceptions and structural testing. Students should be able to demonstrate an understanding of the programming and runtime environment of Java.

Methods Class sessions, recommended textbook, worksheets, feedback from markers and extensive web support.

Assessment Marked coursework, written examination, class tests, automated feedback.

Skills

Aims Students should be able to produce written work in a number of different formats; analyse problems, formulate strategies to solve them, design a plan, carry out the required research, implement and evaluate the solution; recognise the need for information, and then locate and access that information.

Learning Outcomes On successful completion of the module students should be able to:

- work with integrated development environments (IDEs).
- understand the design and implementation of object oriented data structures in Java.
- understand the development and implementation of suitable solution strategies for Java based applications.
- design and implement Java programs to satisfy problems of moderate complexity.

Methods Class sessions, worksheets with feedback from markers, Linux, Java 1.6, JRE plugin for web browser, Xemacs, JDE, CourseMarker, JUnit, Eclipse.

Assessment Marked coursework, class tests, written examination.

Explanation of Prerequisites Since its purpose is to lead the student on to more advanced programming concepts, the module assumes that CO1003, Program Design, has already been taken.

Course Description The purpose of the module CO1005 is to take the student beyond the elementary parts of the Java language as covered in CO1003, introducing advanced features of the language which require sophisticated design and development tools, techniques and algorithms. In particular,
students would learn powerful features of Object oriented programming, complex data structures, exception handling techniques, methodologies and algorithms for sorting and searching over data structures. Additionally basic techniques for debugging and packaging java applications would be introduced.

**Detailed Syllabus**  
Inheritance, abstract classes and interfaces, stacks, linked lists, queues, trees. Abstract data types and their implementation in Java. Algorithms to handle structured data objects: arrays; sorting and searching, recursion. Basic exception handling. Testing, JUnit, structural testing; Java environments, command line compilation and linking. Debugger, tool support. Integrated Development Environments.

**Reading List**


Also available online at:

http://java.sun.com/docs/books/tutorial/index.html


**Resources**  
Departmental web page, text book web site, study guide, worksheets, handouts, lecture rooms computer projection facilities and OHPs, past examination papers.

**Module Evaluation**  
Course questionnaires, course review.
CO1007 Study Skills and Professional Practice

Credits: 10  Convenor: Dr. R. L. Crole & Prof. R. Heckel  Semester: 1st

Prerequisites: none  Exam: 0%
Assessment: Coursework: 100%  Class Tests: 2 hours
Workshops: 18 hours  Other: 20 hours
Private Study: 35 hours

Subject Knowledge

Aims  To introduce students to professional bodies (especially the BCS, The Chartered Institute for IT, and aspects of computing law and ethics.

Learning Outcomes  Describe the main features of professional bodies, law, social issues and ethics in Computing.

Methods  Lectures, workshops and surgeries. Support with web based materials and learning packs from the Student Development Unit.

Assessment  Essays, group work, and class test.

Skills

Aims  To equip students with a range of study skills and organisation techniques, so as to enable independent and life-long learning.

Learning Outcomes

Students should be able to

- A.Identify ways of understanding and using taught classes, assessment and feedback to support and develop their learning.
- B.Describe and assess the main features of professional bodies, law, social issues and ethics in Computing.
- C.Write clearly in ways appropriate to the study of Computing.
- D.Understand and apply principles of effective time management to their studies.
- E.Develop their reflective practice to aid effectiveness of personal development planning.
- F.Understand and apply principles of team working and so contribute effectively to group work.
- G.Develop good scholarship in the searching, evaluating and using of information sources.
- H.Explain key techniques in developing their revision and exam skills.

Methods  Lectures, workshops and surgeries. Support with web based materials and learning packs from the Student Support Unit.

Assessment  Essays, group work, and class test.

Explanation of Prerequisites  Some previous experience of study and time organization.
Course Description  Study and learning at University requires a large volume of private study and independent learning. The teaching methods are very different from those typically encountered at schools. This module teaches the required skills, and does so from the viewpoint of a computing professional. As such, you will learn about the BCS, computing law and ethics, and the role of computers in today’s society.

Detailed Syllabus

- Lectures/workshop on professional societies, professional practise, law and ethics.
- Guest external lecture on professionalism.
- SLC workshop on making use of classes.
- SLC workshops on writing skills, information sources and plagiarism.
- SLC workshop on Personal Development (PDP).
- SLC workshop on time management and work organization.
- SLC workshop on group work, and assessment and feedback.
- Workshop on practical group work.
- Tutorials on examples of good writing, note taking, plagiarism, feedback and assessment.

Reading List

[B]  Zobel, Justin, Writing for computer science, 2005.

Resources  Workshop notes, web page, study guide, worksheets, lecture rooms.

Module Evaluation  Questionnaires and module review.
CO1012 Discrete Structures

Credits: 10  Convenor: tbc  Semester: 1st

Prerequisites: Essential: GCSE Mathematics (or equivalent)

Assessment: Coursework: 30%  Three Hour Examination in January: 70%

Lectures: 14 hours  Problem Classes: 6 hours
Surgeries: 16 hours  Class Tests: 4 hours
Private Study: 35 hours

Subject Knowledge

Aims This module introduces some basic concepts from discrete mathematics that are essential in the study of Computing or Computer Science.

Learning Outcomes Students should be able to:

- translate basic logical propositions to and from English;
- understand basic set notation and solve simple problems concerning sets;
- define relations, specify the matrix representation of a graph or a relation, and perform basic operations on matrices;
- solve simple problems on functions, including problems concerning partiality and composition;
- solve simple problems involving exponentials and logarithms, factorials, combinatorics and order notation.

Methods Class sessions together with course notes, surgeries, worksheets, problem classes.

Assessment One marked coursework, four class tests, traditional written examination.

Skills

Aims To teach students scientific writing and problem solving skills.

Learning Outcomes Students should be able to:

- understand statements expressed in formal notation;
- solve abstract and concrete problems (both routine seen and simple unseen);
- write neat presentations of mathematical problems and their solutions;
- apply problem solving skills.

Methods Class sessions together with course notes, surgeries, worksheets, problem classes.

Assessment One marked coursework, four class tests, traditional written examination.

Explanation of Prerequisites There is no prerequisite knowledge required for this module apart from some topics from GCSE Mathematics.
Course Description

The main purpose of this course is to teach the basic concepts from discrete mathematics that are needed in the study of Computer Science. While the main purpose is to learn the necessary mathematics, the course is taught from a Computer Science viewpoint throughout. We do not assume any prior knowledge of mathematics other than some basic concepts from GCSE Mathematics (or equivalent).

There will be problem classes (for going through the assessed work) and a surgery session each week (to enable to students to attempt questions and overcome any difficulties they are having with the material).

Detailed Syllabus


7. Elementary probability. Big O notation: concept and basic properties.

Reading List


Resources  Textbook, web page, study guide, surgery questions, class tests, past examination papers, assignment; lecture rooms with whiteboards and data projector, surgery room with assistants.

Module Evaluation  Course questionnaires, course review.
Subject Knowledge

**Aims**  This module teaches the basic principles and technical details of the structure and operation of a modern computer.

**Learning Outcomes**  Students should be able to: describe the memory-I/O model and top-level hardware; solve problems in computer arithmetic; give an account of, and solve problems, in propositional logic and circuit theory and practice; give a summary of, and solve simple problems in, the MIPS assembly and machine language, including addressing methods; summarise the technical details of an elementary processor.

**Methods**  Class sessions together with course notes, recommended textbook, worksheets, printed solutions, and some additional hand-outs and web support.

**Assessment**  Marked problem-based worksheets, class tests, traditional written problem-based examination.

Skills

**Aims**  To teach students scientific writing and problem solving skills.

**Learning Outcomes**  Students will be able to: write short, clear, note based, summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen), including numerical data.

**Methods**  Class sessions together with worksheets.

**Assessment**  Marked problem-based worksheets, class tests, traditional written examination.

Explanation of Prerequisites  No specific knowledge is required, but a very rudimentary understanding of logic and discrete mathematics will be helpful.

Course Description  While modern (personal) computers are complex devices, there are a small number of key components from which the majority of computers are composed. This module will provide a broad picture of a modern computer, covering key hardware and software components: Very roughly speaking, hardware refers to physical artefacts such as a keyboard or memory chip, and software to programs which are stored using magnetic or electrical systems (although we shall see that we need to be a little more precise in the module). In particular, there is an emphasis on hardware, and we only look at very “simple/low-level” software. The module will teach details of computer arithmetic (arithmetic calculations take place when almost any program runs), processors (the central circuits which organize how a program runs, and do the required arithmetic) and memory (circuits for storing data). The design of fundamental circuits is explained, together with the details of many of the basic hardware circuits which are built from the fundamental circuits. The course also explains in detail how computers perform simple arithmetic, covering the theory and also the actual circuits. The circuit details of a very small processor are explained, based upon all of the previous material, and the program instructions that the processor executes are explained in some detail. This complete description of a processor is
the culmination of the module, and students will then be equipped to read about full-scale modern processors.

**Detailed Syllabus**  
*Examples throughout the course will be based on the MIPS Instruction Set Architecture.*

The top level view of a modern computer: memory, processors, I/O, the fetch, decode, execute cycle. Memory layouts and the Endian systems. The memory hierarchy and simple details of cache memory.

The binary number system, elementary logic, and truth tables. Binary arithmetic: basic definitions, algorithms for computing arithmetic operations. 2s-complement integers. Overflow and correctness conditions.


The MIPS instruction set and simple MIPS programs. A subset of the MIPS language treated in detail at the assembly and machine levels. Semantics, machine fields, branch calculations, and assembly/machine translations.

Construction of a simple datapath via composition of atomic ALUs. Description of MIPS control program. The interaction of the datapath and control to make a processor. Computing performance.

**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

**Module Evaluation**  
Course questionnaires, course review.
CO1019 Databases and Web Applications

Credits: 20  Convenor: Dr. Stuart Kerrigan  Semester: 2nd

Prerequisites: Essential: CO1012  Desirable: CO1003
Assessment: Coursework: 40%  Three hour exam in January: 60%
Lectures: 30 hours
Surgeries: 10 hours
Laboratories: 20 hours
Private Study: 90 hours

Subject Knowledge

Aims  This module teaches how to design, implement, maintain and query relational databases. They should also be able to create simple web pages with database connectivity.

Learning Outcomes  Students should be able to demonstrate an understanding of the basic techniques involved in data organisation, storage and retrieval based on the relational database model. They should be able to implement, maintain, and query simple databases using database management system software MySQL. The students should be able to create static web pages using XHTML and dynamic web pages with content obtained from a database. They will learn the need for sessions for interactive web applications and how to program sessions with PHP.

Methods  Lectures, surgeries, laboratory practical sessions together with course notes (available on the Web and in the printed form), recommended textbooks and software manuals, class and laboratory worksheets, printed solutions, and Web support.

Assessment  Marked coursework, laboratory assessments, traditional written problem-based examination.

Skills

Aims  To teach students scientific writing, problem solving and information handling skills.

Learning Outcomes  Students will be able to: write short summaries of technical material as well as short reports describing database and web page design process; solve abstract and concrete problems (both routine seen, and simple unseen); and locate, access, organise and evaluate, and build upon existing information regarding database solutions.

Methods  Class and laboratory sessions, course notes, software manuals, class and laboratory worksheets, printed solutions, and web support.

Assessment  Marked coursework, laboratory assessments, traditional written problem-based examination.

Explanation of Prerequisites  No specific knowledge is required.

Course Description

Detailed Syllabus  HTML (static webpages): Internet Technologies; The Internet Protocol Stack; HTML, XHTML; HTML Forms.
MySQL: revision of sets, relations and classical logic, querying a database, views; database implementation; database normalization (1NF, 2NF and 3NF); Data Definition Language (CREATE, ALTER, DROP), Data Manipulation Language (SELECT, UPDATE, DELETE), Data Control Language (GRANT, REVOKE).
PHP (dynamic pages): php scripting, obtaining content from databases, sessions and cookies.
Reading List


Resources  Course notes, text books in library, study guide, worksheets, handouts, past examination papers, module web pages, lecture rooms with fixed computer, data projector and OHPs, laboratories with PCs and demonstrators, Microsoft Access and MySQL software tools, electronic coursework submission facility, surgeries with assistants, Internet.

Module Evaluation  Course questionnaires, course review.
CO1094 Computers and Society

Credits: 20  Convenor: Dr. E. Law  Semester: 2nd

Prerequisites: none

Assessment: Coursework: 40%  Three Hour Examination in May/June: 60%

Lectures: 40 hours  Problem Classes: 10 hours
Private Study: 100 hours

Subject Knowledge

Aims  The aim of this module is to explain the notion of an Information Society and the impact and effects on society as a whole.

Learning Outcomes  Students should be able to describe the Information Society and Information Revolution; explain the effect that computers and IT have had employment in general and both individuals’ jobs and corporate organisations; examine the impact of the computer revolution on the conditions of work and life in contemporary society such as the usage of social networking sites; explain the issues of access to computers such as privacy and security, the inequality that can arise, and the impacts on society; describe the issues surrounding information access rights; examine human-computer interaction issues and their impacts on different IT-enhanced sectors such as healthcare, education, electronic commerce and environment; be able to outline a brief history of digital computing.

Methods  Class sessions together with course notes, recommended textbooks, worksheets, additional hand-outs and web support.

Assessment  Marked coursework, class test, traditional written examination.

Skills

Aims  To teach students scientific writing and critical thinking.

Learning Outcomes  Students will be able to: write short, clear, note based summaries of general knowledge and present ideas orally.

Methods  Class sessions together with discussions.

Assessment  Marked essays on specific topics, in-class presentation, a class test, traditional written examination.

Explanation of Prerequisites  While there are no formal essential prerequisites, students should already have a basic knowledge of computing, the use of computers in society, and ideally some appreciation of coding.

Course Description  The module examines the historical development of computing technologies, the impact of the computer revolution on the conditions on work and life in contemporary society. It discusses ethical, legal, and professional issues in computer uses. It also analyses relevant theoretical frameworks to understand the dynamic and dialectical interactions between humans and technologies. Critical thinking is essential to reflect deeply on the ever-evolving human-technology relationships.

Detailed Syllabus

Topic 1: History of Digital Computing
Topic 2: Social Change and Technology
Topic 3: Open Source Movement
Topic 4: Computer-mediated Social Networks
Topic 5: Digital Identity
Topic 6: Human-computer Interaction: Usability and User Experience
Topic 7: Computers and Education
Topic 8: Privacy and Security
Topic 9: Digital Dilemma

Resources   Web page, study guide, worksheets, handouts, lecture rooms.

Module Evaluation   Course questionnaires, course review.
CO1097 Internet Computing

Credits: 10  Convenor: Dr. Fer-Jan de Vries  Semester: 1st

Prerequisites: none
Assessment: Coursework: 40%  Two hour exam in January: 60%

Lectures: 15 hours  Private Study: 45 hours
Laboratories: 15 hours

Subject Knowledge

Aims
The module will give the student a basic grounding for accessing and disseminating data across the Internet with special focus on the World Wide Web.

Learning Outcomes
Students will be able to give a coherent account of the basic technology, organisation and architecture of the Internet and the World Wide Web. They will be able to discuss the issues of Internet security and relate these to a given scenario. Students will be able to use Internet search tools to find information. They will be able to create and write static web pages using appropriate layout and graphics etc. Students will be able to organise and maintain a web site of moderate size.

Methods
Lectures, laboratory classes, recommended reading, worksheets, additional hand-outs and web support.

Assessment
Marked coursework, traditional written examination.

Skills

Aims
To teach students how to methodically solve problems given the techniques available to them.

Learning Outcomes
Students will be able to identify information needs; retrieve information relevant to those needs; organize and present information for dissemination.

Methods
Lectures, laboratory classes, and web support.

Assessment
Marked coursework, traditional written examination.

Explanation of Prerequisites
This is a basic introductory course and hence no prerequisites are required.

Course Description
The rapid growth of the Internet has affected all areas of life including how students of all disciplines obtain and present data. One of the easiest ways of doing this is via the World Wide Web, eg an Arts student may want to produce an on-line bibliography or web site of literary resources, while a Science student may want to make various data sets available on the Web.
This course will teach students how to collect data by searching the Web and how to create and maintain a web-site for disseminating such information. As such the course will cover the structure of the Internet and Web, the construction and maintenance of a web-site and issues pertaining to the security of websites.

**Detailed Syllabus**


**Reading List**


**Resources**

Lecture slides, web page, study guide, worksheets, handouts, past examination papers, lecture rooms with data-projector and OHP, laboratory access.

**Module Evaluation**

Course questionnaires, course review.
CO1098 Information Management

Credits: 10  Convenor: Dr. Stuart Kerrigan  Semester: 1st

Prerequisites: none
Assessment: Coursework: 100%  none: %
Lectures: 3 hours
Laboratories: 20 hours  Private Study: 52 hours

Subject Knowledge

Aims This module aims to teach the use of the computer as a tool.

Learning Outcomes Basic computer literacy. Students will learn to understand and gain experience with a variety of components of the Windows XP operating system.

Methods Class sessions plus extensive laboratory classes and coursework.

Assessment Marked coursework.

Skills

Aims To develop some basic IT skills, in particular, students will learn skills with the Windows XP operating system.

Learning Outcomes The student will develop skills in handling Word and Excel. The full scope of these packages is explored and practical skills are developed through a series of laboratory exercises. Students will also gain skills in using communication packages, in both sending and receiving e-mail, and in accessing the Internet. The course emphasises a ‘hands-on’ approach. Experience with computers is not a prerequisite for this module.

Methods Coursework with a variety of laboratory exercises

Assessment Marked coursework.

Course Description The module comprises a series of ten laboratory classes with on-line instruction, supplemented by three lecture demonstrations. Assessment is solely on the laboratory course work. The course teaches hands on experience with a number of much used software packages.

Detailed Syllabus Operating systems and Windows. Document preparation and word processing in Microsoft Word: formatting, layout, styles, sectioning, tables, etc. Storage, analysis and presentation of data using Microsoft Excel: formulae and calculation; charts; databases; applications. Electronic mail, Internet Explorer & Netscape.

Reading List

There is no recommended textbook for this module. All of the documentation required to complete the laboratory exercises is provided on the Web pages associated with module.

In case you really want to buy a book: any introductory book on Word or Excel will do. They all tend to be rather similar so just choose one that you like.

Resources Study guide, Web page, study guide, coursework, electronic coursework submission facility, demonstrator and postgraduate support, computer labs, lecture rooms with two OHPs.

Module Evaluation Course questionnaires, course review.

The Department of Computer Science
CO2001 User Interfaces and HCI

Credits: 10  Convenor: Dr. M. Hoffmann  Semester: 1st

Prerequisites: Essential: CO1003, CO1005, CO1012
Assessment: Coursework: 100%

Lectures: 15 hours  Class Tests: 2 hours
Surgeries: 8 hours  Private Study: 32 hours
Laboratories: 18 hours

Subject Knowledge

Aims
To introduce students to the subject of Human Computer Interaction through the medium of object-oriented programming using relevant programming concepts.

Learning Outcomes
Students should be able to demonstrate an understanding of advanced object-oriented techniques such as Graphical User Interface (GUI) concepts; and the event-driven model of programming, and threading. They should be able to construct GUI based applications and applets in Java. Students should be able to demonstrate a knowledge of and be able to apply basic HCI concepts.

Methods
Class sessions together with course notes, recommended textbook, worksheets, and some additional hand-outs and web support.

Assessment
Four pieces of marked coursework consisting of 1 assessed worksheet, two class tests and an individual mini-project.

Skills

Aims
Students should be able to produce written work in a number of different formats; analyse problems, formulate strategies to solve them, design a plan, carry out the required research, implement and evaluate the solution; recognise the need for information, and then locate and access that information.

Learning Outcomes
On successful completion of the module students should be able to:

- analyse user interface requirements, understand the development and implementation of suitable solution strategies for user friendly graphical interfaces.
- develop graphical user interfaces in Java to satisfy problems of moderate complexity using threads, applets, applications and Files.

Methods
Class sessions, worksheets with feedback from markers, Linux, Java 1.6, Java swing, JRE plugin for web browser, JDE.

Assessment
Four pieces of marked coursework consisting of 1 assessed worksheet, two class tests and an individual mini-project.

Explanation of Prerequisites
The purpose of the module is to enable students to design and implement interactive graphical user interfaces using advanced object oriented techniques and data structures in Java. The module therefore assumes that CO1003 - Program design, CO1005 - Data structures and Development environment and CO1012 - Discrete Structures have been taken.

Course Description
Graphical user interfaces are a vast class of software systems that are designed for interacting with the users. Programs with GUIs are event driven, i.e., the program reacts to actions of the users which are called events. GUI based applications are also often “multithreaded”. Multithreaded execution is an essential feature of the Java platform and enables concurrency.
The objectives of the module are to lead the students on to advanced event driven programming techniques for building multithreaded graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications.

**Detailed Syllabus**  

**Reading List**


Also available online at:  
http://java.sun.com/docs/books/tutorial/uiswing/index.html


**Resources**  
Online resources, course notes, departmental web page, study guide, worksheets, handouts, lecture rooms with projection facilities and OHPs.

**Module Evaluation**  
Course questionnaires, course review.
CO2002 Business and Financial Computing

Credits: 10  Convenor: Dr. G. Koutsoukos  Semester: 2nd

Prerequisites: Essential: CO1003, CO1005, Desirable: CO1012

Assessment: Coursework: 40%  Two hour exam in May/June: 60%

Lectures: 15 hours
Surgeries: 10 hours
Laboratories: 15 hours

Class Tests: 1 hours
Private Study: 39 hours

Subject Knowledge

Aims The aim of this module is to introduce students to the fundamentals of financial and business computing, giving them a clear idea of the financial principles, the business organization and the software applications to support these domains. Moreover, the module will explore some basic information systems concepts and the role of IT departments in such large organizations.

Learning Outcomes At the end of the course the student should be able to: be aware of some of the fundamental concepts, terminology and processes of the financial domain; understand the categories and functions of business and financial systems and applications; be aware of the different roles and functions of IT professionals within such organizations.

Methods Lectures, tutorials and practical sessions together with course notes, recommended reading, worksheets and some additional handouts.

Assessment Marked problem-based worksheets, class tests, traditional written problem-based examination.

Skills

Aims To help students improve their analytical and problem solving skills.

Learning Outcomes Students will be able to apply logical thinking in order to solve abstract and concrete problems and make decisions based on available information.

Methods Class sessions together with worksheets

Explanation of Prerequisites You will need to have a good understanding of object oriented programming and some knowledge of basic mathematics. It will be helpful to have an acquaintance with databases and web-based front ends, but these tools do not play a primary role in this module.

Course Description Business organizations in general and financial services ones in particular are experiencing a number of events that have a significant impact on the way they used to think on and operate their IT systems. These events can be categorized into two interrelated dimensions: business and technology. On the business side, customers more and more require differentiated products and services according to their needs and lifestyle, are better informed and with different life patterns, regulation further intensifies after the 2008 economic crisis, mergers and acquisitions continue, the cost factor becomes even more critical and economy and business are more global and more dynamic. On the technology side, the advent of even more advanced communications technologies with new exciting capabilities, the increasing use of mobility solutions, the popularity of social networks, the Web 2.0, the Open-Source solutions and the Green movement among others. All the above are raising new challenges for IT professionals in such organizations. To be able to effectively compete in such an environment, sole technical knowledge and skills no longer suffice. A sound understanding of the fundamental principles according to which business and financial organizations operate, a good grasp of key business processes and of the relationship between business and IT has become a critical success factor. Taking
the above into account, this module is designed to achieve the following: (i) introduce students to the fundamentals of financial and business computing, giving them a clear idea of the financial principles, the business organization and the software applications to support these domains (ii) explore some basic information systems concepts and (iii) provide a good overview of the role of IT departments in such large organizations.

**Detailed Syllabus** Topics to be covered include: The financial services domain (basic definitions and terminology), financial markets and organizations with emphasis in banks, key processes (e.g. loan origination, payments), types of information systems in such organizations and their functional and architectural perspectives, business intelligence systems, IT roles and functions, current industry trends and issues.

**Reading List**


**Resources** Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

**Module Evaluation** Course questionnaires, course review.
# CO2006 Software Engineering and System Development

**Credits:** 20  
**Convenor:** tbc  
**Semester:** 1st

<table>
<thead>
<tr>
<th>Prerequisites:</th>
<th>Essential: CO1003, CO1005, CO1019</th>
<th>Desirable: CO1001, CO1012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessment:</td>
<td>Coursework: 40%</td>
<td>Three hour exam in January: 60%</td>
</tr>
<tr>
<td>Lectures:</td>
<td>30 hours</td>
<td>Problem Classes: 0 hours</td>
</tr>
<tr>
<td>Surgeries:</td>
<td>10 hours</td>
<td>Private Study: 90 hours</td>
</tr>
<tr>
<td>Laboratories:</td>
<td>20 hours</td>
<td></td>
</tr>
</tbody>
</table>

## Subject Knowledge

### Aims
According to a report of British Computer Society, only about 16% of IT projects can be considered truly successful and over 60% 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 and implementation of software systems.

### Learning Outcomes
At the end of this course, students should be able to:

- Specify customer requirements in a structured requirements document
- Analyze customer requirements and produce an object-oriented system design
- Use UML for consistent specification of software systems and business processes
- Implement and test the system
- Use appropriate tools for software development and testing
- Understand the software development processes

### Methods
Lecture notes, class sessions, recommended textbooks, work-sheets, supervised laboratories, feedback from markers and extensive web support.

### Assessment
Programming tests, small implementation projects, problem-based worksheets, lab tests, examination.

## Skills

### Aims
The student should be able to analyse problems, formulate solution strategies and solve the problems. They will be also able to effectively work in a group and to communicate ideas to other group members.

### Learning Outcomes
Students will learn how to develop object-oriented software systems, how to plan the development and how to work in groups. This will involve requirements specification and analysis as well as system design, implementation and testing.

### Methods
Class sessions, individual and group work-sheets, feedback from module convenors.

### Assessment
Marked coursework, lab test.

## Explanation of Prerequisites
A sound knowledge of basic algorithms, data structures and programming is required. An understanding of logic and discrete structures is important for rigorous specification of software systems. Some knowledge of database systems is desirable.
Course Description  This module teaches engineering principles and methods needed to specify, design and implement a large system using object oriented techniques.

Detailed Syllabus

Introduction: Software crisis and historical background of Software Engineering; software products and their characteristics: maintainability, dependability, efficiency and usability.

Introduction to OO Development: The inherent complexity of software; mastering complex systems; examples of complex systems; function oriented vs object-oriented methods.

Requirements specification: requirements documents; use cases and scenarios; model of problem domain; functional and non-functional requirements.

Object-oriented analysis: analysis class and sequence diagrams.

Object-oriented design: design class and sequence diagrams; object diagrams; state machines; design patterns; consistency of multi-view models; software architectures.

Implementation: mapping object-oriented designs to code; container/collection classes in code; testing and JUnit.

Reading List


Resources  Course notes, module web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

Module Evaluation  Course questionnaires, course review.
CO2008 Functional Programming

Credits: 10  Convenor: Dr. F. J. de Vries  Semester: 2nd

Prerequisites: Essential: CO1001, CO1003, Desirable: CO1005, CO1012

Assessment: Coursework: 40%  Two hour exam in June: 60%

Lectures: 15 hours
Surgeries: 5 hours  Private Study: 45 hours
Laboratories: 10 hours

Subject Knowledge

Aims  The module will give the student an introduction to programming in the functional style using the language Haskell.

Learning Outcomes  Students will be able to demonstrate: skilled use of basic functions and techniques to solve simple problems, with some practical applications; detailed knowledge of numbers, lists, recursion, and patterns; some understanding of higher order functions; and ability to apply Haskell’s mechanism for defining new datatypes.

Methods  Class sessions together with lecture slides, recommended textbook, worksheets, printed solutions, and some additional hand-outs and web support.

Assessment  Marked coursework, traditional written examination.

Skills

Aims  To teach students how to methodically solve problems given the techniques available to them.

Learning Outcomes  Students will be able to: breakdown simple problems to identify essential elements; create a plan to solve a problem; implement a planned solution and evaluate the implementation.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework, traditional written examination.

Explanation of Prerequisites  It is essential that students have taken a first course in basic (imperative) programming, which includes skills involving both coding and design, to a level which includes data structures such as lists and trees.

A grounding in the basic mathematical concepts of sets and functions is useful, but detailed knowledge of other areas of elementary discrete mathematics and logic is not essential.

Course Description  Many of the ideas used in imperative programming arose through necessity in the early days of computing when machines were much slower and had far less memory than they do today. Languages such as C(++) and Pascal carry a substantial legacy from the past. Even Java, despite its OO features, has been devised to look ‘a bit like C’. If one were to start again and design a programming language from scratch what would it look like?

For many applications, the chief concern should be to produce a language which is concise and elegant. It should be expressive enough for a programmer to work productively and efficiently but simple enough to minimize the chance of making serious errors. Rapid development requires the programmer to be able to write algorithms and data structures at a high level without worrying about the details of their machine-level implementation. These are some of the criteria which have led researchers to develop the functional programming language Haskell.

The flavour of programming in Haskell is very different from that in an imperative language. Much of the
irrelevant detail has been swept away. For example, there are at least two different uses for a variable in Java: as a storage location, and as parameter in a method. There is only one use for a variable in Haskell: it stands for a quantity that you don’t yet know, as is standard in mathematical practice. The constructs in Java include expressions, commands, and methods; whereas in Haskell there are only expressions and functions. The meaning of a program in Java or C is understood by the effect it has on the ‘state’ of the machine as it runs. Haskell does away with the idea of ‘state’—the meaning of a program is the values it computes.

On the other hand, Haskell is a very expressive language. The type system allows functions to be written polymorphically so that the same code can be re-used on data of different types, e.g. the same length function works equally well on lists of integers as on lists of reals or lists of strings. Furthermore, it allows one to write functions which take other functions as parameters. These are known as higher order functions and they give a second form of code re-use. There are powerful mechanisms for introducing user-defined datatypes such as trees, sets, graphic objects, etc. Haskell also makes a great deal of use of recursion. The combination of these features makes for very clean, short programs, which, with some experience, are easier to understand than many imperative programs.

This course teaches how to program in Haskell, which exemplifies the functional style.

**Detailed Syllabus**

Basic types, such as Int, Float, String, Bool; examples of expressions of these types; overloading. Functions and declarations, with a high level explanation of a function with general type \( a_1 \rightarrow a_2 \rightarrow a_3 \ldots \rightarrow a_n \). Booleans and guards; correspondence of guards with if-then-else expressions. Pairs and n-tuples; \( \text{fst} \) and \( \text{snd} \) functions for dismantling pairs and tuples. Pattern matching and cases, especially defining functions on lists and tuples. Numeric calculation. Simple recursion, with examples on the natural numbers and lists; list comprehension; list processing examples which use patterns, recursion and comprehensions. Higher-order functions, polymorphism and code re-use; examples such as the reversal of a list. Algebraic and recursively defined datatypes. Examples such as lists and trees.

**Reading List**


**Resources**

Lecture slides, web page, study guide, worksheets, handouts, lecture rooms with OHP, dataprojector and whiteboard; past examination papers.

**Module Evaluation**

Course questionnaires, course review.
Subject Knowledge

Aims The aim of this module is to give an understanding of some of the basic theory of language recognition. The module will also aim to provide a general model of computation and thereby to illustrate the limits of the power of computers, both in terms of the problems for which a solution exists and also the problems for which a feasible solution exists.

Learning Outcomes By the end of the module students should be able to describe some abstract models of the process of computation such as finite automata, pushdown automata and Turing machines. They should be able to construct basic arguments couched in terms of these models.

Methods Class sessions together with course notes, exercises and web support. Recommended textbooks for extra information and supplementary reading.

Assessment Marked class tests, ongoing assessment and written examination.

Skills

Aims To teach students a range of comprehension, writing and problem-solving skills.

Learning Outcomes Students should be able to solve problems and produce reasoned arguments about the power of the computational models studied in the course (using their understanding of these models to solve the problems). They should be capable of writing such arguments clearly and correctly with a proper use of formal notation where appropriate.

Methods Class sessions together with exercises.

Assessment Marked class tests, ongoing assessment and written examination.

Explanation of Prerequisites There is not much in the way of pre-requisite knowledge required for this module. We need the basic concepts of sets, relations and functions as introduced in CO1012. In order to help understand the motivation, it would be helpful to have done some programming before; however, while previous experience of programming is desirable, it is not essential. Some of the methods in this module are expressed in a sort of pseudocode notation, but there is no actual programming content; a student who had not done programming before could still take this module if he/she wanted to. Such students are welcome to discuss their suitability for the course with the module convenor.

Course Description In this course we are primarily concerned with what computers can do. It turns out that there are problems that cannot be solved by computer, or, at least, by machines corresponding to the mathematical models of computers we shall present. It is clearly sensible to investigate which problems cannot be solved; there is no point trying to program a computer to solve a problem that is unsolvable! A problem may be unsolvable in the sense that no computer program exists that will solve it or in the sense that any program that would solve it would take longer than the lifetime of the universe to run. We will give some precise mathematical models of the process of computation; within these models, we will see what sort of tasks can be performed.

At first sight, it may appear that these models are unduly simple and do not really capture all the
subtleties of the process of computation. The advantages of using such models is two-fold. First, they are very simple to reason about, so that we can reach our conclusions much more simply than (for example) considering actual hardware and software components in fine detail. Second, they have proved to be very robust, in that successive generations of computers have all been shown to be no more powerful than the most general model we will present, and so the analysis based on these models has been useful throughout the history of Computer Science, whereas an analysis based on the specifics of various machines and programming languages quickly becomes obsolete.

**Detailed Syllabus**


**Reading List**


**Resources**  Course notes, web page, study guide, exercises, lecture rooms with board space and two OHPs.

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

*The Department of Computer Science*
Subject Knowledge

Aims According to a report of British Computer Society, only about 16% of IT projects can be considered truly successful and over 60% experience severe problems. The purpose of this module is to gain a practical understanding of the software development process, to discuss the management, professional and ethical issues of software development, and get acquainted with industrial best practices in preparation for the software engineering group project.

Learning Outcomes At the end of the module a student should:

• be able to demonstrate a broad understanding of the development processes involved in producing a large software system;
• be able to apply the techniques acquired in the companion module CO2006 in a small team in the course of a web engineering mini project;
• be able to demonstrate the need for quality assurance, project and risk management techniques, and be able to apply suitable strategies in simple cases;
• be able to demonstrate what “professionalism” means in the context of the software industry, and be aware of ethical and legal issues, like the Data Protection Act, likely to affect every professional in the software industry.

Methods Class sessions together with course notes, recommended textbooks, worksheets, mini project.

Assessment Marked coursework, including written essays, mini-projects, lab tests and class tests.

Skills

Aims To develop analytical and problem solving skills, including the ability to make appropriate abstractions. To make reasoned judgements based on quantitative data. To learn skills in research and presenting ideas in a written form.

Learning Outcomes Students will be able to: formulate technical problems and their solution in a methodical way; justify solutions; research an issue and present their findings in writing in a balanced manner.

Methods Class sessions together with worksheets.

Assessment Marked coursework, including a written essay, a mini-project and a class test.

Explanation of Prerequisites A sound knowledge of basic algorithms, data structures and programming is required. An understanding of logic and discrete structures is important for rigorous specification of software systems. Some knowledge of database systems is desirable. The audience for this module will be a subset of that of CO2006, and the syllabi will be coordinated to use some of the conceptual and linguistic background provided by CO2006.
Course Description  This module teaches software engineering management principles and methods needed to specify, design and implement a large system using object oriented techniques, and gives them practice in developing such systems.

Detailed Syllabus
Professional and legal issues: professional liability and its limits; data protection and freedom of information; intellectual property.
Software project management: software process models; risk assessment; project planning and execution.
Languages, methods, and tools for Web Engineering.

Reading List

Resources  Course notes, web pages, study guide, worksheets, handouts, lecture rooms with OHP and data-projector.

Module Evaluation  Course questionnaires, course review.
### CO2015 Software Engineering Project

**Credits:** 20  
**Convenor:** tbc  
**Semester:** 2nd

**Prerequisites:**  
Essential: CO2006, CO2012, CO1003, CO1005, CO1019  
Desirable: CO1012

**Assessment:**  
Individual work: 40%  
Group work: 60%

**Lectures:**  
approx. 5 hours

**Surgeries:**  
approx. 10 hours

**Private Study:** 135.0 hours

---

### Subject Knowledge

**Aims**  
According to a recent report of British Computer Society, only above 16% of IT projects can be considered truly successful and over 60% of projects experience sever problems in. The main reason is that software is still developed in an ad hoc way. The purpose of the module is to teach systematic methods of software and system development. Students will gain skills essential in Software Engineering (SE). They will gain understanding of the difficulties and benefits of working in a group on a large SE project.

**Learning Outcomes**  
Students will learn how interact with a customer, how to analyze customer’s requirements, and how to design and construct a large software system. They will learn how to apply modelling and programming concepts in the SE process. They will learn also how to schedule and manage a SE process life-cycle.

**Methods**  
Class sessions together with course notes, textbooks, group discussions and web support.

**Assessment**  
Assessment of the project deliverables.

### Skills

**Aims**  
Communication, team work, problem solving, group work

**Learning Outcomes**  
Students will be able to: work as part of a team; analyze customer requirements; design, implement and test a software system; produce project reports and system documentation; manage resources. They will be able to document the system and demonstrate it to the customer.

**Methods**  
Lectures, meetings with group supervisors, group discussions, marked group work, reading.

**Assessment**  
Marked course-work, presentation, project reports, and demonstration of the software system.

### Explanation of Prerequisites

In order to implement their system students need to be familiar with the basic techniques of programming as taught in CO1003, CO1005, and web/database development as in CO1019. They will specify and design systems using the software engineering methods taught in CO2006, and also in CO2012.

### Course Description

In this course, students will put into practice methods of Software Engineering that have been studied so far. Students will work in groups of about six. Groups will follow a light weight form of the Unified Process tailored to the needs of this project.

### Detailed Syllabus

The practicing of software engineering methods; software life-cycle management; definition and prioritization of project goals; system specification; system design, implementation and testing; quality assurance; system validation against requirements.

### Reading List

The Department of Computer Science 32


**Resources** Various course notes, web pages, books, study guide, handouts.

**Module Evaluation** Course questionnaires, course review.
CO2016 Multimedia and Computer Graphics

Credits: 10  Convenor: Dr. M. Hoffmann  Semester: 2nd

Prerequisites: Essential: CO1003, CO1005
Assessment: Coursework: 50%  Two hour exam in May/June: 50%
Lectures: 15 hours  Private Study: 45 hours
Laboratories: 15 hours

Subject Knowledge

Aims This module teaches the principles and technical details of multimedia data and 3D-environments.

Learning Outcomes Students should be able to demonstrate understanding of: the basic representation and handling of multimedia data (sound, pictures and animation), the basic components of a 3D-environments.

Methods Class sessions together with course notes, recommended textbook, worksheets, and some additional hand-outs and web support.

Assessment Marked coursework, written examination.

Skills

Aims Produce animation. Create a 3D representation

Learning Outcomes Students will be able to: reason about different multimedia formats, write short animation; write Java 3D components and reason about their behavior; create dynamic 3D environment.

Methods Class sessions together with worksheets.

Assessment Marked coursework, written examination.

Explanation of Prerequisites It is essential that students have a good working knowledge of Java, up to and including the use of abstract classes and exceptions. No specific knowledge about multimedia data is required. It is beneficial if students taking this module have a very rudimentary understanding of a 3 dimensional space.

Course Description The area of multimedia includes a wide variety of data. In this module we will deal with pictures, animation, audio and 3D landscapes. Images are built out of pixels. Each pixel has a certain color or grey tone. Handling Images on the level will allow us to analyse and manipulate images. On the practical side we will program these effects in Java, but also understand what information, if any, is lost by certain effects. Bringing images to life, e.g. placing an animation, we will use the recent established Internet standard SVG. Images in SVG are described using XML documents. This allows scalability and animation. SVG has similar features to FLASH. Completing the introduction to multimedia data we draw our attention to audio data. The understanding of how to digitalize sound and how to deal with sound in the digitalized format (e.g. placing sound effects) and its practical implementation will be the focus here. In the last part of this module we will create virtual landscapes using Java 3D. Apart from the basic concepts these landscapes contain different forms of lighting and lighting effects, moving objects and objects with different behaviors (e.g. collision behaviors) The main computer language for this module is Java including Java3D.

Detailed Syllabus

1. Image analysis
2. Image resizing and dithering
3. Audio data handling
4. Basic SVG concepts
5. Scene graphs in Java3D
6. Textures, lighting in Java3D
7. Rotation and movement of 3D objects
8. Behaviors of 3D objects

Reading List


Resources Course notes, departmental web page, study guide, worksheets, handouts, lecture rooms with projection facilities and OHPs, example examination papers.

Module Evaluation Course questionnaires, course review.
Subject Knowledge

Aims  To understand the role, structure and basic design of computer operating systems; the fundamental theory and practice of networks; and the theory and design of systems distributed through the use of networks.

Learning Outcomes  Students should be able to: describe the fundamentals of current computer operating systems, and communications between computers; to use the Unix operating system; describe key operating system features such as processes, threads, scheduling and synchronization; solve simple problems concerning the benefits and costs of distribution of computer systems; give detailed accounts of the structure and organization of network hardware and software; describe the common physical attributes of networks.

Methods  Class sessions together with recommended textbooks, lab practicals, worksheets, web support.

Assessment  Marked lab practicals, marked coursework, traditional written examination.

Skills

Aims  To teach students scientific writing, problem solving and information handling skills.

Learning Outcomes  Students will be able to: write short, clear summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen), including numerical data; locate and access information.

Methods  Class sessions together with worksheets, lab practicals.

Assessment  Marked lab practicals, marked coursework, traditional written examination.

Explanation of Prerequisites  Some knowledge of Java programming and of hardware is required.

Course Description  An operating system forms the interface between the computer’s hardware and the user; examples include Windows NT, Linux (and other versions of Unix), and MacOS. The operating system has many tasks, such as: managing processes, allocating processor time between different processes; allocating the memory between different processes; organizing input and output; and managing files. The operating system is responsible for protecting the user from other users, and where possible from himself/herself. The Operating Systems part of the module explains how these tasks are carried out in modern computers, and the details of why it is desirable to link together distributed systems to form a single unit.

Linking computers so they may communicate is very much a part of modern life, with the ever-rising popularity of the Internet and the World Wide Web. In the Networks part of the course we will study the science underpinning such communications. Topics of interest will include the underlying physical media, the way data is represented, how errors in transmission can be detected and dealt with, the way information is routed over a large network, and the details of some actual networks which yield distributed computing systems.
Detailed Syllabus

Operating systems/Distributed systems

Introduction Overview; interrupts.

Process management Programs and processes; multitasking; dispatcher; scheduling and scheduling policies; interprocess communication, in particular joint access to shared resources; threads; Java thread programming.

Memory management Memory allocation methods; paging; virtual memory; segmentation; protection and sharing.

File management Concept of file; directory structure; file management techniques; directory implementation.

Networks

Introduction Overview; different sorts of networks; layered protocols.

The Physical Layer A short overview.

The Data Link Layer Error detection and correction; flow control; channel allocation; protocols for local area networks; bridges.

The Network Layer Datagrams and virtual circuits; routing; congestion control; internetworking; the network layer in the Internet.

The Transport Layer Connection management; transport layer in the Internet; congestion control; socket concept; Java socket programming.

The Application Layer Domain name system; E-mail system.

Reading List


Resources Study guide, computer lab, lecture rooms, worksheets, handouts, web page, course notes.

Module Evaluation Course questionnaires, course review.
CO3002 Analysis and Design of Algorithms

Credits: 20  Convenor: Dr. S. Fung  Semester: 2nd

| Prerequisites: | Essential: CO1003, CO1012 | Desirable: CO1001, CO2011 |
| Assessment: | Coursework: 30% | Three hour exam in May/June: 70% |
| Lectures: | 30 hours |
| Surgeries: | 10 hours |
| Problem Classes: | 10 hours |
| Class Tests: | 1 hour |
| Private Study: | 99 hours |

Subject Knowledge

Aims  The module aims to introduce students to the design of algorithms as a means of problem-solving. Students will learn how to analyze the complexity of algorithms. Major algorithm design techniques will be presented and illustrated with fundamental problems in computer science and engineering. Students will also learn the limits of algorithms and how there are still some problems for which it is unknown whether there exist efficient algorithms.

Learning Outcomes  Students should be able to demonstrate how the worst-case time complexity of an algorithm is defined; compare the efficiency of algorithms using asymptotic complexity; design efficient algorithms using standard algorithm design techniques; demonstrate a number of standard algorithms for problems in fundamental areas in computer science and engineering such as sorting, searching, and problems involving graphs.

Methods  Class sessions together with lecture slides, recommended textbook, worksheets, printed solutions, and web support.

Assessment  Marked coursework, class test, traditional written examination.

Skills

Aims  Students will become more experienced in the application of logical and mathematical tools and techniques in computing. They will develop the skills of using standard algorithm design techniques to develop efficient algorithms for new problems. They will develop skills to judge the quality of the algorithms.

Learning Outcomes  Students will be able to solve problems which are algorithm based by using various design techniques. They will be able to apply prior knowledge of standard algorithms to solve new problems, and mathematically evaluate the quality of the solutions. They will be able to produce concise technical writing for describing the solutions and arguing their correctness.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework, class test, traditional written examination.

Explanation of Prerequisites  Typical materials assumed for this module are: the basic notions associated with an imperative programming language such as arrays, while loops, for loops, linked lists, recursion, etc.; and logical and discrete mathematical notions such as induction, asymptotic notation, recurrence relations and their solution, geometric and arithmetic series, etc.

Course Description  This module introduces students to the design and analysis of algorithms. Algorithms are step-by-step procedures, such as those executed by computers, to solve problems. Typical problems include, for example, “what is the shortest path between two locations in a network?”, or “what is the maximum set of activities that can be chosen subject to time constraints?” Just because a problem can be solved, does not mean that there exists a practically time-efficient solution. It is the goal of algorithm designers to develop better and better algorithms for the solution of fundamental or
new problems. The main methods used to design algorithms will be illustrated through examples of fundamental importance in computer science and engineering. These design methods not only apply to the problems illustrated in the module, but also to a much wider range of problems in computer science and engineering. As a result, students can apply the design methods learned to other problems they encounter. Alternatively, it can be the case that no algorithms of a certain quality exist; algorithm designers then need to identify this limitation of algorithms. Techniques for analyzing the efficiency of algorithms and the inherent complexities of problems will be explained.

**Detailed Syllabus**

Asymptotic analysis of algorithms: the notion of asymptotic complexity using big-O notation; solving recurrence relations; master theorem; limitations of algorithms (lower bounds using decision trees).

Algorithm design techniques: divide and conquer; greedy algorithms; dynamic programming.

Algorithms for fundamental problems: sorting (mergesort, Quicksort); searching (binary search); minimum spanning trees (Kruskal’s and Prim’s algorithms); graph traversal; shortest paths (Dijkstra’s algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm); network flow (Ford-Fulkerson algorithm).

**Reading List**


**Resources**

Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

**Module Evaluation**

Course questionnaires, course review.
# CO3007 Communication and Concurrency

**Credits:** 20  
**Convenor:** Dr. I. Ulidowski  
**Semester:** 1st

<table>
<thead>
<tr>
<th>Prerequisites:</th>
<th>Essential: CO1003, CO1001, Desirable: CO2011, CO1005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessment:</td>
<td>Coursework: 40% Three hour exam in January: 60%</td>
</tr>
<tr>
<td>Lectures:</td>
<td>30 hours</td>
</tr>
<tr>
<td>Surgeries:</td>
<td>10 hours</td>
</tr>
<tr>
<td>Class Tests:</td>
<td>1 hours</td>
</tr>
<tr>
<td>Private Study:</td>
<td>109 hours</td>
</tr>
</tbody>
</table>

## Subject Knowledge

### Aims

This module provides students with an introduction to theories and applications of concurrency. In particular, it will familiarise students with the process algebras CCS (Calculus of Communicating Systems) and its operational semantics. The module will teach, via individual and collective work, how to specify, design and implement simple concurrent and distributed systems.

### Learning Outcomes

Students should be able to: demonstrate understanding of the notions of concurrency, communication, and concurrent systems; and CCS and its operational and axiomatic semantics; They should be able to develop informal and formal specifications of simple concurrent systems, and be able to produce systems’ designs from specifications; able to reason about the behaviour of simple concurrent systems using the techniques of equational reasoning and bisimulation.

### Methods

Class and laboratory sessions together with course notes (available on the Web and in the printed form), Bisimulation Games workshop, recommended textbooks, class worksheets, printed solutions, and Web support.

### Assessment

Marked problem-based worksheets, class tests, and traditional problem-based written examination.

## Skills

### Aims

To teach students problem solving and scientific writing skills.

### Learning Outcomes

Students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen); write short summaries of technical material.

### Methods

Class and laboratory sessions, course notes and text books, software manuals, class and laboratory worksheets, printed solutions.

### Assessment

Marked problem-based worksheets, class tests, and traditional problem-based written examination.

## Explanation of Prerequisites

Basic knowledge of discrete mathematics and logic is essential.

## Course Description

A *concurrent system* is a system consisting of several components such that each component acts *concurrently* with, and independently of, the other components, and the components can also *communicate* (or interact) with each other to synchronize their behaviour or to exchange information. In recent decades there has been much interest in and demand for concurrent systems such as, for example, communication networks, air traffic controllers and industrial plant control systems. As concurrent systems are often very complex and essential in our everyday life, it is vital that they are highly reliable. Therefore, there is a growing need for formal description languages and software tools that can assist us in the design and construction of reliable concurrent systems. The module will provide students with the opportunity to study the language CCS and how it can be used to describe, design
and verify simple concurrent and communicating systems.

**Detailed Syllabus**

*Introduction.* An introduction to concurrent and distributed systems, the notions of concurrency, communication and mobility, and a motivation for a formal theory of communication and concurrency.

*Modelling concurrency and communication.* An introduction, by means of examples, to the basic ideas and principles involved in the modelling of concurrency and communication. Transition rules, inference trees and transition graphs.

*Process algebra.* Syntax and operational semantics of CCS.

*Equational laws and algebraic reasoning.* Equational laws for CCS and their justification. Techniques for equational reasoning.

*Bisimulation.* Strong and weak bisimulations, strong and weak congruences (observational congruence). Techniques for establishing bisimulation equivalences, including (strong and weak) bisimulation games, differences and relationships between various bisimulation relations. Compositional reasoning.

*Case studies.* Specifications and designs of simple concurrent systems in CCS.

**Reading List**


**Resources** Course notes, text books in library, study guide, worksheets, handouts, past examination papers, module web pages, lecture rooms with OHPs, laboratories with PCs and demonstrators, surgeries.

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

Aims The aim of the Mathematics and Computer Science project is for the student to combine skills acquired in the other Mathematics and Computer Science modules in the production of a substantial project. In doing this, the student will assimilate information from a variety of sources and demonstrate the ability to pursue independent study.

It is intended that the project should produce some end product for users other than the author. A collection of course exercises, a literature search or a descriptive evaluation would not be suitable.

During the first part of the semester, the student will establish the lines of enquiry to be followed and produce a plan of the work to be carried out. The rest of the semester will be devoted to designing and implementing the end product and writing a final report detailing the progress made.

Learning Outcomes Students will be able to establish the nature of the deliverables to be produced by the project, to plan the timescales involved in developing these, and to identify the design issues involved. They will be able to undertake appropriate specification and design, and be able to implement an end product. They will be able to test and evaluate the end product.

Methods Individual research, meetings with supervisors.

Assessment Plan, viva, effort and final report.

Skills

Aims To teach students scientific writing and problem solving skills.

Learning Outcomes Students will be able to produce a plan of timescales for project work. They will be able to demonstrate general problem solving skills, and will be able to write a substantial written report on the project.

Methods Individual research, meetings with supervisors.

Assessment The assessment of CO3014 is broken down as follows:

1. 5%: Project plan document.
2. 10%: Oral examination and demonstration of software.
3. 80%: Final project report.
4. 5%: Mark for student effort and participation, based on a weekly diary.

Explanation of Prerequisites The idea of the project is that the student should develop and build on material which has already been learned, so it is important that a reasonable amount of second year study should have been undertaken.

Course Description The aim of the Mathematics and Computer Science project is for the student to combine skills acquired in the other Mathematics and Computer Science modules in the production
of a substantial project. In doing this, the student will assimilate information from a variety of sources and demonstrate the ability to pursue independent study.

It is intended that the project should produce some end product for users other than the author. A collection of course exercises, a literature search or a descriptive evaluation would not be suitable.
CO3015 Computer Science Project

Credits: 40
Convenor: Dr. Stuart Kerrigan
Semester: 1 + 2

Prerequisites: Essential: CO2006, CO2015
Desirable: 40 other credits of Computer Science Modules

Assessment:
Coursework: 100%
Examination: 0%

Lectures: 5 hours
Surgeries: 10 hours
Laboratories: 2 hours
Private Study: 283 hours

Subject Knowledge

Aims  Students will select a project topic chosen from an area of Computer Science that interests them, and then conduct two semesters worth of individual study of that topic, resulting in a substantial written dissertation. Projects should be of a problem solving nature; typically they will provide a software solution to a practical computing problem.

It is intended that the project should also produce an end product, usually a software system, for users other than the author. Further, a theoretical essay, a literature search, or a descriptive evaluation, by themselves, would not be suitable.

Learning Outcomes  Students will be able to demonstrate that they can carry out significant background research which underpins project work; work out the nature of the deliverables to be produced; identify the specification and design issues involved; undertake appropriate specification and design work; and implement the end (software) product according to their design work. They will be able to test and evaluate the end product. They will also be able to produce a substantial written dissertation.

Methods  Individual research, meetings with supervisors.

Assessment  Assessed by a project plan; oral presentation; interim report; two interviews; viva; effort, participation and organization; and final report (dissertation).

Skills

Aims  To teach students planning, scientific writing and problem solving skills.

Learning Outcomes  Students will be able to produce a plan of timescales for project work. Students will also be able to prepare and deliver a lecture style oral presentation, and be able to produce a short interim report on progress made to date and any revisions made to their original plan. They will be able to demonstrate general problem solving skills, and will be able to write substantial written reports.

Methods  Individual research, meetings with supervisors.

Assessment  Assessed by oral presentation; interim report; two interviews; viva; effort, participation and organization; and final report (dissertation).

Explanation of Prerequisites  All Computer Science students will have a common core of knowledge on which to build in the third year.

Course Description  The purpose of the Computer Science Project is for the student to combine knowledge and skills acquired in level one and two Computer Science modules in the production of a suitable project. Project work consists of independent private study, guided by regular short meetings with a member of staff who will advise the student on how to proceed with the year’s work. Students may choose a project title and subject area from a large list of project descriptions, or they may suggest a project of their own for possible approval. The project has a number of goals which the student
must achieve, but the key ones are the writing of a dissertation summarising the year’s work, and the development of a practical computer system.
CO3016 Computing Project

Credits: 40  Convenor: Dr. Stuart Kerrigan  Semester: 1 + 2

Prerequisites: Essential: 240 credits of Computing Modules

Assessment:

- Coursework: 100%
- Examination: 0%
- Private Study: 283 hours

Lectures: 5 hours
Surgeries: 10 hours
Laboratories: 2 hours

Subject Knowledge

Aims  Students will select a project topic chosen from an area of Computing that interests them, and then conduct thirty credits worth of individual study of that topic, resulting in a substantial written dissertation. Projects should be of a problem solving nature; typically they will provide a software solution to a practical computing problem.

It is intended that the project should also produce an end product, usually a software system, for users other than the author. Further, a theoretical essay, a literature search, or a descriptive evaluation, by themselves, would not be suitable.

Learning Outcomes  Students will be able to demonstrate that they can carry out background research which underpins project work; work out the nature of the deliverables to be produced; identify the specification and design issues involved; undertake appropriate specification and design work; and implement the end (software) product according to their design work. They will be able to test and evaluate the end product. They will also be able to produce a substantial written dissertation.

Methods  Individual research, meetings with supervisors.

Assessment  Assessed by a project plan; oral presentation; two interviews; viva; effort, participation and organization; and final report (dissertation).

Skills

Aims  To teach students planning, scientific writing and problem solving skills.

Learning Outcomes  Students will be able to produce a plan of timescales for project work. Students will also be able to prepare and deliver a lecture style oral presentation, and be able to produce a short interim report on progress made to date and any revisions made to their original plan. They will be able to demonstrate general problem solving skills, and will be able to write substantial written reports.

Methods  Individual research, meetings with supervisors.

Assessment  Assessed by oral presentation; two interview; viva; effort, participation and organization; and final report (dissertation).

Explanation of Prerequisites  All Computing students will have a common core of knowledge on which to build in the third year.

Course Description  The purpose of the Computing Project is for the student to combine knowledge and skills acquired in level one and two Computer Science modules in the production of a suitable project. Project work consists of independent private study, guided by regular short meetings with a member of staff who will advise the student on how to proceed with the year’s work. Ten credits of work will take place in semester one, and the remaining twenty credits in semester two. Students may choose a project title and subject area from a large list of project descriptions, or they may suggest a project of their own for possible approval. The project has a number of goals which the student must achieve,
but the key ones are the writing of a dissertation summarising the year’s work, and the development of a practical computer system.
CO3090 Distributed Systems and Applications

Credits: 20  Convenor: Dr. E. Tuosto  Semester: 2nd

| Assessment: | Coursework: 40%  | Three hour exam in May/June: 60% |
| Lectures: | 30 hours |
| Surgeries: | 10 hours |
| Laboratories: | 10 hours |
| Private Study: | 100 hours |

Subject Knowledge

Aims This course intends to equip students with notions for analysing/designing distribution of data and computations when designing and programming applications. The overall goal is to provide a critical understanding of distributed applications and middlewares.

Learning Outcomes Students will be able to: tackle distributed programming issues and analyse problems that require distribution of resources/computations; analyse and choose among the middleware models described in the course; understand and tackle issues like multi-threading and transactional interactions in distributed application; apply principles of component-based distributed programming (e.g., with respect to technologies like RMI, J2EE, etc.).

Methods Class sessions, textbook, worksheets, additional hand-outs and web support.

Assessment Marked coursework, traditional written examination.

Skills

Aims To teach students the basic principles of distributed computing and their middlewares.

Learning Outcomes Students will be able to: pinpoint the main features of middlewares for distributed applications; to identify essential elements that enable them to choose amongst the various type of middlewares; to apply prior knowledge on programming, designing and implementing to distributed applications development; to implement and evaluate a planned solution.

Methods Class sessions together with worksheets.

Assessment Marked coursework together with worksheets.

Explanation of Prerequisites A relevant aspect of the module is the reinforcement of material delivered in lectures with practicals involving students in the critical analysis of the middlewares presented in the lectures. Knowledge of Java, as provided in CO1003 and CO1004, is essential (inheritance, interfaces, exceptions). A basic knowledge of networks, client-server architecture and socket programming, as provided in CO2017, is required (and assumed).

Course Description Computer networks and distributed applications have a paramount role in all-day life. Nowadays, it is hard to imagine stand-alone systems or applications. Practically, any modern computing device offers the possibility of being connected with other devices. At higher level, applications aim at exploiting networking capabilities of systems and tend to be more and more interconnected and communicating themselves.

Designing and programming this kind of distributed applications can result a hard task if not done at the appropriate level of abstraction. There are two main complex aspects to face: (i) distributed systems are frequently made of heterogeneous devices and interact through many different communication infrastructure; (ii) modern distributed systems have different tiers (e.g., TCP/IP level, operating system, network system, etc.). Middlewares provide an abstraction of many low-level details of systems. They
are meant to simplify software development and application integration by interfacing the application level with lower tier of distributed systems so that the programmer does not have to worry about implementation details. Also, middlewares allow the programmer to integrate applications developed for different execution context and in different times.

The course reviews some notions of concurrent and distributed programming (e.g., threads and RMI) and presents the main models and principles behind the middlewares that in the last years many vendors (Microsoft, IBM, Sun, Oracle) have proposed. In fact, these proposals differ each other not only with respect to the technologies or architectures adopted, but also with respect to the underlying coordination models.

**Detailed Syllabus**  
Introduction to distributed systems; Programming with threads; RPC and JAVA/RMI; Message oriented Middlewares; Event/Notification; Distributed coordination.

**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs.

**Module Evaluation**  
Course questionnaires, course review.
## CO3094 System Modelling

**Credits:** 20  
**Convenor:** Dr. F.J. de Vries  
**Semester:** 1st

<table>
<thead>
<tr>
<th>Prerequisites:</th>
<th>Essential: CO1003, CO1012, CO2006</th>
</tr>
</thead>
</table>
| Assessment:    | Coursework: 40%  
Three hour exam in January: 60% |
| Lectures:      | 20 hours |
| Surgeries:     | 10 hours |
| Laboratories:  | 20 hours |
| Private Study: | 100 hours |

### Subject Knowledge

**Aims**  
This module aims to give students a hands-on experience with methods, languages and software tools used in industry for the specification and design of a wide range of systems (software, hardware, business process/workflow, embedded systems, etc.).

**Learning Outcomes**  
We will study two different type of specification languages that are often used in conjunction for describing different aspects of systems: the notation Z, which is suitable for describing transformational aspects, i.e. how the actions of a system operate on data; and the process language CSP, which allows to describe the reactive behaviour of systems, i.e. the processes as part of which actions are executed and the communication channels through which processes communicate.

**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 to write system specifications and problem solving skills.

**Learning Outcomes**  
Students will be able to: write system specifications at various levels of abstraction; to solve abstract and concrete problems (both routine seen, and simple unseen).

**Methods**  
Class and practical sessions together with worksheets.

### Explanation of Prerequisites

No specific knowledge is required, but some familiarity with the notation of Sets and Predicate Logic as well as general program design skills will be helpful.

### Course Description

We will study and practice with two different type of specification languages that are often used in conjunction for describing different aspects of systems: the notation Z, which is suitable for describing transformational aspects, i.e. how the actions of a system operate on data; and the process language CSP, which allows to describe the reactive behaviour of systems, i.e. the processes as part of which actions are executed and the communication channels through which processes communicate.

Both languages come with various tools to verify properties of the specifications and to help in the refinement process from specification to program.

### Detailed Syllabus

**Background:** Lack of formalism in software development can lead to a loss of precision and correctness in the resulting software. Formal specification languages and their tools can improve this situation.

Z is a state based language for specifying systems. We will study: Notation of proposition and predicate
logic, sets, relations and schemas; simple refinement; object-Z; producing Z code with LaTeX; type checking Z code with the tool FUZZ

**Communicating Sequential Processes**: Notation, semantics, refinement, verification; tool support ProBe and FDR.

**Reading List and Tools**

Material on Z


Material on CSP


**Tools**

- Fuzz: [http://spivey.oriel.ox.ac.uk/mike/zrm/](http://spivey.oriel.ox.ac.uk/mike/zrm/)
- Probe and FDR2: [http://www.fsel.com/software.html](http://www.fsel.com/software.html)

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

**Module Evaluation**  
Course questionnaires, course review.
CO3095 Software Measurement and Quality Assurance

Credits: 20  Convenor: Dr. H. Janicke  Semester: 1st

Prerequisites: Essential: C01003, C01005, C01007  or  EG207+EG223, CO2006, CO2012, CO2015  Desirable: CO1019

Assessment: Coursework: 40%  Three hour exam in January: 60%

Lectures: 30 hours  Problem Classes: 5 hours
Surgeries: 10 hours  Private Study: 105 hours

Subject Knowledge

Aims  The module approaches the issue of quality assurance in the software development process at an advanced level. This includes a rigorous account of the strategies for software testing and quality control, and the introduction of software metrics for quality assurance and project cost estimation. The module is focussed around the notion of software process improvement.

Learning Outcomes  Students will be able to describe how quality issues affect each aspect of the software development life-cycle. They will be able to choose appropriate strategies for software testing and validation, and discuss how to implement them. They will be able to demonstrate understanding of the theory of software metrics and be able to make software measurements in practice. They will be able to relate quality to the current standards for process improvement.

Methods  Class sessions together with course notes; recommended textbooks; worksheets; additional hand-outs including articles, case studies and research papers; web resources.

Assessment  Marked coursework, group presentation, written examination.

Skills

Aims  Students will learn how to research current issues in software quality assurance, and how to present their findings. They will learn how to turn theoretical ideas into practical process improvement steps in an industrial context.

Learning Outcomes  Students will be able to research a given topic using a variety of sources including books, current articles and research papers and web-resources. They will be able to give a written account of their findings (suitable for inclusion in a company report). They will be able to give a seminar-style presentation of their findings using appropriate audio-visual aids.

Methods  Class sessions together with worksheets. Library and web resources.

Assessment  Marked coursework, group presentation, written examination.

Explanation of Prerequisites

The module will build on many of the ideas that are introduced in C01006 and CO2006 so these modules are essential. The sections on testing will also require experience of programming in Java. Software Engineering is as much a practical subject as an intellectual discipline so it is an advantage to have had the experience of working in a team on the CO2015 Software Engineering Project.

Course Description

A major aim of Software Engineering is to ensure the quality of the final product of the software development process. Quality is not an extra that can be added at the end of the process. To achieve it one must consider how each stage contributes to the quality of the final product. Quality assurance should be seen as an intrinsic part of the software life-cycle. One key mechanism of quality control is
software testing, another is inspections and reviews. However, these should be implemented as part of a wider Quality Assurance Plan.

The first step towards quality is to understand what it is and how to measure it. The overall quality of a product is a rather vague idea that cannot be measured directly. It can be seen as an amalgamation of different attributes: correctness, reliability, maintainability, ease of use, and so on; which can be measured by developing the right software metrics. The use of metrics is thus an important tool in quality assurance. Furthermore, quality cannot be considered without reference to the associated cost. Metrics can also be used to gauge the size and complexity of software and hence are employed in project cost estimation.

This module will look in depth at the issues of software quality assurance from an industrial perspective, exploring the techniques available and how these might be employed. It will focus around the idea of software process improvement, as seen in the SEI process Capability Maturity Model and SPICE (ISO/IEC 15504).

**Detailed Syllabus**

**Quality** Quality issues in the life-cycle model. Quality planning and management. Quality at the requirements stage: negotiation, setting achievable goals. Aspects of quality: reliability, maintainability, correctness, usability. Risk analysis and management.


**Inspection** Code walk-through, inspections, reviews. Comparison of different approaches. Effective follow-up: collection of data, review feedback.

**Testing** Different levels: unit testing, integration testing, system testing, validation. Structural testing: coverage techniques. Behavioural testing: domain testing, finite state testing. Mutation and fault seeding. Tools and instrumentation.


**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with OHP and data-projector, past examination papers.

**Module Evaluation**  
Course questionnaires, course review.
**CO3096 Compression Methods for Multimedia**

<table>
<thead>
<tr>
<th>Credits: 20</th>
<th>Convenor: Prof. R. Raman</th>
<th>Semester: 2nd</th>
</tr>
</thead>
</table>

**Prerequisites:**
- Essential: CO1012 or CO1011
- Desirable: CO1016, CO2016

**Assessment:**
- Coursework: 30%
- Three hour exam in January: 70%

<table>
<thead>
<tr>
<th>Lectures: 30 hours</th>
<th>Surgeries: 5 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Classes: 5 hours</td>
<td>Class Tests: 3 hours</td>
</tr>
<tr>
<td>Private Study: 107 hours</td>
<td></td>
</tr>
</tbody>
</table>

**Subject Knowledge**

**Aims**
To study methods for compression of symbolic data as well as audio, image and video data. To gain an appreciation of the ubiquity and importance of compression technologies.

**Learning Outcomes**
Students should achieve: broad knowledge of compression techniques as well as the mathematical foundations of data compression; factual knowledge about existing compression standards or commonly-used compression utilities; understanding of the ubiquity and importance of compression technologies in today’s environment; elementary understanding of the need for modeling data and the underlying issues.

**Methods**
Class sessions together with course notes, recommended textbooks, problem classes with worksheets and model solutions, web support.

**Assessment**
marked courseworks, class tests using Blackboard VLE, traditional written examination.

**Skills**

**Aims**
To teach students how to compute basic statistics of data, and how to apply nontrivial algorithms to real-world problems.

**Learning Outcomes**
Students will be able to: understand and describe various models of data; understand the basic data compression algorithms and show how they work on a particular input; implement these algorithms; compare their efficiency in terms of speed and compression ratio.

**Methods**
Class sessions and problem classes.

**Assessment**
marked coursework, class tests, traditional written examination.

**Explanation of Prerequisites**
There are two main prerequisites. Firstly, students should have some knowledge of how data of various kinds (numbers, characters, images and sound) are represented digitally in uncompressed format. This will be reviewed rapidly at the start of the course. Some elementary mathematics is also required. In particular, trigonometry: basic functions–cos, sin and measuring angles in radians; probability: basic definitions and expected values; matrices: transposition and multiplication and recurrence relations: basic familiarity. Basic familiarity with the elements of computer systems and networks is also desirable.

**Course Description**
Data compression is about finding novel ways of representing data so that it takes very little storage, with the proviso that it should be possible to reconstruct the original data from the compressed version. Compression is essential when storage space is at a premium or when data needs to be transmitted and bandwidth is at a premium (which is almost always). The first thing that one learns about compression is that it is not “one size fits all” approach: the essence of compression is to determine characteristics of the data that one is trying to compress (typically one is looking for patterns that one can exploit to get a compact representation). This gives rise to a variety of data modeling and representation techniques, which is at the heart of compression. The convergence of the communications,
computing and entertainment industries has made data compression a part of everyday life (e.g. MP3, DVD and Digital TV) and has thrown up a number of exciting new opportunities for new applications of compression technologies.

**Detailed Syllabus**  

**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with a computer to CFS, data projector, two OHPs, past courseworks and examination papers.

**Module Evaluation**  
Module questionnaires, course review.
Subject Knowledge

Aims The aim of this course is to teach the students the technologies and techniques for creating large-scale systems on the WWW. We consider these large scale distributed systems in the context of how they emerged before concentrating on two specific aspects: Java servlets and XML data representation and processing.

Learning Outcomes At the end of the course the student should be able to: Understand the architectural foundations for Web Technologies, use XML and AJAX based techniques appropriately to create documents and handle data, be aware of security and session handling issues and use supporting techniques, understand Java servlet technology and use it to create web applications.

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

Assessment Assessed coursework, traditional written examination.

Skills

Aims To teach students problem solving skills.

Learning Outcomes Students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen).

Methods Class sessions together with worksheets.

Explanation of Prerequisites No specific knowledge is required, but an understanding of Database (SQL) and Programming in Object Oriented Paradigms (Java) as well as general program design skills will be helpful.

Course Description Software engineering in the time the internet and e-commerce provides challenges that go beyond what is taught in traditional software engineering courses. In particular we are dealing with a large, distributed system that is not under particular control by anyone. This course discusses the issues that are relevant for designing useful, stable and secure systems in this context highlighting many of the currently prevailing technologies.

The course takes students from a background of 'traditional' middleware to the emerging paradigm of Service Oriented Computing. We introduce scalable techniques for developing applications for the web (e.g. JavaServlets, .net) – by both discussing their respective merits as well as getting hands-on experience in writing applications using these techniques.

One important aspect of web applications, that also occurs in enterprise application integration, is to deal with different data formats, and the de-facto standard these days is XML and its related technologies. XML Schema, XPATH and Style Sheets (XSL), AJAX as well as DOM and SAX as programming paradigms will be explored.

The course concludes with placing the previous two parts into the context of Service Oriented Archi-
tecture, by looking at Web Services and discussing why they are the next generation technology for distributed (web) applications.

**Detailed Syllabus**

**Background:** The emergence of web technologies in the context of distributed computing, supporting architectures, static and dynamic content provisioning techniques and standards.

**Current Web data standards:** XML related technologies, such as AJAX, DTD, XML Schema, XLink, XSLT, as well as Java programming support for them

**Security and session handling:** session handling with cookies, sessions with servlet session APIs

**Java servlets:** designing and deploying servlets

**Web Services:** Web Services motivation

**Reading List**


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

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

Aims  This course will equip students with the knowledge required to build cryptographically secure applications in Java, and the knowledge of common security problems and solutions in Internet applications.

Learning Outcomes  Students will be able to: describe the working principles of modern public-key cryptosystems; build cryptographically secure network applications using Java’s cryptographic extensions; and describe the basic security principles of some internet applications relying on cryptographic mechanisms.

Methods  Class sessions together with lecture slides, recommended textbook, worksheets, printed solutions, and some additional hand-outs and web support.

Assessment  Marked coursework, traditional written examination.

Skills

Aims  To teach students how to methodically solve problems given the techniques available to them.

Learning Outcomes  Students will be able to: breakdown a problem to identify essential elements; apply prior knowledge of subject to analyze problems; design a plan to solve a problem; implement a planned solution and evaluate the implementation.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework, traditional written examination.

Explanation of Prerequisites  A significant aspect of the module will be the reinforcement of material delivered in lectures with practicals involving students implementing cryptographically secure network applications in Java. Hence a basic knowledge of Java, as provided in CO1003 and CO1004, will be essential. A basic knowledge in networks, client-server architecture and Java socket programming, as provided in CO2017, will be very useful. A basic grounding in discrete mathematics will be helpful during the lectures on cryptography.

Course Description  The use of computers and computer networks, in particular the Internet, is becoming an integral part of our lives in different application areas, such as e-banking and e-commerce. This has given us numerous advantages and convenience. However, at the same time, the security of computer systems becomes a critical issue. How can computer systems defend themselves against network attacks? How can we ensure that our data have not been tampered with, or disclosed without our consent? How can we be sure of the identity of the party whom we are communicating with? These are some of the security issues that must be addressed properly. This module will provide students with knowledge of the security issues in computer systems.

A fundamental part of security systems is cryptography, the science of secret writing. There have been rapid advances in cryptography in the past few decades, and cryptography has become an integral part of many commercial computer applications. The module will explain the principles of modern
public key cryptography, a cornerstone of many security-enabled network applications in current use. A number of cryptographic primitives, including message digests, digital signatures and certificates, will be discussed. The module will go through all details of how to write secure network applications using these cryptographic primitives.

The course presents the security model of Java introducing elements of its access control model (e.g., Security manager and policies). Also, a few notation and techniques for the analysis of cryptographic protocols commonly adopted in distributed applications are introduced. Such techniques are used to argue about security aspects of some amongst the most popular applications of cryptographic protocols (e.g., Pretty Good Privacy and digital signatures).

**Detailed Syllabus**

Cryptography: Security issues and concerns. Key management including generation, translation and agreement protocols (Diffie-Hellman). Principles of classical symmetric ciphers. Modern symmetric and asymmetric ciphers including DES, RSA. Block cipher concepts e.g. chaining and padding. Authentication and integrity with message digests, MAC’s, signatures, and certificates. Simple cryptographic protocols, e.g. bit commitment. Java Support: Java Cryptography Architecture (JCA), Java Cryptography Extension (JCE).


**Reading List**


**Resources** Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

**Module Evaluation** Course questionnaires, course review.
CO3120 Computing with Management Project

Credits: 40  Convenor: Dr. Stuart Kerrigan  Semester: 1 + 2

Prerequisites: Essential: 240 credits of Computing/Management Modules

Assessment: Coursework: 100%  Examination: 0%

Lectures: 5 hours
Surgeries: 10 hours  Private Study: 283 hours
Laboratories: 2 hours

Subject Knowledge

Aims Students will select a project topic chosen from an area of Computing (possibly with connections to Management or Business) that interests them, and then conduct thirty credits worth of individual study of that topic, resulting in a substantial written dissertation. Projects should be of a problem solving nature; typically they will provide a software solution to a practical computing problem.

It is intended that the project should also produce an end product, usually a software system, for users other than the author. Further, a theoretical essay, a literature search, or a descriptive evaluation, by themselves, would not be suitable.

Learning Outcomes Students will be able to demonstrate that they can carry out background research which underpins project work; work out the nature of the deliverables to be produced; identify the specification and design issues involved; undertake appropriate specification and design work; and implement the end (software) product according to their design work. They will be able to test and evaluate the end product. They will also be able to produce a substantial written dissertation.

Methods Individual research, meetings with supervisors.

Assessment Assessed by a project plan; oral presentation; two interviews; viva; effort, participation and organization; and final report (dissertation).

Skills

Aims To teach students planning, scientific writing and problem solving skills.

Learning Outcomes Students will be able to produce a plan of timescales for project work. Students will also be able to prepare and deliver a lecture style oral presentation, and be able to produce a short interim report on progress made to date and any revisions made to their original plan. They will be able to demonstrate general problem solving skills, and will be able to write substantial written reports.

Methods Individual research, meetings with supervisors.

Assessment Assessed by oral presentation; two interview; viva; effort, participation and organization; and final report (dissertation).

Explanation of Prerequisites All Computing with Management students will have a common core of knowledge on which to build in the third year.

Course Description The purpose of the Computing with Management Project is for the student to combine knowledge and skills acquired in level one and two Computer Science and Management modules in the production of a suitable project. Project work consists of independent private study, guided by regular short meetings with a member of staff who will advise the student on how to proceed with the year’s work. Ten credits of work will take place in semester one, and the remaining twenty credits in semester two. Students may choose a project title and subject area from a large list of project descriptions, or they may suggest a project of their own for possible approval. The project has a number
of goals which the student must achieve, but the key ones are the writing of a dissertation summarising the year’s work, and the development of a practical computer system.
Subject Knowledge

Aims  This module teaches the principles of C++ programming and the design of algorithms on modern computer architectures.

Learning Outcomes  Students should be able to: understand the components of a C++ program, the structures required to write basic algorithms, algorithm analysis and design, modern computer architectures and memory hierarchies and some algorithms for scientific computing.

Methods  Class sessions, recommended textbook and worksheets.

Assessment  Marked coursework, written examination.

Skills

Aims  To develop design, analysis and problem solving skills.

Learning Outcomes  Students will be able to mathematically analyse the computing performance of algorithms. They will be able to design algorithms to solve computing problems.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework and traditional written examination.

Explanation of Prerequisites  It is assumed that students are already familiar with a programming language such as Fortran, Java or C. Students with little previous programming experience will be required to attend 24 additional hours of programming lectures and laboratories at the start of the course.

Course Description  Over the past 15 years C++ has become one of the world’s most popular programming languages, due to its potential for producing efficient and compact code. As such any scientist wishing to develop efficient programs should be familiar with the use of its central features. This module is intended to give the student a basic grasp of its use for scientific computing.

Algorithms are traditionally designed, analysed and compared on a theoretical computer model. However an algorithm that is fast on the theoretical model may be relatively slow on modern computers. This module introduces the memory hierarchies on modern computer architectures and its effects on algorithm performance. The module then teaches how to design fast algorithms on modern computers.

Detailed Syllabus

• Introduction to C++.
• Classes, constructors and destructors.
• Pointers, arrays and references.
• Methods and operators.
• Overloading.
• Templates and inheritance.
• Introduction to LAPACK++.
• Algorithm analysis and design.
• Modern computer architectures and memory hierarchies.
• Design of algorithms for matrix operations, polynomials and the fast fourier transform.

**Reading List**


**Resources**  Course notes, web page, study guide and worksheets.

**Module Evaluation**  Course questionnaires, course review.
CO4015 MComp Computer Science Project

Credits: 30  Convenor: Dr. Stuart Kerrigan  Semester: 1 + 2

Prerequisites: Essential: CO2006, CO2015, CO3015 or CO3016 or CO3120
Desirable: 45 other credits of Computer Science Modules

Assessment: Coursework: 100%  Examination: 0%

Lectures: 0 hours  Surgeries: 0 hours  Private Study: 300 hours
Laboratories: 0 hours

Subject Knowledge

Aims  Students will select a project topic chosen from a challenging area of Computer Science that interests them, and then conduct two semesters worth of individual study of that topic, resulting in a substantial written dissertation that aspires to research level quality. Projects should be of a problem solving nature incorporating Masters level material; sometimes they will provide a software solution to a practical computing problem, or may be more theoretical in nature.

It is intended that the project should also produce an end product, usually (but certainly not limited to) a software system, for users other than the author. Further, a theoretical essay, a literature search, or a descriptive evaluation, by themselves, would not be suitable.

Learning Outcomes  Students will be able to demonstrate that they can carry out significant background research which underpins project work; work out the nature of the deliverables to be produced; identify the specification and design issues involved; undertake appropriate specification and design work; and implement the end (software) product according to their design work. They will be able to test and evaluate the end product. They will also be able to produce a substantial written dissertation of near-research level quality at minimum.

Methods  Individual research, meetings with supervisors.

Assessment  Assessed by a project plan; oral presentations; preliminary report; viva; effort, participation and organization; and final report (dissertation).

Skills

Aims  To teach students planning, scientific writing and problem solving skills.

Learning Outcomes  Students will be able to produce a plan of timescales for project work. Students will also be able to prepare and deliver a lecture style oral presentation, and be able to produce a short interim report on progress made to date and any revisions made to their original plan. They will be able to demonstrate general problem solving skills, and will be able to write substantial written reports.

Methods  Individual research, meetings with supervisors.

Assessment  Assessed by oral presentation; preliminary report; viva; effort, participation and organization; and final report (dissertation).

Explanation of Prerequisites  All Computer Science students will have a common core of advanced knowledge on which to build in the fourth year.

Course Description  The purpose of the MComp Computer Science Project is for the student to combine knowledge and skills acquired in level one, two and three Computer Science modules in the production of a suitable project. Project work consists of independent private study, guided by regular short meetings with a member of staff who will advise the student on how to proceed with the year's
work. Students may choose a project title and subject area from a large list of project descriptions, or they may suggest a project of their own for possible approval. The project has a number of goals which the student must achieve, but the key ones are the writing of a dissertation summarising the year’s work, and the development of a practical computer system.
CO7002 Analysis and Design of Algorithms

Credits: 15  Convenor: Dr. S. Fung  Semester: 2nd

Prerequisites: none
Assessment: Coursework: 30%  Three hour exam in May/June: 70%
Lectures: 24 hours  Problem Classes: 8 hours
Surgeries: 8 hours  Class Tests: 1 hour
Class Tests: 8 hours  Private Study: 71.5 hours

Subject Knowledge

Aims The module aims to introduce students to the design of algorithms as a means of problem-solving. Students will learn how to analyze the complexity of algorithms. Major algorithm design techniques will be presented and illustrated with fundamental problems in computer science and engineering. Students will also learn the limits of algorithms and how there are still some problems for which it is unknown whether there exist efficient algorithms.

Learning Outcomes Students should be able to demonstrate how the worst-case time complexity of an algorithm is defined; compare the efficiency of algorithms using asymptotic complexity; design efficient algorithms using standard algorithm design techniques; demonstrate a number of standard algorithms for problems in fundamental areas in computer science and engineering such as sorting, searching, and problems involving graphs.

Methods Class sessions together with lecture slides, recommended textbook, worksheets, printed solutions, and web support.

Assessment Marked coursework, class test, traditional written examination.

Skills

Aims Students will become more experienced in the application of logical and mathematical tools and techniques in computing. They will develop the skills of using standard algorithm design techniques to develop efficient algorithms for new problems. They will develop skills to judge the quality of the algorithms.

Learning Outcomes Students will be able to solve problems which are algorithm based by using various design techniques. They will be able to apply prior knowledge of standard algorithms to solve new problems, and mathematically evaluate the quality of the solutions. They will be able to produce concise technical writing for describing the solutions and arguing their correctness.

Methods Class sessions together with worksheets.

Assessment Marked coursework, class test, traditional written examination.

Explanation of Prerequisites Typical materials assumed for this module are: the basic notions associated with an imperative programming language such as arrays, while loops, for loops, linked lists, recursion, etc.; and logical and discrete mathematical notions such as induction, asymptotic notation, recurrence relations and their solution, geometric and arithmetic series, etc.

Course Description This module introduces students to the design and analysis of algorithms. Algorithms are step-by-step procedures, such as those executed by computers, to solve problems. Typical problems include, for example, “what is the shortest path between two locations in a network?”, or “what is the maximum set of activities that can be chosen subject to time constraints?” Just because a problem can be solved, does not mean that there exists a practically time-efficient solution. It is the goal of algorithm designers to develop better and better algorithms for the solution of fundamental or
new problems. The main methods used to design algorithms will be illustrated through examples of fundamental importance in computer science and engineering. These design methods not only apply to the problems illustrated in the module, but also to a much wider range of problems in computer science and engineering. As a result, students can apply the design methods learned to other problems they encounter. Alternatively, it can be the case that no algorithms of a certain quality exist; algorithm designers then need to identify this limitation of algorithms. Techniques for analyzing the efficiency of algorithms and the inherent complexities of problems will be explained.

**Detailed Syllabus**  
Asymptotic analysis of algorithms: the notion of asymptotic complexity using big-O notation; solving recurrence relations; master theorem; limitations of algorithms (lower bounds using decision trees).

Algorithm design techniques: divide and conquer; greedy algorithms; dynamic programming.

Algorithms for fundamental problems: sorting (mergesort, Quicksort); searching (binary search); minimum spanning trees (Kruskal’s and Prim’s algorithms); graph traversal; shortest paths (Dijkstra’s algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm); network flow (Ford-Fulkerson algorithm).

**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

**Module Evaluation**  
Course questionnaires, course review.
CO7007 Communication and Concurrency

Credits: 15  Convenor: Dr. I. Ulidowski  Semester: 1st

Prerequisites: none

Assessment: Coursework: 40%  Three hour exam in May/June: 60%

Lectures: 30 hours  Class Tests: 1 hours
Surgeries: 10 hours  Private Study: 67.5 hours
Laboratories: 4 hours

Subject Knowledge

Aims
This module provides students with an introduction to theories and applications of concurrency. In particular, it will familiarise students with the process algebras CCS (Calculus of Communicating Systems) and its operational semantics. The module will teach, via individual and collective work, how to specify, design and implement simple concurrent systems. Students will also learn how to verify correctness of concurrent systems using the Concurrency Workbench software tool.

Learning Outcomes
Students should be able to: demonstrate understanding of the notions of concurrency, communication, and concurrent systems; and CCS and its operational and axiomatic semantics; They should be able to develop informal and formal specifications of simple concurrent systems, and be able to produce systems’ designs from specifications; able to reason about the behaviour of simple concurrent systems, both by hand and with the aid of the Concurrency Workbench, using the techniques of equational reasoning and bisimulation.

Methods
Class and laboratory sessions together with course notes (available on the Web and in the printed form), Bisimulation Games workshop, recommended textbooks, the Concurrency Workbench manual, class and laboratory worksheets, printed solutions, and Web support.

Assessment
Marked problem-based worksheets, class tests, and traditional problem-based written examination.

Skills

Aims
To teach students problem solving and scientific writing skills.

Learning Outcomes
Students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen); write short summaries of technical material.

Methods
Class and laboratory sessions, course notes and text books, software manuals, class and laboratory worksheets, printed solutions.

Assessment
Marked problem-based worksheets, class tests, and traditional problem-based written examination.

Explanation of Prerequisites
Basic knowledge of discrete mathematics and logic is essential.

Course Description
A concurrent system is a system consisting of several components such that each component acts concurrently with, and independently of, the other components, and the components can also communicate (or interact) with each other to synchronize their behaviour or to exchange information. In recent decades there has been much interest in and demand for concurrent systems such as, for example, communication networks, air traffic controllers and industrial plant control systems. As concurrent systems are often very complex and essential in our everyday life, it is vital that they
are highly reliable. Therefore, there is a growing need for formal description languages and software tools that can assist us in the design and construction of reliable concurrent systems. The module will provide students with the opportunity to study the language CCS and the software tool Concurrency Workbench of New Century, and how they can be used to describe, design and verify simple concurrent and communicating systems.

**Detailed Syllabus**

*Introduction.* An introduction to concurrent and distributed systems, the notions of concurrency, communication and mobility, and a motivation for a formal theory of communication and concurrency.

*Modelling concurrency and communication.* An introduction, by means of examples, to the basic ideas and principles involved in the modelling of concurrency and communication. Transition rules, inference trees and transition graphs.

*Process algebra.* Syntax and operational semantics of CCS.

*Equational laws and algebraic reasoning.* Equational laws for CCS and their justification. Techniques for equational reasoning.

*Bisimulation.* Strong and weak bisimulations, strong and weak congruences (observational congruence). Techniques for establishing bisimulation equivalences, including strong and weak bisimulation games, differences and relationships between various bisimulation relations. Compositional reasoning.

*Case studies.* Specifications and designs of simple concurrent systems in CCS. Verification of correctness using the Concurrency Workbench tool.

**Reading List**


**Resources** Course notes, text books in library, study guide, worksheets, handouts, past examination papers, module web pages, lecture rooms with OHPs, laboratories with PCs and demonstrators, the Concurrency Workbench software tool, surgeries.

**Module Evaluation** Course questionnaires, course review.
CO7090 Distributed Systems and Applications

Credits: 15  Convenor: Dr. E. Tuosto  Semester: 2nd

Prerequisites: Essential: Knowledge of Java and/or Object Oriented programming; basic knowledge of computer networks (client-server architectures, socket programming)

Desirable: Basic concepts of operating systems

Assessment: Coursework: 40%  Three hour exam in May/June: 60%

Lectures: 24 hours  Private Study: 74.5 hours

Surgeries: 7 hours

Laboratories: 7 hours

Subject Knowledge

Aims  This course intends to equip students with notions for analysing/designing distribution of data and computations when designing and programming applications. The overall goal is to provide a critical understanding of distributed applications and middlewares.

Learning Outcomes  Students will be able to: tackle distributed programming issues and analyse problems that require distribution of resources/computations; analyse and choose among the middleware models described in the course; understand and tackle issues like multi-threading and transactional interactions in distributed application; apply principles of component-based distributed programming (e.g., with respect to technologies like RMI, J2EE, etc.).

Methods  Class sessions, textbook, worksheets, additional hand-outs and web support.

Assessment  Marked coursework, traditional written examination.

Skills

Aims  To teach students the basic principles of distributed computing and their middlewares.

Learning Outcomes  Students will be able to: pinpoint the main features of middlewares for distributed applications; to identify essential elements that enable them to choose amongst the various type of middlewares; to apply prior knowledge on programming, designing and implementing to distributed applications development; to implement and evaluate a planned solution.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework, traditional written examination.

Explanation of Prerequisites  A relevant aspect of the module is the reinforcement of material delivered in lectures with practicals involving students in the critical analysis of the middlewares presented in the lectures. Knowledge of Java, as provided in CO1003 and CO1004, is essential (inheritance, interfaces, exceptions). A basic knowledge of networks, client-server architecture and socket programming, as provided in CO2017, is required (and assumed).

Course Description  Computer networks and distributed applications have a paramount role in all-day life. Nowadays, it is hard to imagine stand-alone systems or applications. Practically, any modern computing device offers the possibility of being connected with other devices. At higher level, applications aim at exploiting networking capabilities of systems and tend to be more and more interconnected and communicating themselves.

Designing and programming this kind of distributed applications can result a hard task if not done at the appropriate level of abstraction. There are two main complex aspects to face: (i) distributed
systems are frequently made of heterogeneous devices and interact through many different communication infrastructure; (ii) modern distributed systems have different tiers (e.g., TCP/IP level, operating system, network system, etc.). Middlewares provide an abstraction of many low-level details of systems. They are meant to simplify software development and application integration by interfacing the application level with lower tier of distributed systems so that the programmer does not have to worry about implementation details. Also, middlewares allow the programmer to integrate applications developed for different execution context and in different times.

The course reviews some notions of concurrent and distributed programming (e.g., threads and RMI) and presents the main models and principles behind the middlewares that in the last years many vendors (Microsoft, IBM, Sun, Oracle) have proposed. In fact, these proposals differ each other not only with respect to the technologies or architectures adopted, but also with respect to the underlying coordination models.

**Detailed Syllabus**  
Introduction to distributed systems; Programming with threads; RPC and JAVA/RMI; Message oriented Middlewares; Event/Notification; Distributed coordination.

**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs.

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

Aims  This module aims to give students a hands-on experience with methods, languages and software tools used in industry for the specification and design of a wide range of systems (software, hardware, business process/workflow, embedded systems, etc.).

Learning Outcomes  We will study two different type of specification languages that are often used in conjunction for describing different aspects of systems: the notation Z, which is suitable for describing transformational aspects, i.e. how the actions of a system operate on data; and the process language CSP, which allows to describe the reactive behaviour of systems, i.e. the processes as part of which actions are executed and the communication channels through which processes communicate.

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 to write system specifications and problem solving skills.

Learning Outcomes  Students will be able to: write system specifications at various levels of abstraction; to solve abstract and concrete problems (both routine seen, and simple unseen).

Methods  Class and practical sessions together with worksheets

Explanation of Prerequisites  No specific knowledge is required, but some familiarity with the notation of Sets and Predicate Logic as well as general program design skills will be helpful.

Course Description  We will study and practice with two different type of specification languages that are often used in conjunction for describing different aspects of systems: the notation Z, which is suitable for describing transformational aspects, i.e. how the actions of a system operate on data; and the process language CSP, which allows to describe the reactive behaviour of systems, i.e. the processes as part of which actions are executed and the communication channels through which processes communicate.

Both languages come with various tools to verify properties of the specifications and to help in the refinement process from specification to program.

Detailed Syllabus

Background: Lack of formalism in software development can lead to a loss of precision and correctness in the resulting software. Formal specification languages and their tools can improve this situation.

Z  is a state based language for specifying systems. We will study: Notation of proposition and predicate
logic, sets, relations and schemas; simple refinement; object-Z; producing Z code with LaTeX; type checking Z code with the tool FUZZ

**Communicating Sequential Processes**: Notation, semantics, refinement, verification; tool support ProBe and FDR.

**Reading List and Tools**

Material on Z

- J. M. Spivey. *The Z Notation: a reference manual*, Prentice Hall, 2001:
  http://spivey.oriel.ox.ac.uk/mike/zrm/


Material on CSP

- C.A.R. Hoare, *Communicating Sequential Processes (CSP)*, Prentice Hall, 1985:


Tools

- Fuzz: http://spivey.oriel.ox.ac.uk/mike/zrm/

- Probe and FDR2: http://www.fsel.com/software.html

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

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

Aims The module approaches the issue of quality assurance in the software development process at an advanced level. This includes a rigorous account of the strategies for software testing and quality control, and the introduction of software metrics for quality assurance and project cost estimation. The module is focussed around the notion of software process improvement.

Learning Outcomes Students will be able to describe how quality issues affect each aspect of the software development life-cycle. They will be able to choose appropriate strategies for software testing and validation, and discuss how to implement them. They will understand the theory of software metrics and be able to make software measurements in practice. They will be able to relate quality to the current standards for process improvement.

Methods Class sessions together with course notes; recommended textbooks; worksheets; additional hand-outs including articles, case studies and research papers; web resources.

Assessment Marked coursework, group presentation, written examination.

Skills

Aims Students will learn how to research current issues in software quality assurance, and how to present their findings. They will learn how to turn theoretical ideas into practical process improvement steps in an industrial context.

Learning Outcomes Students will be able to research a given topic using a variety of sources including books, current articles and research papers and web-resources. They will be able to give a written account of their findings (suitable for inclusion in a company report). They will be able to give a seminar-style presentation of their findings using appropriate audio-visual aids.

Methods Class sessions together with worksheets. Library and web resources.

Assessment Marked coursework, group presentation, written examination.

Course Description A major aim of Software Engineering is to ensure the quality of the final product of the software development process. Quality is not an extra that can be added at the end of the process. To achieve it one must consider how each stage contributes to the quality of the final product. Quality assurance should be seen as an intrinsic part of the software life-cycle. One key mechanism of quality control is software testing, another is inspections and reviews. However, these should be implemented as part of a wider Quality Assurance Plan.

The first step towards quality is to understand what it is and how to measure it. The overall quality of a product is a rather vague idea that cannot be measured directly. It can be seen as an amalgamation of different attributes: correctness, reliability, maintainability, ease of use, and so on; which can be measured by developing the right software metrics. The use of metrics is thus an important tool in quality assurance. Furthermore, quality cannot be considered without reference to the associated cost. Metrics can also be used to gauge the size and complexity of software and hence are employed in project cost estimation.
This module will look in depth at the issues of software quality assurance from an industrial perspective, exploring the techniques available and how these might be employed. It will focus around the idea of software process improvement, as seen in the SEI process Capability Maturity Model and SPICE (ISO/IEC 15504).

**Detailed Syllabus**

**Quality** Quality issues in the life-cycle model. Quality planning and management. Quality at the requirements stage: negotiation, setting achievable goals. Aspects of quality: reliability, maintainability, correctness, usability. Risk analysis and management.


**Inspection** Code walk-through, inspections, reviews. Comparison of different approaches. Effective follow-up: collection of data, review feedback.

**Testing** Different levels: unit testing, integration testing, system testing, validation. Structural testing: coverage techniques. Behavioural testing: domain testing, finite state testing. Mutation and fault seeding. Tools and instrumentation.


**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with OHP and data-projector, past examination papers.

**Module Evaluation**  
Course questionnaires, course review.
CO7096 Compression Methods for Multimedia

Credits: 15  Convenor: Prof. R. Raman  Semester: 2nd

| Prerequisites: | Essential: Computer Science MSc students are assumed to have the required background. Students should consult the convenor in case of any doubt. |
| Assessment: | Coursework: 30%  Three hour exam in May/June: 70% |
| Lectures: | 30 hours |
| Surgeries: | 5 hours |
| Problem Classes: | 5 hours |
| Class Tests: | 3 hours |
| Private Study: | 69.5 hours |

Subject Knowledge

Aims  To study methods for compression of symbolic data as well as audio, image and video data. To gain an appreciation of the ubiquity and importance of compression technologies.

Learning Outcomes  Students should achieve: broad knowledge of compression techniques as well as the mathematical foundations of data compression; factual knowledge about existing compression standards or commonly-used compression utilities; understanding of the ubiquity and importance of compression technologies in today’s environment; elementary understanding of the need for modeling data and the underlying issues.

Methods  Class sessions together with course notes, recommended textbooks, problem classes with worksheets and model solutions, web support.

Assessment  marked courseworks, class tests, traditional written examination.

Skills

Aims  To teach students how to compute basic statistics of data, and how to apply nontrivial algorithms to real-world problems.

Learning Outcomes  Students will be able to: understand and describe various models of data; understand the basic data compression algorithms and show how they work on a particular input; implement these algorithms; compare their efficiency in terms of speed and compression ratio.

Methods  Class sessions and problem classes.

Assessment  marked coursework, class tests Blackboard, traditional written examination.

Explanation of Prerequisites  There are two main prerequisites. Firstly, students should have some knowledge of how data of various kinds (numbers, characters, images and sound) are represented digitally in uncompressed format. This will be reviewed rapidly at the start of the course. Some elementary mathematics is also required. In particular, trigonometry: basic functions cos, sin and measuring angles in radians; probability: basic definitions and expected values; matrices: transposition and multiplication and recurrence relations: basic familiarity. Basic familiarity with the elements of computer systems and networks is also desirable.

Course Description  Data compression is about finding novel ways of representing data so that it takes very little storage, with the proviso that it should be possible to reconstruct the original data from the compressed version. Compression is essential when storage space is at a premium or when data needs to be transmitted and bandwidth is at a premium (which is almost always). The first thing that one learns about compression is that it is not “one size fits all” approach: the essence of compression is to determine characteristics of the data that one is trying to compress (typically one is looking for patterns that one can exploit to get a compact representation). This gives rise to a variety of data modeling and representation techniques, which is at the heart of compression. The convergence of the communications,
computing and entertainment industries has made data compression a part of everyday life (e.g. MP3, DVD and Digital TV) and has thrown up a number of exciting new opportunities for new applications of compression technologies.

**Detailed Syllabus**  

**Reading List**


**Resources**  
Course notes, web page, study guide, worksheets, handouts, lecture rooms with a computer to CFS, data projector, two OHPs, past courseworks and examination papers.

**Module Evaluation**  
Course questionnaires, course review.
CO7098 Web Technologies

Credits: 15  Convenor: S. Kerrigan  Semester: 1st

Prerequisites: none

Assessment: Coursework: 40%  Three hour exam in January: 60%

Lectures: 20 hours
Surgeries: 10 hours
Laboratories: 20 hours
Private Study: 62.5 hours

Subject Knowledge

Aims The aim of this course is to teach the students the technologies and techniques for creating large-scale systems on the WWW. We consider these large scale distributed systems in the context of how they emerged before concentrating on two specific aspects: Java servlets and XML data representation and processing.

Learning Outcomes At the end of the course the student should be able to: Understand the architectural foundations for Web Technologies, use XML and AJAX based techniques appropriately to create documents and handle data, be aware of security and session handling issues and use supporting techniques, understand Java servlet technology and use it to create web applications.

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

Assessment Assessed coursework, traditional written examination.

Skills

Aims To teach students problem solving skills.

Learning Outcomes Students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen).

Methods Class sessions together with worksheets.

Explanation of Prerequisites No specific knowledge is required, but an understanding of Database (SQL) and Programming in Object Oriented Paradigms (Java) as well as general program design skills will be helpful.

Course Description Software engineering in the time the internet and e-commerce provides challenges that go beyond what is taught in traditional software engineering courses. In particular we are dealing with a large, distributed system that is not under particular control by anyone. This course discusses the issues that are relevant for designing useful, stable and secure systems in this context highlighting many of the currently prevailing technologies.

The course takes students from a background of traditional' middleware to the emerging paradigm of Service Oriented Computing. We introduce scalable techniques for developing applications for the web (e.g. JavaServlets, .net) – by both discussing their respective merits as well as getting hands-on experience in writing applications using these techniques.

One important aspect of web applications, that also occurs in enterprise application integration, is to deal with different data formats, and the de-facto standard these days is XML and its related technologies. XML Schema, XPATH and Style Sheets (XSL), AJAX as well as DOM and SAX as programming paradigms will be explored.

The course concludes with placing the previous two parts into the context of Service Oriented Archi-
tecture, by looking at Web Services and discussing why they are the next generation technology for distributed (web) applications.

**Detailed Syllabus**

**Background:** The emergence of web technologies in the context of distributed computing, supporting architectures, static and dynamic content provisioning techniques and standards.

**Current Web data standards:** XML related technologies, such as AJAX, DTD, XML Schema, XLink, XSLT, as well as Java programming support for them

**Security and session handling:** session handling with cookies, sessions with servlet session APIs

**Java servlets:** designing and deploying servlets

**Web Services:** Web Services motivation

**Reading List**


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

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

Aims  This course will equip students with the knowledge required to build cryptographically secure applications in Java, and the knowledge of common security problems and solutions in Internet applications.

Learning Outcomes  Students will be able to: describe the working principles of modern public-key cryptosystems; build cryptographically secure network applications using Java’s cryptographic extensions; and describe the basic security principles of some internet applications relying on cryptographic mechanisms.

Methods  Class sessions together with lecture slides, recommended textbook, worksheets, printed solutions, and some additional hand-outs and web support.

Assessment  Marked coursework, traditional written examination.

Skills

Aims  To teach students how to methodically solve problems given the techniques available to them.

Learning Outcomes  Students will be able to: breakdown a problem to identify essential elements; apply prior knowledge of subject to analyze problems; design a plan to solve a problem; implement a planned solution and evaluate the implementation.

Methods  Class sessions together with worksheets.

Assessment  Marked coursework, traditional written examination.

Explanation of Prerequisites  A significant aspect of the module will be the reinforcement of material delivered in lectures with practicals involving students implementing cryptographically secure network applications in Java. Hence a basic knowledge of Java will be essential. A basic knowledge in networks, client-server architecture and Java socket programming, will be very useful. A basic grounding in discrete mathematics will be helpful during the lectures on cryptography.

Course Description  The use of computers and computer networks, in particular the Internet, is becoming an integral part of our lives in different application areas, such as e-banking and e-commerce. This has given us numerous advantages and convenience. However, at the same time, the security of computer systems becomes a critical issue. How can computer systems defend themselves against network attacks? How can we ensure that our data have not been tampered with, or disclosed without our consent? How can we be sure of the identity of the party whom we are communicating with? These are some of the security issues that must be addressed properly. This module will provide students with knowledge of the security issues in computer systems.

A fundamental part of security systems is cryptography, the science of secret writing. There have been rapid advances in cryptography in the past few decades, and cryptography has become an integral part of many commercial computer applications. The module will explain the principles of modern public key cryptography, a cornerstone of many security-enabled network applications in current use. A
number of cryptographic primitives, including message digests, digital signatures and certificates, will be discussed. The module will go through all details of how to write secure network applications using these cryptographic primitives.

The course presents the security model of Java introducing elements of its access control model (e.g., Security manager and policies). Also, a few notation and techniques for the analysis of cryptographic protocols commonly adopted in distributed applications are introduced. Such techniques are used to argue about security aspects of some amongst the most popular applications of cryptographic protocols (e.g., Pretty Good Privacy and digital signatures).

**Detailed Syllabus**  
Cryptography: Security issues and concerns. Key management including generation, translation and agreement protocols (Diffie-Hellman). Principles of classical symmetric ciphers. Modern symmetric and asymmetric ciphers including DES, RSA. Block cipher concepts e.g. chaining and padding. Authentication and integrity with message digests, MAC’s, signatures, and certificates. Simple cryptographic protocols, e.g. bit commitment. Java Support: Java Cryptography Architecture (JCA), Java Cryptography Extension (JCE).


**Reading List**


**Resources** Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHP’s, past examination papers, past tests.

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

**Aims**  This module teaches the principles of C++ programming and the design of algorithms on modern computer architectures.

**Learning Outcomes**  Students should be able to: understand the components of a C++ program, the structures required to write basic algorithms, algorithm analysis and design, modern computer architectures and memory hierarchies and some algorithms for scientific computing.

**Methods**  Class sessions, recommended textbook and worksheets.

**Assessment**  Marked coursework, written examination.

**Skills**

**Aims**  To develop design, analysis and problem solving skills.

**Learning Outcomes**  Students will be able to mathematically analyse the computing performance of algorithms. They will be able to design algorithms to solve computing problems.

**Methods**  Class sessions together with worksheets.

**Assessment**  Marked coursework and traditional written examination.

Explanation of Prerequisites  It is assumed that students are already familiar with a programming language such as Fortran, Java or C. Students with little previous programming experience will be required to attend 24 additional hours of programming lectures and laboratories at the start of the course.

Course Description  Over the past 15 years C++ has become one of the world’s most popular programming languages, due to its potential for producing efficient and compact code. As such any scientist wishing to develop efficient programs should be familiar with the use of its central features. This module is intended to give the student a basic grasp of its use for scientific computing.

Algorithms are traditionally designed, analysed and compared on a theoretical computer model. However an algorithm that is fast on the theoretical model may be relatively slow on modern computers. This module introduces the memory hierarchies on modern computer architectures and its effects on algorithm performance. The module then teaches how to design fast algorithms on modern computers.

Detailed Syllabus

- Introduction to C++.
- Classes, constructors and destructors.
- Pointers, arrays and references.
- Methods and operators.
• Overloading.
• Templates and inheritance.
• Algorithm analysis and design.
• Modern computer architectures and memory hierarchies.
• Design of algorithms for matrix operations, polynomials and the fast fourier transform.

Reading List


Resources  Course notes, web page, study guide and worksheets.

Module Evaluation  Course questionnaires, course review.
CO7200 Algorithms for Bioinformatics

Credits: 15  Convenor: Prof. Thomas Erlebach  Semester: 2nd

Prerequisites: Desirable: Java programming

Assessment: Coursework: 40%  1.5 hour exam in May/June: 60%
Lectures: 24 hours
Laboratories: 12 hours
Problem Classes: 4 hours
Private Study: 72.5 hours

Subject Knowledge

Aims  This module introduces students to the algorithmic solution of computational problems in bioinformatics, including aspects of probabilistic modelling.

Learning Outcomes  Students should be able to: describe a number of computational problems arising in bioinformatics; state and discuss algorithmic approaches to the solution of such problems; discuss probabilistic models underlying computational tasks in bioinformatics; design and implement efficient algorithms; apply modelling and algorithm design to the solution of bioinformatics problems.

Methods  Class sessions together with course notes, recommended textbooks, worksheets, and some additional hand-outs and web support.

Assessment  Marked problem-based worksheets and programming assignments, traditional written problem-based examination.

Skills

Aims  To teach students scientific writing, modelling and problem solving skills.

Learning Outcomes  Students will be able to: write short, clear, note based, summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen).

Methods  Class sessions together with worksheets.

Assessment  Marked problem-based worksheets, traditional written examination.

Explanation of Prerequisites  A basic understanding of discrete mathematics and probability will be helpful.

Course Description  Processing biological data requires complex computations on large volumes of data. To ensure that these computations complete within a reasonable amount of time, one must design the algorithms (computer procedures) after a careful study of the characteristics of underlying data and making use of existing algorithm design principles. This module aims to introduce students to the algorithmic solution of computational problems in bioinformatics. Students will learn a number of probabilistic models that underlie the formulation of biological data processing tasks as computational problems, and will be introduced to efficient computer algorithms for solving, and some key principles for designing efficient algorithms for solving, these problems.

Detailed Syllabus

- Introduction to algorithms
- String matching
- Pairwise sequence alignment
- Hidden Markov models
• Restriction site mapping
• Multiple sequence alignment
• Phylogenetic trees
• Genome rearrangement

**Reading List**


**Resources**  Web page, study guide, worksheets, handouts, lecture rooms with OHP and data projector.

**Module Evaluation**  Course questionnaires, course review.
CO7201 Individual Project

Credits: 60  Convenor: Dr. N. Walkinshaw and Dr. N. Piterman  Semester: Spring or Summer

Prerequisites: Essential: Successful completion of taught part of the course.
Assessment: Coursework: 100%

Private Study: 450 hours

Subject Knowledge

Aims  The aim of this module is to demonstrate a student’s ability to undertake a substantial investigation of a technical problem and its domain, by evaluating tools and methods, and by developing a professional information technology project; also, it intends to give students the opportunity to

- show individual creativity and originality;
- analyse information from a critical point of view;
- apply, where appropriate, and go beyond, where necessary, the knowledge and skills taught throughout the course;
- investigate/solve new and/or intellectually demanding problems (from specification through implementation and critical evaluation of results);
- conduct and sustain a complex argument in a coherent and lucid fashion.

Learning Outcomes  The main learning outcomes of this module are to initiate, plan, manage and deliver a substantial information technology project. Upon successful completion of the project, students will be able to:

- select suitable methods and tools for analysing a substantial problem and for developing a computer-based solution for it, within known constraints;
- access, retrieve and organize information relevant to the problem under study by making use of resources such as the internet and textbooks, but also of scholarly articles published in journals and conferences;
- prepare a project plan and conduct regular reviews of the plan;
- present a properly referenced, well-structured dissertation, in a format suitable for professional dissemination;
- communicate effectively in a presentation environment;
- perform a critical reflection of the achievements in the project after its completion.

Methods  After examinations, an individual project is undertaken full-time. Students choose a topic to work on and explore it by privately studying under the supervision of a member of the academic staff. The project is driven by a challenging problem to be solved.

Assessment  Meeting reports (diaries), preliminary report, interim report, dissertation, software artifacts and viva. Typically, a dissertation usually contains 10,000 to 12,000 words (40-60 pages).

Skills
Learning Outcomes Among other transferable skills, successful students will improve the following skills: solving practical and abstract problems, communication, writing, editing, searching/gathering/evaluating information, developing evaluation strategies and managing time.

Methods Development of technical deliverables, including software artifacts, specifications, written reports (preliminary report, interim report, meeting reports), dissertation, and oral presentation.

Assessment Meeting reports (diaries), preliminary report, interim report, dissertation, software artifacts and viva. Typically, a dissertation usually contains 10,000 to 12,000 words (40-60 pages).

Course Description After examinations, an individual research project can be undertaken full-time by the students who qualify to do so. The Individual Project is carried out under the supervision of a member of the academic staff that students are invited to contact as early as possible to supervise their work. Students may wish to complement the foundational material of the first two terms with practical, applied work during the project. It is possible to involve informal collaboration with other organisations, subject to previous approval of the project supervisor.

The individual project will lead to submission of a dissertation where the original elements of the student research are described. Projects can involve a number of activities that range from substantial programming to purely theoretical research. In any case, projects must be at postgraduate level; they are expected to contain some element of original work and cannot simply be a review of literature.

Resources Relevant resources for the individual project depend on the project topic. General resources are

1. Module web page: https://campus.cs.le.ac.uk/teaching/resources/CO7201/
2. Calendar: https://campus.cs.le.ac.uk/teaching/resources/CO7201/#calendar
3. SVN Local Pages: https://campus.cs.le.ac.uk/demonstrators/info/VCS/
4. MSc Course handbook: http://www.cs.le.ac.uk/admissions/masters/MScCourseHandbook.pdf
5. Information about plagiarism: https://campus.cs.le.ac.uk/ForStudents/plagiarism/Plagiarism.html
6. Student Learning Centre: http://www.le.ac.uk/offices/ssds/sd/
7. Enhancing writing: http://www2.le.ac.uk/offices/ssds/slc/help-with/dissertations
8. Presentation skills: http://www2.le.ac.uk/offices/ssds/slc/resources/presentation/index

Module Evaluation Course questionnaires, course review.

Reading List

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 Complexity is a recurrent issue in Software Engineering. Since the early days of IT, building complex software systems has been a major challenge and the problems that it raises keep making headlines on the press. In order to address these problems, more and more emphasis is being put into techniques that support the higher levels of system design, namely advanced modelling languages and software architecture. These levels allow us to move away from code and understand how systems are required to operate based on more abstract models. Models reflect an architecture for the software system in the sense that they reflect an organisation based on components that perform relatively simple computations and connectors that coordinate the interactions between the components. This module provides an introduction to the topic of software architecture in general and modelling techniques for service-oriented architecture in particular.

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 software organisational principles of service-oriented architectures in particular; model systems using a service-oriented modelling language.

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: decompose system requirements according to the principles of service-oriented computing; modularise service applications by identifying the dependencies that services have on several kinds of external parties; model service orchestrations; model abstract semantic interfaces for services; model the protocols that coordinate service interactions; model service-level agreements.

Methods Class and lab 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 Complexity is a recurrent issue in Software Engineering. Since the early days of IT, building complex software systems has been a major challenge and the problems that it raises keep making headlines on the press. In order to address these problems, more and more emphasis is being put into techniques that support the higher levels of system design, namely advanced modelling languages and software architecture. These levels allow us to move away from code and understand...
how systems are required to operate based on models. Models reflect an abstract architecture for the software system in the sense that they are organised in terms of components that perform relatively simple computations and connectors that coordinate the interactions between the components. This module provides an introduction to the topic of software architecture in general and modelling techniques for service-oriented architecture in particular.

**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 for Service-Oriented Architectures** Use-case diagrams for capturing requirements on business activities and services: the roles that different parties can play in a service-oriented system organisation. Conversational protocols and patterns for service-oriented interactions. Orchestration of complex services. Service-level agreements as constraints on service discovery and selection.

**Reading List**


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

**Module Evaluation** Course questionnaires, course review.
CO7206 System Re-engineering

Credits: 15  Convenor: Neil Walkinshaw  Semester: 1st

Prerequisites: none
Assessment: Coursework: 40%  Two hour exam in January: 60%
Lectures: 20 hours
Surgeries: 7 hours
Laboratories: 14 hours
Private Study: 71.5 hours

Subject Knowledge

Aims To understand the problems and issues associated with legacy software systems and the methods used in reverse engineering, comprehending, maintaining, migrating, evolving and reengineering them.

Learning Outcomes By the end of the module, students should be able to: understand software ageing phenomenon and the issues related to it; understand the challenges in renovating and maintaining legacy software systems and the available methods for dealing with them; make reasoned decisions on which reengineering methods to apply for certain types of legacy system renovation tasks.

Methods Class sessions, tutorials and lab sessions together with course notes, course readings, assignments and class tests.

Assessment Assignment, class test, written examination.

Skills

Aims To develop analytical and problem solving skills in dealing with legacy systems and software integration challenges. To develop on hands experience in reverse engineering and reengineering existing software systems.

Learning Outcomes By the end of the module, students will be able to apply the methods learned to assess the situation of a small-scale legacy system and decide a suitable reengineering strategy for it, in the light of the objectives of the reengineering/renovation effort. They will learn how to reverse engineer and reengineer moderate size legacy software systems using some of the available commercial and research tools.

Methods Class sessions, tutorials, lab sessions and industrial tutorials by industry experts together with course notes, course readings, assignments and class tests.

Assessment Two marked assignments, class test.

Explanation of Prerequisites This module assumes that the student has reasonable programming experience in more than one high level language, preferably Java, C#, or C++. It assumes that the student has done some non-trivial programming tasks and has some understanding of the challenge inherent in trying to understand and modify old software or software that was written by other people.

Course Description Software development is not always a "green-fields" process. More often than not, new software engineers are hired to maintain and evolve existing systems, not to develop new ones. If a new system is to be developed, it has to be integrated with other existing "legacy" software systems. Legacy systems are valuable software systems that are still in use but are difficult to maintain, change or migrate because they were developed with technologies of the past and/or because they were not engineered properly. Very often, these systems were developed without proper documentation, version control, or proper design. Many such systems had undergone numerous changes by different people that violate the original system design, if any ever existed. As a result, it is challenging to understand,
modify or migrate these systems. Fresh software developers are usually neither equipped with the
necessary skills nor have the desire to work with these software "legacies". Fresh software development
is usually considered superior to software maintenance and reengineering. The year 2000 problem and
the deployment of the Euro gave rise to research and practice of software system reverse engineering
and reengineering. Reengineering is "the examination of a subject system to reconstitute it in a new
form and the subsequent implementation of the new form." [Chikofsky, and Cross II, 1990]. Part of any
reengineering efforts is a reverse engineering process, which is "the process of analyzing a subject system
with two goals in mind: (1) to identify the system’s components and their interrelationships; and, (2)
to create representations of the system in another form or at a higher level of abstraction." [Chikofsky,
and Cross II, 1990].

In the Internet era, it is very important to have the skills to deal with legacy systems because it is not
always the case that Web applications will be developed from scratch. In many cases it is required to
open the available information systems to Web access or integrate them with other Web applications.

This module is an introduction to the main issues related to software systems ageing and evolution.
We will examine some of the available methods and technologies for software reverse engineering and
reengineering as well as some of the managerial and planning issues specific to software reengineering
projects.

**Detailed Syllabus**  This module will cover the following topics.

**Software Ageing:** How and why software systems age.

**Legacy Systems:** Their issues and challenges.

**Introduction to Software Evolution, Maintenance and Reengineering.**

**Reverse Engineering:** Program Analysis, Architecture Recovery, Software Complexity and Mainte-
nance Metrics, Program Visualization.

**Forward Engineering:** Refactoring, Code Transformation, Reengineering.

**Software Reengineering Strategies and Management.**

**Reading List**


2005.

[A] Oscar Nierstrasz, Stéphane Ducasse, Serge Demeyer, *Object-Oriented Reengineering Patterns*,


[A] Raihan Al-Ekram and Kostas Kontogiannis, *An XML-based Framework for Language Neutral Pro-
gram Representation and Generic Analysis*, 9th European Conference on Software Maintenance
and Reengineering (CSMR 2005), Manchester, UK; March 21 - 23, 2005.

Wesley.

gorization*, Oquendo (ed.), Proc. of European Conf. Software Architecture (ECSA). LNCS 4758,

neering Projects Fail*, Software Engineering Institute.

*The Department of Computer Science*
Resources Course notes.

Module Evaluation Course questionnaires, course review.
Subject Knowledge

Aims  The functionality and size of software systems grow in time. This requires more adequate methods for system development. This module covers the main principles and techniques of generative development. It focuses on system modelling and code generation. The module also covers the foundations of aspect-oriented programming (AOP).

Learning Outcomes  At the end of this course, successful students will be able to: be aware of the main approaches for automating software development; critically evaluate the role of modelling and code generation in software development; use UML and OCL for designing views of software systems; check the consistency of the UML design of an application; use techniques for model-driven development; explain concepts of aspect-oriented programming and apply them.

Methods  Class sessions together with course notes, textbooks, printed solutions, and some additional hand-outs and web support.

Assessment  Assessed coursework, traditional written examination.

Skills

Aims  To teach students abstraction and higher-level modelling skills and to provide the basic skills required to use generative development methods. In particular model-driven development techniques and aspect-oriented programming.

Learning Outcomes  At the end of this course, successful students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen); model and specify software systems; use state of the art tools for code generation; use aspect-oriented programming; develop software in a systematic, automated way.

Methods  Class sessions together with worksheets and practical programming experience.

Assessment  Computer-based exercises, traditional written examination.

Explanation of Prerequisites  Basic knowledge of UML and Java is desirable.

Course Description  Software engineering is a very dynamically developing discipline. There are new specification, modelling and programming languages, new tools and paradigms for development of software systems. To the most promising new ideas in recent years are:

- **UML** for modelling of software systems
- **Generative methods** for code generation
- **Aspect-oriented programming** for compositional development of complex systems
- **Model-driven development** for software system development, e.g., OMG’s Model-Driven Architecture (MDA) initiative.
Detailed Syllabus

The course will provide a broad picture of new developments in the area of modelling and code generation. It will teach methods of proper system modelling using UML diagrams, methodical system development from UML model to implementation using generative methods, the principles of Aspect Oriented Programming and MDA. In this course we will use state of the art software tools.

Reading List


Resources

Study guide, worksheets, lecture rooms with data projector, computer lab access, handouts.

Module Evaluation

Course questionnaires, course review.
CO7209 Software Reliability

Credits: 15  Convenor: Dr. A. Kurz  Semester: 1st

Prerequisites: none
Assessment: Coursework: 40%  Two hour exam in January: 60%
Lectures: 23 hours
Laboratories: 16 hours
Private Study: 73.5 hours

Subject Knowledge

Aims The aims of this module are to (1) present a collection of methods for dealing with software reliability; (2) explore in detail some of the methods and tools that have been developed in recent years for automatic verification of properties of software systems; (3) raise awareness to the limitations of current technologies, and how they may be overcome in the future.

Learning Outcomes This module introduces techniques and tools for verifying that computer systems are reliable in the sense that they have the properties intended. The module covers: languages for modelling systems and their properties; model checking and algorithms; a selection of tools (e.g. SPIN); specification, verification and validation of typical properties of reactive systems; limitations of automatic verification techniques; Relationship to software testing techniques (black-box checking).

Methods Class sessions, tutorials and laboratory 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 the role and nature of software reliability methods; to develop in the students the ability to separate concerns during system verification, namely in what concerns modelling software systems, specifying required properties, and applying model-checking based verification techniques.

Learning Outcomes On completion of this module, the student should be able to: use tools (e.g. SPIN) to verify and debug small-scale systems; understand and explain the principles and algorithms behind those tools; understand the application of the tools to different domains; understand the limitations of current verification techniques (e.g. the state explosion problem) and efforts to overcome them.

Methods Class sessions together with worksheets.

Explanation of Prerequisites Knowledge of Discrete Maths will be helpful.

Course Description Software reliability methods are now past the stage of “promising curiosities” and offer techniques and tools that can be employed in a variety of application domains to verify required functional properties against models of system behaviour.

This module covers some of these techniques and tools, namely those which, like SPIN, are based on model-checking. It introduces students to languages for modelling systems and their properties, some of the algorithms that can be employed for automatic verification, and the limitations of current implementations.

Detailed Syllabus

- Modelling Software systems with state transition systems.
- Introduction to propositional and temporal logic.
• Specifying software system properties with temporal logic.
• Automatic verification techniques based on model-checking.
• Relationship of model-checking and black-box testing.

Reading List


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

Module Evaluation  Course questionnaires, course review.
CO7210 Personal and Group Skills

Credits: 15  Convenor: Dr A. Murawski  Semester: 1st and 2nd

Prerequisites: none
Assessment: Group discussions: 20%  Oral presentation: 30%  Collective essay: 50%
Lectures: 10 hours  Seminars: 8 hours  Other: 8 hours  Private Study: 86.5 hours

Skills

Aims  The aim of this module is to provide students with skills that are way up the value chain of any IT employer: critical analysis, appraisal of evidence, communication, working relationships, managing learning, and research.

Learning Outcomes  At the end of the module, students should have improved their ability to:

• locate, organise and marshal evidence, report on findings, analyse complex ideas and construct sophisticated critical arguments;
• know how and when to draw on the knowledge and expertise of others;
• contribute and comment on ideas in syndicate groups;
• reflect on and write up results;
• plan and present research clearly and effectively using appropriate IT resources;
• deliver oral presentations to professional standard;
• respond to questioning;
• write cogently and clearly

Methods  Seminars by guest speakers, handouts and recommended texts, moderated group discussions, oral presentation, collective writing, workshops on transferable skills and career planning.

Assessment  Moderated group discussions, 3,000 collective essay, 10 minute oral presentation.

Explanation of Prerequisites  none

Course Description  This module combines attendance of seminars especially commissioned from speakers selected for their presentation skills and state-of-the-art research, group discussions and collective essay writing on topics selected for the seminars, as well as a series of workshops on transferable skills and career planning run by the Student Learning Centre of the university. At the end, students should have improved their ability to:

• locate, organise and marshal evidence, report on findings, analyse complex ideas and construct sophisticated critical arguments;
• know how and when to draw on the knowledge and expertise of others;
• contribute and comment on ideas in syndicate groups;
• reflect on and write up results;
• plan and present research clearly and effectively using appropriate IT resources;
• deliver oral presentations to professional standard;
• respond to questioning;
• write cogently and clearly

Resources       Seminars by guest speakers, hand-outs and recommended texts, moderated group discussions, oral presentation, collective writing, web support.

Module Evaluation    Course questionnaires, course review.
CO7211 Discrete Event Systems

Credits: 15  Convenor: Dr N Piterman  Semester: 2nd

Prerequisites: Desirable: There are no specific pre-requisites for this module. A basic knowledge of discrete mathematics is assumed and some experience with probability theory would be useful. Familiarity with programming would help put some of the aspects of the module into context.

Assessment: Coursework: 40%  Three hour exam in May/June: 60%

Lectures: 22 hours  Problem Classes: 4 hours
Surgeries: 8 hours  Private Study: 78.5 hours

Subject Knowledge

Aims To teach the theory and practice of Discrete Event Systems.

Learning Outcomes At the end of this module, students should be able to

• employ some basic formalisms of behavioural modelling (such as automata and Petri nets) to model real-world examples;
• discuss the basic theory of the formalisms, especially their uses, limitations and methods of verification.

Methods Lectures, surgeries, problem classes, worksheets, course notes, and textbook.

Assessment Marked worksheets, traditional written examination.

Skills

Aims The ability to apply mathematical formalisms to model and analyse event driven systems.

Learning Outcomes Students will be able to: write short, clear note-based summaries of technical knowledge; solve abstract and concrete problems (both routine seen and simple unseen), including the formal modelling of discrete systems.

Methods Lectures, surgeries, problem classes, worksheets, course notes and textbook.

Assessment Marked worksheets and traditional written examination.

Explanation of Prerequisites There are no specific pre-requisites for this module. A basic knowledge of discrete mathematics is assumed and some experience with probability theory would be useful. Familiarity with programming would help put some of the aspects of the module into context.

Course Description A discrete event system is a mathematical model of a system (such as computational device) that communicates with its environment by atomic actions (called events). For example, a user of the system pressing a button could send a signal to a controller. These events are assumed to be discrete in the sense that they occur instantaneously (as opposed to over a period of time).

The module will present an overview of various modelling and analysis techniques for discrete event systems. We start by looking at sequential systems (where no two events can occur simultaneously). Systems of this kind will be modelled by finite automata. This class is then extended to allow for events occurring simultaneously; these are modelled by Petri nets. Subsequently, we will study techniques that allow us to extract quantitative information about the behaviour of systems. This gives rise to the class of probabilistic systems (where we assume that a certain event occurs with a given probability) and we can then estimate the likelihood of situations such as system failure. Included in this section is an introduction to queuing theory.

The Department of Computer Science
Detailed Syllabus  

The module will be divided into three parts.

Automata and Languages. We study automata as models of sequential discrete event systems. Topics covered are:

- Modelling of Discrete Event Systems by Automata.
- Automata and Languages.
- Nondeterministic Automata.
- Operations on Automata and Modelling of Systems.
- Optimizing Automata.
- Limitations of Automata.

Petri Nets. Petri nets allow us to extend the class of systems to those where events can happen concurrently. Topics covered are:

- Petri Nets and Languages.
- Safety in Petri Nets.
- Comparison of the Petri Net Model and the Automata Model.

Markov Chains and Probabilistic Models. This class of models allows to analyse systems that embody uncertainty in the sense that events are only known to happen with a certain probability. Topics covered are:

- Review of Basic Probability Theory.
- Markov Chains as Discrete Event Systems.
- Introduction to Queueing Theory.

Reading List


Resources  
Study guide, worksheets, handouts, lecture rooms with whiteboards and a data projector.

Module Evaluation  
Module questionnaires, course review.
CO7212 Game Theory in Computer Science

Credits: 15  Convenor: Dr. Roy L. Crole  Semester: 2nd

Prerequisites: none
Assessment: Coursework: 40%  Two hour exam in May/June: 60%
Lectures: 24 hours
Surgeries: 8 hours
Problem Classes: 8 hours
Private Study: 72.5 hours

Subject Knowledge

Aims  This module teaches the fundamental concepts of game theory in the context of applications in computer science.

Learning Outcomes  Students should be able to: describe different mathematical models of games; state and discuss basic concept from game theory, such as Nash equilibria; calculate Nash equilibria of game trees and strategic form games; list a number of application areas of computer science where game theoretical models are relevant; apply methods from algorithmic game theory to the modelling and analysis of real-world problems.

Methods  Class sessions together with course notes, recommended textbooks, worksheets, and some additional hand-outs and web support.

Assessment  Marked problem-based worksheets, class tests, traditional written examination.

Skills

Aims  To teach students scientific writing, modelling and problem solving skills.

Learning Outcomes  Students will be able to: write short, clear, note based, summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen).

Methods  Class sessions together with worksheets.

Assessment  Marked problem-based worksheets, class tests, traditional written examination.

Explanation of Prerequisites  A basic understanding of discrete mathematics, calculus, linear algebra, and probability will be helpful.

Course Description  Modern computer science has to deal with large, heterogeneous networks in which a large number of autonomous agents interact. Often, such systems are not centrally planned, but evolve in a distributed fashion as a result of the interaction of agents. They can be modelled using concepts from game theory. This module introduces the basic concepts from game theory and discusses their use in the solution and modelling of problems faced by computer scientists. Examples include the prediction of the equilibrium state reached via the interaction of selfish users in a communication network, and the comparison of that state with a globally optimised state. Other examples include the design of mechanisms ensuring that individual players will behave in a way that achieves a desirable global state of the system.

Detailed Syllabus  The course will roughly cover the following topics:

- Part 1: An Introduction to Non-Cooperative Game Theory
  - Definition of game trees (extensive games with perfect information)
  - Backward induction
  - Strategies and strategy profiles
- Games in strategic form
- Symmetric games
- Dominance and elimination of dominated strategies
- Nash equilibrium
- Reduced strategies
- Subgame perfect Nash equilibrium (SPNE)
- Commitment games
- Bimatrix games
- Matrix notation of expected payoffs
- The best-response condition
- Existence of mixed equilibria
- Degenerate games
- Zero-sum games
- Extensive games with imperfect information
- Perfect recall

- Part 2: Some Topics at the Interface of Game Theory and Computer Science
  - The price of anarchy (coordination ratio) and the price of stability
  - Network routing games
    - Atomic and non-atomic routing games
    - Pigou’s example
    - Braess’s paradox
  - Network design games
  - Complexity of computing equilibria
  - Vickrey auctions
  - Applications of game theory in computer science

**Reading List**


**Resources** Web page, study guide, worksheets, handouts, copies of research articles, lecture rooms with OHP and data projector.

**Module Evaluation** Course questionnaires, course review.
CO7213 Networking and Distributed Computing

Credits: 15  Convenor: Prof. Thomas Erlebach  Semester: 1st

Prerequisites: Desirable: Java programming

Assessment: Coursework: 40%  Two hour exam in January: 60%

Lectures: 24 hours  Problem Classes: 5 hours
Surgeries: 5 hours  Class Tests: 3 hours
Laboratories: 3 hours  Private Study: 72.5 hours

Subject Knowledge

Aims  This module teaches the basic concepts of communication networks, the use of algorithms in the design and efficient operation of such networks, and the principles of distributed computing.

Learning Outcomes  Students should be able to: describe the layered architecture, routing mechanisms and protocols used in communication networks, in particular the Internet; describe mathematical models of optimisation problems arising in the design and operation of networks; implement simple distributed applications such as peer-to-peer network software.

Methods  Class sessions together with course notes, recommended textbooks, worksheets, and some additional hand-outs and web support.

Assessment  Marked problem-based worksheets and programming assignments, traditional written problem-based examination.

Skills

Aims  To teach students scientific writing, modelling and problem solving skills.

Learning Outcomes  Students will be able to: write short, clear, note based, summaries of technical knowledge; solve abstract and concrete problems (both routine seen, and simple unseen), including modelling aspects.

Methods  Class sessions together with worksheets.

Assessment  Marked problem-based worksheets, class tests, traditional written examination.

Explanation of Prerequisites  Basic understanding of discrete mathematics and some programming experience will be helpful.

Course Description  In today’s world, computers are no longer used in isolation, but are connected via networks. In particular, Internet-based communication is becoming more and more important for our society and economy. This module introduces the basic concepts of networking, discusses problems arising in the design and efficient operation of networks, and explains models and algorithmic techniques in distributed computing. It discusses the different network layers, routing mechanisms, and important protocols, focusing on the Internet but covering also certain aspects of wireless networks. It explains optimisation problems arising in networks, such as those related to routing and fault-tolerance. The design and analysis of algorithms for such problems are covered as well. The module briefly introduces models and basic principles for distributed computing, including architecture and implementation of peer-to-peer networks.

Detailed Syllabus  Introduction to basic networking concepts, including discussion of ISO/OSI network model with seven layers and TCP/IP network model with four layers. Basics of the socket programming interface.

Performance analysis including calculations with transmission rates, bandwidth, throughput and latency.
Direct link technologies, transmission of frames (including brief discussion of SONET), error detection, sliding window protocol, Ethernet, 802.11 wireless networks.


End-to-end protocols, in particular the use of TCP as a basis for reliable transport, and UDP for, e.g., real-time transmissions such as Voice-over-IP.

Introduction to the principles underlying efficient and scalable overlay and peer-to-peer networks (unstructured such as Gnutella, or structured). Implementation of a simple peer-to-peer network using the socket interface.

Reading List


Resources Course notes, web page, study guide, worksheets, handouts, lecture rooms with OHP and data projector, past examination papers.

Module Evaluation Module questionnaires, course review.
CO7214 Service-Oriented Architectures

Credits: 15  Convenor: Reiko Heckel  Semester: 2nd

Prerequisites: Desirable: UML, XML, Java
Assessment: Coursework: 40%  Two hour exam in May/June: 60%
Lectures: 24 hours
Surgeries: 8 hours
Laboratories: 8 hours
Private Study: 72.5 hours

Subject Knowledge

Aims The aim of this module is to give students knowledge and skills in the principles and functions of service-oriented systems and applications, as well as in their development based on Java, UML, and XML. This shall enable them to use such technologies in practice or to embark on a PhD in this area.

Learning Outcomes Students should be acquainted with the conceptual and technological foundations of Service-Oriented Architectures (SOA), i.e.

- the motivation, basic mechanisms, and open problems of SOA;
- service-oriented development and its relation to object-oriented and component-based development;
- the realisation of SOA based on XML and Web service technology.

Methods Class sessions together with course notes, labs, recommended textbooks, and worksheets.
Assessment Multiple choice and short answer tests, written examinations.

Skills

Aims The module shall provide the basic skills required to use SOA technologies in practice.

Learning Outcomes Students should be able to create and understand descriptions of services and systems using both high-level models and XML-based languages, including

- UML models of service-oriented applications at different levels of abstraction;
- their implementation in Java and Web services.

Methods Worksheets and practical programming experience.
Assessment Computer-based exercises, computer programmes.

Explanation of Prerequisites Basic knowledge of XML, UML, and Java will be helpful.

Course Description A Web service is an application component deployed on a Web accessible platform, provided by a service provider to be discovered and invoked over the Web by a service requestor. Service-oriented architectures, the underlying architectural style of Web services, combine ideas from component-based and distributed systems, adding the idea of services as loosely coupled components that may be discovered and linked at runtime. Applications range from enterprise application integration, via electronic commerce, to dynamic e-business scenarios.

The lecture shall give an introduction to the basic technologies that underly Web services and present a systematic, model-based development approach using the UML. This includes the specification of service interfaces by means of UML diagrams, the systematic (and partly automatic) generation of the corresponding XML-based descriptions, and the implementation of services in Java.
Detailed Syllabus  In detail, the module will cover the following topics.

- Motivation and Concepts of SOA;
- Modelling XML Languages with UML;
- Model-Based Data Integration using XSLT;
- XML-based Messaging using SOAP;
- Describing and Publishing Web Services using WSDL and UDDI;
- Processes for Web Services based on BPEL4WS;
- Web Services and Semantic Web technology;

Reading List


Resources  Course notes, web pages, study guide, worksheets, handouts, lecture rooms with data projector/OHP.

Module Evaluation  Course questionnaires, course review.
CO7215 Advanced Web Technologies

Credits: 15  Convenor: Dr. S Reiff-Marganiec  Semester: 1st

Prerequisites: none
Assessment: Coursework: 40%  Two hour exam in January: 60%
Lectures: 16 hours  Surgeries: 8 hours  Private Study: 72.5 hours  Laboratories: 16 hours

Subject Knowledge

Aims  The aim of this course is to teach the students the concepts, technologies and techniques for creating large-scale distributed software system using the Service oriented Paradigm.

Learning Outcomes  At the end of the course the student should be able to: understand the fundamental ideas and standards underlying Web Service Technology; discuss concepts at the frontier of industrial practice and emerging standards; link the concepts of services and business processes and understand the role and functionality of BPEL; discriminate between the major frameworks allowing to develop web services and to develop web services using the .Net framework and apply BPEL4WS.

Methods  Lectures, tutorials and practical sessions together with course notes, recommended reading, worksheets and some additional handouts.

Assessment  Assessed coursework; traditional written exam

Skills

Aims  To teach students problem solving skills.

Learning Outcomes  Students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen).

Methods  Class sessions together with worksheets.

Explanation of Prerequisites

Course Description  Service oriented Computing and their predominant implementation as Web Services are at the forefront of industrial practice in software engineering. There are two major technologies supporting their development: Microsoft’s .net and Java based technologies. In this course we will use the former.

One crucial aspect of SoA is the marrying of IT artefacts with business processes and objectives, so part of the course will concentrate on business processes and their relation to services.

Detailed Syllabus  Topics to be covered include fundamental ideas and standards underlying Web Service Technology, concepts at the frontier of industrial practice, emerging standards and business processes.

Reading List


The Department of Computer Science  110
Resources Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, past examination papers, past tests.

Module Evaluation Course questionnaires, course review.
CO7216 Semantic Web

Credits: 15  Convener: tba  Semester: 2nd

| Prerequisites: Essential: Background in HTML and XML |
| Assessment: Coursework: 40%  Two hour exam in May/June: 60% |
| Lectures: 24 hours |
| Surgeries: 8 hours  Class Tests: 1 hours |
| Laboratories: 8 hours  Private Study: 72.5 hours |

Subject Knowledge

Aims  The aim of this course is to teach the students the concepts, technologies and techniques underlying and making up the Semantic Web.

Learning Outcomes  At the end of the course the student should be able to: understand and discuss fundamental concepts, advantages and limits of the semantic web; understand and use ontologies in the context of Computer Science and the semantic web; use the RDF framework and associated technologies such as RDFa; understand the relationship between Semantic Web and Web 2.0.

Methods  Lectures, tutorials and practical sessions together with course notes, recommended reading, worksheets and some additional handouts.

Assessment  Assessed coursework; traditional written exam

Skills

Aims  Students who have taken this module should be able to understand the rationale behind Semantic web. They should be able to model and query domain knowledge as ontologies defined using standards such as RDF and OWL. Students should be able to apply the principles of ontological engineering to modelling exercises. Finally they should be able to understand the applications of semantic web to web services and Web 2.0.

Learning Outcomes  On successful completion of the module students should be able to:

- understand the rationale behind Semantic Web.
- model ontologies using Resource Description Framework (RDF).
- design RDF Schemas for ontologies.
- model and design ontologies using Web Ontology Language (OWL).
- query ontologies using SPARQL.
- understand and reflect on the principles of Ontology Engineering.
- make an association between Semantic web and Web 2.0.
- apply Semantic web technologies to real world applications.

Methods  Class sessions together with worksheets and lab assignments.

Explanation of Prerequisites  Students should have a basic understanding and knowledge of HTML and XML and related technologies.
Course Description  The Web, as it exists today, primarily supports human understanding and the interpretation of the vast information space it encompasses. However the Web was originally designed with a goal to support not only human-human communication but also as one that would enable automated machine processing of data with minimal human intervention. The Semantic Web is Tim Berners-Lee’s vision of a machine understandable and unambiguously computer interpretable Web. The rationale behind such a system is that most of the data currently posted on the web is buried in HTML files suitable for human reading and not for computers to manipulate meaningfully. The semantic Web, an extension of the current web, can be thought of as a globally linked database where information is given well-defined meaning using metadata for better enabling computers and humans to work in close co-operation. The realisation of a Semantic Web will thus make machine reasoning more ubiquitous and devastatingly powerful, creating an environment where intelligent software agents can roam, carrying out sophisticated tasks for their users.

This course is about investigating the next generation of the Web whose key distinguishing characteristics will be the support for and use of semantics in new, more effective, more intelligent, ways of managing information and supporting applications.

Detailed Syllabus  Topics to be covered include:

- Introduction to the Semantic Web
- Introduction to Ontologies
- Ontology Languages for the Semantic Web
  - Resource Description Framework (RDF)
  - Lightweight ontologies: RDF Schema
  - Web Ontology Language (OWL)
  - A query language for RDF: SPARQL
- Ontology Engineering
- Semantic web and Web 2.0
- Applications of Semantic Web

Reading List

Resources  Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs.

Module Evaluation  Course questionnaires, course review.
Subject Knowledge

Aims The aim of this module is to give students knowledge and skills in the principles and functions of domain-specific languages (DSLs) and their applications, as well as in their development based on OMG standards (MOF, OCL, QVT) and the Eclipse Modeling Framework (EMF). This shall enable them to use such technologies in practice or to embark on a PhD in this area.

Learning Outcomes At the end of this module, successful students will be acquainted with the conceptual and technological foundations of DSLs, i.e., the motivation, basic mechanisms, and open problems of DSLs; model-driven development and its relation to object-oriented development; the realisation of modeling environments based on OMG standards and Eclipse technology.

Methods Class sessions together with course notes, recommended textbooks, and worksheets.

Assessment Multiple choice and short answer tests, written examinations.

Skills

Aims The module shall provide the basic skills required to use technology for defining DSLs and for using them in practice.

Learning Outcomes At the end of this module, successful students will be able to design and develop DSLs using both OMG standards and model-based technology, including: MOF metamodels, describing the abstract syntax of a DSL, their implementation as EMF metamodels, and their realization as Java-based modeling environments; OCL constraints, adding semantics to a MOF metamodel; the definition of the concrete syntax of a DSL, either graphical or textual; automated generative techniques: model transformations and code generation techniques.

Methods Worksheets and practical programming experience.

Assessment Computer-based exercises, computer programmes.

Explanation of Prerequisites Basic knowledge of UML Class Diagrams, Eclipse and Java will be helpful.

Course Description According to the Standish Group, businesses in the United States spend about $250 billion annually on software development. Only 16% of these projects are completed on schedule and within budget. Another 31% are canceled, primarily because of quality problems, creating losses of about $81 billion annually. Another 53% cost more than planned, exceeding their budgets by an average of 189%, creating losses of about $59 billion annually. Projects that reach completion deliver an average of only 42% of the originally planned features. These losses are primarily due to the complexity of the systems that are developed and to changes in user requirements.

Model-Driven Development (MDD) is intended for increasing the quality and the productivity of a software development process. This is achieved by raising the level of abstraction from code to models and by automating tasks by means of generative techniques, which enhance the reuse of knowledge, patterns and code. A modeling language that permits defining such models in a given domain constitutes
a decisive keystone in a MDD process.

The module shall give an introduction to the basic technologies that underly the definition of DSLs and their use in a MDD process. This includes the definition of the syntax of a DSL at different levels: at a conceptual level by means of MOF metamodels, and at a concrete level by means of either textual-based techniques or graphical techniques; their definition in the Eclipse Modeling Framework; the use of OCL constraints to add semantics to the syntax of a DSL; and the use of automated techniques for manipulating models: model transformations by using QVT and code generation techniques.

**Detailed Syllabus**  In detail, the module will cover the following topics.

- Motivation and concepts of DSLs in MDD Processes;
- Definition of the conceptual layer of DSLs by using MOF and OCL;
- Definition of the concrete syntax of DSLs by using textual and graphical syntax;
- Implementation of DSLs by using the Eclipse Modeling Framework and the Graphical Modeling Framework;
- Automated model transformations by using QVT, OCL and Graph Transformations;
- Automated code generation.

**Reading List**


**Resources**  Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs.

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

Aims

The aim of this course is to explore information systems concepts, technologies and techniques and the role of IT departments in the context of the structure, objectives and business processes of financial services organizations.

Learning Outcomes

At the end of the course the student should be able to: understand some of the fundamental concepts, terminology, structure and processes of the financial services domain; be aware of the key organizational units and their respective functions in financial services organizations; differentiate categories of financial systems and applications and be able to comprehend their characteristics and their relationships from different perspectives, namely business, functional, architectural and technological; understand the different roles and functions of IT professionals within financial services.

Methods

Lectures, tutorials and practical sessions together with course notes, recommended reading, worksheets and some additional handouts.

Assessment

Assessed coursework; traditional written exam

Skills

Aims

To teach students problem solving skills.

Learning Outcomes

Students will be able to apply logical thinking in order to solve abstract and concrete problems and make decisions based on available information.

Methods

Class sessions together with worksheets.

Explanation of Prerequisites

Course Description

Financial services companies are one of the two largest IT consumers worldwide and the IT systems in such companies are critical not only for the continuous operation of the business, but also as transformation and differentiation enablers. In order to be able to manage, develop and operate IT systems effectively in such organizations, IT people not only need to have good technical knowledge and skills but also a sound understanding of the financial services domain and of the relationships between the operating environment, organizational structures, business processes and IT systems. Focusing on the latter, this course will: (i) provide an introduction to the key financial services domain concepts, organizational units, functions and IT systems, (ii) discuss their inter-relationships and, (iii) provide an overview of the different roles and responsibilities of IT professionals in such organizations.

Detailed Syllabus

Topics to be covered include: The financial services market, types of financial services organizations, key concepts and terminology (e.g. financial instruments, loans, deposits, risks), key organizational units, functions and processes (e.g. loan origination, securities trading, payments), types of information systems in financial services and their functional and architectural perspectives, banking applications landscape and analysis of key applications, fundamentals of business intelligence systems, application integration, introduction to business process management systems and business
rules management systems, IT roles and functions, current industry trends and issues.

**Reading List**


**Resources**  Course notes, web page, study guide, worksheets, handouts, lecture rooms with two OHPs, sample examination papers, sample tests.

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