University of Leicester

Department of Computer Science

Module Forms

31st January 2017

Contents

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 aquire.

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.

CO1001 Logic and Problem Solving

Credits: *20* **Convenor:** *Dr A. Kurz* **Semester:** *2nd*

Prerequisites:	*Essential: basic discrete mathematics*				
Assessment:	*Coursework: 100%*				
Lectures:	*30*	*hours*	**Problem Classes:**	*6*	*hours*
Surgeries:	*6*	*hours*	**Private Study:**	*100*	*hours*
Laboratories:	*8*	*hours*			

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.

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.

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

[B] I. Bratko, *Prolog, Programming for Artificial Intelligence, 3rd edition; ISBN: 0201403757,* Addison-Wesley, 2001.

[C] L. Sterling and E. Shapiro, *The Art of Prolog, 2nd edition.; ISBN: 0262691639,* MIT Press, 1994.

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. R. van Stee* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures:	*29*	*hours*	**Class Tests:**	*1*	*hours*
Surgeries:	*10*	*hours*	**Private Study:**	*90*	*hours*
Laboratories:	*20*	*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 1.8. 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 modules 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

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. Structured data-types: arrays.
7. Strings and string handling, formatting.
8. Overview of design and development concepts; requirements analysis; basic notions of specification.
9. Fundamentals of object-oriented design.
10. Software problems : errors, faults, and failures.
11. Testing.

Reading List

[A] C. Thomas Wu, *An Introduction to Object-Oriented Programming with Java, 5th (international) edition; ISBN: 978-0071283687,* McGraw-Hill.

[C] R. Pressman and D.Ince, *Software Engineering — a Practitioner's Approach, European 6th edition; ISBN: 0071238409,* McGraw-Hill.

[C] I. Sommerville, *Software Engineering, 7th edition; ISBN: 0321210263,* Addison-Wesley.

Resources Departmental web page, text book web site, automated feedback and assessment tool (JACK), 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:** *Dr T Ridge* **Semester:** *2nd*

Prerequisites:	*Essential: CO1003*				
Assessment:	*Coursework: 100%*				
Lectures:	*30*	*hours*	**Class Tests:**	*3*	*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, JUnit, Eclipse.

Assessment Class tests, formative lab work.

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

[A] C. Thomas Wu, *A Comprehensive Introduction to Object-Oriented Programming with Java,* McGraw-Hill.

[A] David A Watt, Deryck F Brown, *An Introduction to Abstract Data Types, Data Structures, and Algorithms,* John Wiley and Sons.

[A] Kathy Walrath, Mary Campione, Alison Huml, *The Java Tutorial: A Short Course on the Basics (Java Series) (Paperback),* Prentice Hall PTR.

Also available online at:

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

[B] Michael T. Goodrich and Roberto Tamassia , *Data Structures and Algorithms in Java (4th edition),* John Wiley and Sons.

[B] Isaac Rabinovitch, Sharon Zakhour, Scott Hommel, Jacob Royal and Thomas Risser, *The Java Tutorial, Fourth Edition,* Prentice Hall, Fourth edition, 672 pages.

[B] W. J. Collins, *Data Structures and the Java Collections Framework; ISBN: 0071114076,* McGraw-Hill.

[C] Robert Sedgewick, *Algorithms in Java, Parts 1-4, 3rd Edition,* Addison-Wesley.

[C] U. Manber, *Introduction to Algorithms: a Creative Approach,* Addison-Wesley.

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.

CO1008 Requirements Engineering and Professional Practice

Credits: *10* **Convenor:** *Dr N. Verdezoto, Dr R. Craggs* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures:	*11*	*hours*	**Problem Classes:**	*0*	*hours*
Surgeries:	*0*	*hours*	**Class Tests:**	*2*	*hours*
Laboratories:	*0*	*hours*	**Private Study:**	*72*	*hours*

Subject Knowledge

Aims To help student understand the role of Requirements and Requirements Engineering within Software projects. To give students the skills to use requirement modelling approaches to describe what a software tool does (or should do).

Students will be able to explain how issues that are faced when working professionally (e.g. Laws, Codes of Conduct) relate to requirements.

Learning Outcomes At the end of the module a student should be able to:

- list the reasons why requirements are necessary for software projects and the problems that can occur when requirements are missing or poorly documented.
- classify different types of requirements (e.g. functional, non-functional, technical constraints)
- choose the best type of modelling technique to describe an aspect of a software system.
- Identify whether laws relating to software apply in relevant situations.
- evaluate the value of membership of professional bodies in regards to a career and list the responsibilities that it will entail.

Methods Lectures, Group-work including supervised project work. Course notes and recommended reading, Office Hours.

Assessment Formative exercises, marked project work and multiple choice tests.

Skills

Aims Gain experience of applying techniques and processes for requirements engineering on case studies within project work. Also experience of some of the behaviours necessary within a professional software engineering role.

Learning Outcomes Students will be able to:

- create static and dynamic models to describe a software system.
- write high quality requirements to describe a software project
- create sketches and prototypes to materialise and test requirements
- apply quantitative and qualitative elicitation techniques to gather software requirements
- work collaboratively on a group project

Methods Group-work, exercises, individual reading, and discussions.

Assessment Formative exercises, marked project work and multiple choice tests.

Explanation of Prerequisites None

Course Description Most software written within the IT industry is created to solve problems within some organsiation or group of users, or to provide new possibilities for users. The success of IT projects relies heavily of understanding the domain in which software is used, and what the software must do (and not do) to provide users with what they need. The process of gaining this understanding and documenting it in a way that helps everyone to understand it is "Requirements Engineering".

In this module we'll describe the role that requirements engineering plays in all projects and the techniques and outcomes that are applied to ensure success.

Well managed requirements engineering is often a hallmark of a professionally run project. We'll also cover other aspects of professionalism in software development, and how professional bodies and laws play an important role.

Detailed Syllabus

- Written Requirements
- Quantitative Methods
- Qualitative Methods
- Laws
- Static Modelling
- Dynamic Modelling
- Professionalism
- Testing
- Sketching and Low Fidelity Prototyping

This module will also include an introduction to the British Computer Society.

Reading List

[A] Various, *selected articles on requirements modelling, see Blackboard for topic-specific reading,*

[A] BCS, *BCS Code of Conduct,*

[A] BCS, *BCS Code of Good Practice,*

Resources Course notes, study guide, case study, hand-outs

Module Evaluation Course questionnaires, course review.

CO1012 Discrete Structures

Credits: *10* **Convenor:** *Prof R M Thomas* **Semester:** *1st*

Prerequisites: *Essential: GCSE Mathematics (or equivalent)*

Assessment: *Coursework: 100%*

Lectures:	*18*	*hours*	**Problem Classes:**	*6*	*hours*
Surgeries:	*14*	*hours*	**Class Tests:**	*6*	*hours*
			Private Study:	*31*	*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 will 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 Five class tests.

Skills

Aims To teach students scientific writing and problem solving skills.

Learning Outcomes Students will 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 Five class tests.

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

1. Elementary logic. Notation for logical connectives. Translating statements into formal notation. Methods of proof.
2. Sets. Basic operations on sets (union, intersection, difference, powerset). Properties of these operations. Cardinality of a set.
3. Cartesian products. Relations between sets. Examples of relations. Types of relation. Composition of relations.
4. Partial and total functions. Composition of functions. Recursive definitions of functions from the set of natural numbers. Graphs. Trees. Subtrees. Simple searching problems on trees.
5. Matrices. Adjacency matrix of a graph. Operations on matrices. Matrix representation of a relation. Composition of relations via matrices.
6. Exponentials and logarithms. Factorials. Basic concepts of algorithm analysis applied to simple algorithms. Permutations and combinations.
7. Elementary probability. Big O notation: concept and basic properties.

Reading List

[B] Kenneth H. Rosen, *Discrete Mathematics and Its Applications,*; McGraw-Hill.

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

Module Evaluation Course questionnaires, course review.

CO1016 Computer Systems

Credits: *20* **Convenor:** *Dr. R. Crole* **Semester:** *1st*

Prerequisites:	*none*				
Assessment:	*Coursework: 15%*		*Three hour exam in January: 85%*		
Lectures:	*30*	*hours*	**Problem Classes:**	*10*	*hours*
Surgeries:	*10*	*hours*	**Class Tests:**	*2*	*hours*
			Private Study:	*98*	*hours*

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 digital 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 (single-core) 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 worksheet and 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 Formative: Marked problem-based worksheets, class tests. Summative: traditional written problem-based 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 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 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 (we look at a single core in this introductory module) 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–single core–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 single core processor is the culmination of the module, and students will then be equipped to read further material about full-scale modern multi-core processors.

Detailed Syllabus *Examples throughout the course will be based on the MIPS Instruction Set Architecture. See the module web page for industry details.*

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 hierachy and simple details of cache memory.

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

Basic digital electronics: Elementary logic and truthtables. Gates for implementing (Boolean models of) simple logical propositions, and the composition of gates to make more complex circuits. Multiplexors, decoders, and related circuits. Clocks. Implementation of atomic Arithmetic Logic Units (ALUs) via digital circuits. Construction of a 3-bit ALU (and 32-bit ALU, non-examinable). Simple memory circuits, including CD latch. Basics of register files.

The MIPS instruction set and simple MIPS programs. A subset of the MIPS language treated in detail at the assembly and machine levels. [Note: slt instruction is non-examinable.] Semantics, machine fields, branch calculations, and assembly/machine translations.

Construction of a simple (core) datapath via composition of atomic ALUs, register file(s), memory(ies) and other basic components. Description of MIPS control program. The interaction of the datapath and control to make a processor. [Computing performance.]

Reading List

[B] Hennessy and Patterson, *Computer Organization and Design;*
ISBN: 9780123747501 (4th) 9780124077263 (5th), Morgan Kaufmann, 2013, fifth edition.
Note: fourth edition is okay..

[C] Tanenbaum, *Structured Computer Organization;*
ISBN-10: 0132916525. ISBN-13: 9780132916523, Prentice Hall (Pearson), 2013 (sixth edition).

[C] Stallings, *Computer Organization and Architecture;*
ISBN-10: 0132916525. ISBN-13: 9780132916523, Prentice Hall (Pearson), 2013 (eighth edition), 0130351199.

[C] Burrell, *Fundamentals of Computer Architecture,* Palgrave Macmillan.

[C] Carpinelli, *Computer Systems: Organization and Architecture,* Pearson (Addison Wesley).

[C] Clements, *The Principles of Computer Hardware,* Oxford University Press.

[C] Englander, *The Architecture of Computer Hardware and Systems Software,* Wiley.

[C] Hayes, *Computer Architecture and Organization,* McGraw Hill.

[C] Hamacher, Vranesic and Zaky, *Computer Organization, 5th edition,* McGraw Hill.

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. K. Mualla* **Semester:** *2nd*

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

Subject Knowledge

Aims This module consists of three key parts. It first teaches you how the Protocol Stack operates over the network. Secondly, you will learn how to build, design, and program web pages using HTML, CSS, and JavaScript. The third part will teach you how to design, implement, maintain and query relational databases using MySQL.

Learning Outcomes Students should be able to demonstrate a knowledge of the basic techniques involved in the Protocol Stack, web development, data organization, 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 HTML, CSS and JavaScript. They will be able to describe and overview the key database functionalities and queries, and they will be able to illustrate software bugs in relation to web development and design.

Methods Lectures, surgeries, laboratory practical sessions together with course notes (available on Blackboard), in addition to 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 and development process; solve abstract and concrete problems (both routine seen, and simple unseen); and locate, access, organize 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.

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

The relationship between the web, network protocols, and database systems currently forms the hierarchy access model for almost every modern application. This module examines the general concepts of the Network Protocol Stack, Web Development using HTML, CSS and JavaScript, and Database Management Systems using entity relational diagrams, normalization techniques, and MySQL as a key query language.

Detailed Syllabus HTML (static webpages): Internet Technologies; The Internet Protocol Stack; HTML, HTML5; CSS, JavaScript.

Reading List

[A] Bates, *Web Programming (3rd edition); ISBN: 0470017759,* Wiley, 2006.

[A] T. Connolly and C. Begg, *Database Systems (Fourth edition); ISBN: 0321210255,* Addison-Wesley, 2005.

[A] *MySQL Reference Manual; ISBN: 0596002653*, `http://www.mysql.com/documentation/`.

[B] Deitel, Deitel and Nieto, *Internet and World Wide Web: How to Program (2nd edition); ISBN: 0131218557,* Prentice Hall, 2002.

[B] Hege Refsnes, Stle Refsnes, Kai Jim Refsnes, Jan Egil Refsnes and C. Michael Woodward, *HTML and CSS: Learn HTML and CSS with w3schools,* Wiley Publishing, Inc,2010.

Resources Course notes, text books in library, study guide, worksheets, module web pages on Blackboard, lecture rooms with fixed computer, data projector, laboratories with PCs and demonstrators, Workbench 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: 50%*	*Three Hour Examination in May/June: 50%*	
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 idieally 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. F.-J. 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 web-sites.

Detailed Syllabus

Basic architecture of the Web: servers, HTTP, TCP/IP, etc. Security issues: HTTPS, firewalls, viruses. Internet search (relevancy), meta-data, digital object identifiers (DOI). Creating and maintaining a web-site: HTML, CSS, links (rel vs. absolute), W3C standards, accessibility, browser compatibility. Graphics and graphics tools.

Reading List

[C] Steven M. Schafer, *HTML, XHTML, and CSS bible,* Indianapolis, IN: Wiley, c2010.

[C] David Karlins, *Dreamweaver CS5.5 mobile and Web development with HTML5, CSS3, and jQuery : harness the cutting edge features of Dreamweaver for mobile and web development,* Birmingham, England : Packt Pub., 2011.

[C] Joseph Lowery, *Adobe Dreamweaver CS5 bible,* Indianapolis, Ind.: Wiley Pub., Inc., 2010.

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 A Reiff-Marganiec* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures: *3 hours*
Laboratories: *18 hours*

Private Study: *54 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 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 7 operating system.

Learning Outcomes The students 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.

Methods Coursework with a variety of laboratory exercises. The course emphasises a 'hands-on' approach. Experience with computers is not a prerequisite for this module.

Assessment Marked coursework.

Course Description The module comprises a series of nine 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.

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, coursework, electronic coursework submission facility, GTA and postgraduate support, computer labs.

Module Evaluation Course questionnaires, course review.

CO1961 Information Management

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

Prerequisites: *Essential: GCSE Mathematics (or equivalent)*

Assessment: *Coursework: 100%*

Lectures:	*16*	*hours*	**Class Tests:**	*4*	*hours*
Surgeries:	*8*	*hours*	**Private Study:**	*47*	*hours*

Subject Knowledge

Aims This module aims to teach the use of statistical techniques.

Learning Outcomes Students should be able to: Represent simple data in an appropriate manner.

Methods Class sessions with course notes and worksheets. Recommended textbook for extra information and supplementary reading.

Assessment Assignments and marked class tests.

Skills

Aims To equip students with skills in the use of statistical techniques.

Learning Outcomes Students should be able to: Use statistical techniques to calculate information from data and to make projections about the future behaviour of the data; Calculate probabilities from observed data and use sampling techniques to estimate the mean of the data with a certain degree of confidence.

Methods Class sessions with course notes and worksheets. Recommended textbook for extra information and supplementary reading.

Assessment Assignments and marked class tests.

Course Description

The module teaches the use of statistical methods.

CO2001 User Interfaces and HCI

Credits: *10* **Convenor:** *Dr. S.Kerrigan* **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 Graphical User Interfaces and their implementation in Java. Event-driven programming in Java. HCI: introduction to applets. Extended exception handling. Introduction to threading. File input/output.

Reading List

[A] C. Thomas Wu, *An Introduction to Object-Oriented Programming with Java, 5th (international) edition; ISBN: 978-0071283687,* McGraw-Hill.

[B] Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour, *The JFC Swing Tutorial: A Guide to Constructing GUIs (2nd Edition) (Java Series) (Paperback),* Prentice Hall PTR.

Also available online at:

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

[C] Paul Fische, *An Introduction to Graphical User Interfaces with Java Swing,* Addison Wesley.

[C] Mauro Marinilli, *Professional Java User Interfaces,* Wiley.

[C] Scott Oaks, Henry Wong, *Java Threads (Paperback),* Oreilly.

[C] Kathy Walrath, Mary Campione, Alison Huml, *The Java Tutorial: A Short Course on the Basics (Java Series) (Paperback),* Prentice Hall PTR.

[C] Marc Loy, Robert Eckstein, Dave Wood, James Elliott, Brian Cole, *Java Swing, Second Edition,* Oreilly.

[C] Isaac Rabinovitch, Sharon Zakhour, Scott Hommel, Jacob Royal and Thomas Risser, *The Java Tutorial, Fourth Edition,* Prentice Hall, Fourth edition, 672 pages.

[C] Chet Hasse, Romain Guy , *Filthy Rich Clients,* Addison Wesley.

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, CO1012*	*Desirable: CO1019*	
Assessment:	*Coursework: 40%*	*Two hour exam in May/June: 60%*	
Lectures:	*15 hours*	**Private Study:**	*55 hours*
Surgeries:	*5 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 some of the key 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 business/financial domain; understand the categories and functions of business 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 Assessed coursework; traditional written exam

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

Course Description Business organizations 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 has intensified, 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, Open-Source solutions and the Big Data 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 some 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 key software applications to support these domains (ii) explore some basic information systems concepts and (iii) provide an overview of the role of IT departments in such large organizations.

Detailed Syllabus Topics to be covered include: Basic business and financial concepts (definitions and terminology), financial markets and instruments, capital budgeting elements, types of information

systems in such organizations and their functional and architectural perspectives, business intelligence systems, IT roles and functions.

Reading List

[B] , *Business Knowledge for IT in Retail Banking: The Complete Handbook for IT Professionals,* Essvale Corporation Limited; ISBN: 0955412420, 2007 .

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:** *Dr A. Boronat* **Semester:** *1st*

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

Subject Knowledge

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

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

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

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

Skills

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

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

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

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

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

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

Detailed Syllabus

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

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

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

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

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

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

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

Reading List

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

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

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

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

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

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

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

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

Module Evaluation Course questionnaires, course review.

CO2008 Functional Programming

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

Prerequisites:	*Essential: CO1001, CO1003, CO1005, CO1012*	*Desirable:*	
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 a1 $\rightarrow$ a2 $\rightarrow$ a3 ... $\rightarrow$ an. Booleans and guards; correspondence of guards with if-then-else expressions. Pairs and n-tuples; `fst` and `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

[A] G. Hutton, *Programming in Haskell ISBN: 0-521-69269-5,* Cambridge University Press. 2007.

[B] R. Bird and P. Wadler, *Introduction to Functional Programming; ISBN: 0134843460,* Prentice Hall 1988.

[B] S. Thompson, *Haskell: The Craft of Functional Programming, 2nd Edition; ISBN: 0201342758,* Addison-Wesley. 1999.

[C] L. Paulson, *ML for the Working Programmer, 2nd Edition; ISBN: 052156543X,* CUP 1997.

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.

CO2011 Automata, Languages and Computation

Credits: *20* **Convenor:** *Prof R M Thomas* **Semester:** *1st*

Prerequisites:	*Essential: CO1012 (or equivalent)*	*Desirable: CO1003 (or similar)*	
Assessment:	*Coursework: 100%*		
Lectures:	*36 hours*	**Problem Classes:**	*5 hours*
Surgeries:	*4 hours*	**Class Tests:**	*6 hours*
		Private Study:	*99 hours*

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 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 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, graphs 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

Revision of mathematical pre-requisites (sets, relations, graphs and functions). Strings. Formal languages. Operations on languages. Concatenation of strings. Kleene star.

Finite automata. Language acceptors. Regular languages. Equivalence. Complete automata. The concepts of determinism and non-determinism. The pumping lemma for regular languages. Examples of non-regular languages. Regular grammars. Equivalence of regular grammars and finite automata. Closure properties of regular languages. Empty moves. Regular expressions. Equivalence of regular expressions and finite automata.

Stacks. Pushdown automata and context-free grammars. Emptying the stack. Parse trees. Leftmost and rightmost derivations. Equivalence of pushdown automata and context-free grammars. Ambiguous grammars. Inherent ambiguity. Closure properties of context-free languages. Are programming languages context-free? Deterministic context-free languages. Parsing. LL-parsers.

Turing machines. Extensions of Turing machines. Non-determinism. Church-Turing Thesis. Decision-making Turing machines. Recursive languages. Universal Turing machines. The halting problem. Further examples of unsolvable problems.

Complexity. Space and time complexity and the relationship betwen them. Decision-making versus acceptance. Polynomial transformations. Determinism versus non-determinism. P and NP. NP-completeness. Examples of NP-complete problems.

Reading List

[C] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation, second edition; ISBN: 0132727412,* Prentice Hall, 1998.

[C] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation; ISBN: 0321210298,* Addison-Wesley, 2007.

[C] J. G. Brookshear, *Formal Languages, Automata and Complexity,* Benjamin Cummings, 1989 (out of print, but copies available in the Library).

[C] D. Kelley, *Automata and Formal Languages - an Introduction,* Prentice Hall, 1995 (out of print, but copies available in the Library).

[C] D. Wood, *Theory of Computation,* Wiley, 1987 (out of print, but copies available in the Library).

[C] D. I. A. Cohen, *Introduction to Computer Theory; ISBN: 0471137723,* Wiley, 1996.

[C] J. Martin, *Introduction to Languages and the Theory of Computation; ISBN: 0071198547,* McGraw-Hill, 2008.

[C] P. Linz, *An introduction to Formal Languages and Automata; ISBN: 0763714224,* Jones and Bartlett, 2001.

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

Module Evaluation Course questionnaires, course review.

CO2012 Software Project Management and Professionalism

Credits: *10* **Convenor:** *Dr R. Craggs* **Semester:** *1st*

Prerequisites:	*Essential: CO1003, CO1005, CO1007, CO1019*		*Desirable: CO1001, CO1012*		
Assessment:	*Coursework: 100%*				
Lectures:	*9*	*hours*	**Problem Classes:**	*0*	*hours*
Surgeries:	*0*	*hours*	**Class Tests:**	*1*	*hours*
Laboratories:	*13*	*hours*	**Private Study:**	*52*	*hours*

Subject Knowledge

Aims This module will teach you techniques and technologies to run and collaborate on a software project.

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

- be able to compare traditional and agile approaches to project planning and monitoring.
- describe the benefits of continuous integration and test automation.
- behave professionally on a software project.
- demonstrate awareness of ethical and legal issues, like the Data Protection Act, likely to affect every professional in the software industry.
- formulate technical problems and their solution in a methodical way;
- research an issue and present their findings in writing in a balanced manner.

Methods Curated pre-lecture videos, lectures, classroom activities, worksheets, supervised labs for mini project group work.

Assessment Marked coursework, including written essay, mini-projects, monitored use of version control and class test.

Skills

Aims To collaborate in a project in a controlled manner working as a team.

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

- be able to plan a software project using a traditional approach and an agile approach.
- be able to use git version control;
- be able to apply continuous integration to projects and work producively on projects that use it;
- be able to apply quality control measures to a software project;
- 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, labs and with worksheets.

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

Explanation of Prerequisites A basic knowledge of software development is required. For example the software is created by writing source code which is compiled into software.

Course Description This module teaches the techniques required for working in a team on a software project. It provides practice in applying these within lab exercises and a mini-project done in a group.

Detailed Syllabus

- Project Planning and Gantt Charts
- Team collaboration on software projects;
- Version control using Git
- Continuous integration and automated quality tests
- Agile project planning and monitoring
- The scrum software development framework

Reading List

Pluralsight
"Reading" is provided through curated listed of video content from the Pluralsight library.

Resources Video Content, Slides, study guide, worksheets, lecture rooms with projector.

Module Evaluation Course questionnaires, course review.

CO2015 Software Engineering Project

Credits: *20* **Convenor:** *Dr Y. Hong, Dr R. Craggs* **Semester:** *2nd*

Prerequisites:	*Essential: CO2006, CO2012, Desirable: CO1012 CO1003, CO1005, CO1019*
Assessment:	*Coursework: 100%*
Lectures:	*approx. 5 hours*
Laboratories:	*1 hours*
Private Study:	*approx. 144.0 hours*

Subject Knowledge

Aims According to a 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: provide opportunities for students to develop skills in the analysis, design, specification, implementation, testing and documentation of computer software systems; develop skills that will enhance employment prospects, especially in the IT industry or other numerate disciplines.

Learning Outcomes At the end of this course, successful students will be able to demonstrate participation, according to a role description, in all stages of the development lifecycle for a medium sizes software system. Moreover, successful students will be able to: recognise important dependencies between the activities mentioned above; critically assess the software life cycle in terms of general quality attributes and viable trade-offs presented within the given problem; employ a configuration management system effectively; schedule and manage a software development project; identify and address risks in the software life cycle; apply appropriate practices within a professional, legal and ethical framework.

Methods Lectures, lab session, meetings with supervisor, self-taught video content.

Assessment Project deliverables, presentation, and demonstration of the software system.

Skills

Aims To develop written and oral communication skills; to provide students with experience of team-based project work; to develop scientific problem solving abilities, along with an appreciation for mathematical and scientific methods.

Learning Outcomes At the end of this course, successful students will be able to: work as a member of a development team, recognising the different roles within a team; conduct significant background research; retrieve information from different sources and manage it effectively; work with uncertain, limited and possibly contradictory information; solve complex problems with other members of the team; communicate in electronic as well as written and oral form; apply management techniques to allocate resources to projects; undertake a risk assessment for a medium-scale team-based software project, especially for risks arising from the use of the resulting software, and to specify appropriate security requirements; formulate and apply suitable tests to assess the security of their software in relation to its requirements; manage their own learning and development including time management and organisational skills as the foundation of on-going professional development.

Methods Lectures, lab session, meetings with supervisor.

Assessment Project deliverables, presentation, 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 apply software engineering methods that have been studied in the degree programme. Students will work in groups of about seven. Groups will follow a light-weight form of an Agile Development Process tailored to the needs of this project.

Detailed Syllabus The practice of software engineering methods; software life-cycle management; risk assessment; definition and prioritization of project goals; requirements elicitation and analysis; system design, construction and testing; quality assurance; requirements verification and validation; configuration management.

Reading List

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

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

[C] G. Booch, *Object-Oriented Analysis and Design with Applications, 3rd edition,* Addison-Wesley, 2007.

[C] T. Gilb, *Principles of Software Engineering Management,* Addison-Wesley, 1988.

[C] R. Pressman, *Software Engineering — A Practitioner's Approach, European Adaptation 7th edition,* McGraw Hill, 2010.

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

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

Resources Various course notes, web page, books, videos, study guide.

Module Evaluation Course questionnaires, course review.

CO2016 Multimedia and Computer Graphics

Credits: *10* **Convenor:** *Dr. R. Crole* **Semester:** *2nd*

Prerequisites:	*Essential: CO1003, CO1005*		
Assessment:	*Coursework: 40%*	*Two hour exam in May/June: 60%*	
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: explain, discuss and solve simple problems in the basic representation and handling of multimedia data (images, audio and animation), and 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: write programs involving different multimedia formats; create simple 2D animations; write Java 3D components and reason about their behavior; create dynamic 3D environments.

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 3 dimensional space and its coordinate geometry.

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 this level will allow us to analyse and manipulate images. On the practical side we will program effects in Java. For bringing images to life, e.g. for animation, we will use the 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 digitize sound and how to deal with sound in the digitalized format (e.g. writing 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. Basic SVG concepts
4. Audio data handling
5. Scene graphs in Java3D
6. Rotation and movement of 3D objects
7. Textures, lighting in Java3D
8. Behaviors of 3D objects

Reading List

[B] Frank Klawonn, *Introduction to Computer Graphics; ISBN 978-1-84628-848-7,* Springer Verlag, 2008.

[B] Ze-Nian Li and Mark S. Drew, *Fundamentals of Multimedia; ISBN: 0130618721,* Pearson Prentice Hall, 2004.

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.

CO2017 Operating Systems, Networks and Distributed Systems

Credits: *20* **Convenor:** *Dr. G. Laycock* **Semester:** *2nd*

Prerequisites:	*Essential: CO1003*	*Desirable: CO1005,CO1016,CO1018*	
Assessment:	*Coursework: 40%*	*Three hour exam in May/June: 60%*	
Lectures:	*30 hours*		
Surgeries:	*10 hours*	**Private Study:**	*90 hours*
Laboratories:	*20 hours*		

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, memory and file-system management; 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 courseworks, 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 (and subsequent versions), 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

[A] Andrew S. Tanenbaum, *Modern Operating Systems,* Prentice Hall, 2001. ISBN 0130313580.

[A] Andrew S. Tanenbaum, *Computer Networks,* Prentice Hall, 2003. ISBN 0130661023.

[A] Herbert Schildt, *Java2: the complete reference,* McGraw-Hill Osborne Media, 2006. ISBN-10: 0072263857.

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: 40%*	*Three hour exam in May/June: 60%*	
Lectures:	*30 hours*	**Problem Classes:**	*9 hours*
Surgeries:	*10 hours*	**Class Tests:**	*1 hours*
		Private Study:	*100 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 analysing 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

[B] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms, 3rd edition; ISBN: 978-0-262-53305-8,* MIT Press, 2009.

[B] S. Dasgupta, C. H. Papadimitriou and U. Vazirani, *Algorithms,* McGraw-Hill, 2007.

[B] J. Kleinberg and E. Tardos, *Algorithm Design,* Addison-Wesley, 2006.

[B] S. Skiena, *The Algorithm Design Manual, 2nd edition,* Springer, 2008.

Resources Course notes, web page, study guide, worksheets, past examination papers.

Module Evaluation Course questionnaires, course review.

CO3007 Communication and Concurrency

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

Prerequisites:	*Essential: CO1003, CO1001, CO1012*	*Desirable: CO2011, CO1005*	
Assessment:	*Coursework: 40%*	*Three hour exam in January: 60%*	
Lectures:	*38 hours*	**Class Tests:**	*2 hours*
Surgeries:	*10 hours*	**Private Study:**	*100 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 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, including bisimulation games.

Methods Class 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 sessions, course notes and text books, class 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

[A] R. Milner, *Concurrency and Communication; ISBN: 0131150073,* Prentice-Hall 1989.

[B] C.A.R. Hoare, *Communicating Sequential Processes; ISBN: 0131532898,* Prentice-Hall 1985.

[B] J.C. Baeten and W.P. Weijland, *Process Algebra; ISBN: 0521400430,* Cambridge University Press 1990.

[B] C. Fencott, *Formal Methods for Concurrency,* Thomson Computer Press 1996.

[B] A.W. Roscoe, *The Theory and Practice of Concurrency; ISBN: 0136744095,* Prentice-Hall 1997.

[B] S. Schneider, *Concurrent and Real-time Systems; ISBN: 0471623733,* Wiley 2000.

[B] W. Fokkink, *Introduction of Process Algebra; ISBN: 354066579X,* Springer 2000.

[B] C. Stirling, *Modal and Temporal Properties of Processes; ISBN: 0387987177,* Springer 2001.

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

Module Evaluation Course questionnaires, course review.

CO3014 Computer Science Semester Project

Credits: *20* **Convenor:** *Dr. S. Kerrigan* **Semester:** *1st or 2nd*

Prerequisites: *Essential: 40 credits of second year Computer Science modules*

Assessment: *Written reports: 100%* *Examination: 0%*

Surgeries: *5 hours* **Private Study:** *145 hours*

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. S. 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*	**Private Study:**	*283 hours*
Laboratories:	*2 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. S. Kerrigan* **Semester:** *1 + 2*

Prerequisites:	*Essential: 240 credits of Computing 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 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:** *Yi Hong* **Semester:** *2nd*

Prerequisites:	*Essential: CO1003, CO1005, CO2017*	*Desirable: Notions of systems design as taught in CO2015*	
Assessment:	*Coursework: 40%*	*Three hour exam in May/June: 60%*	
Lectures:	*17 hours*	**Class Tests:**	*1 hours*
Surgeries:	*7 hours*	**Private Study:**	*111 hours*
Laboratories:	*14 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; Enterprise Java Beans; Distributed coordination.

Reading List

[B] A. S. Tanenbaum and M. Van Steen,, *Distributed Systems: Principles and Paradigms,* Prentice Hall, Inc..

[B] Rick Leander, *Building Application Servers,* Cambridge University Press, 2000.

[B] Daniel Serain, *Middleware and Enterprise Application Integration,* Springer, 2002.

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

Module Evaluation Course questionnaires, course review.

CO3091 Computational Intelligence and Software Engineering

Credits: *20* **Convenor:** *Dr. L. Minku* **Semester:** *1st*

Prerequisites:	*Essential: CO1003,CO1005*		*Desirable: CO2006, CO2015*		
Assessment:	*Coursework: 40%*		*Three hour exam in January: 60%*		
Lectures:	*29*	*hours*	**Problem Classes:**	*5*	*hours*
Surgeries:	*6*	*hours*	**Class Tests:**	*0*	*hours*
Laboratories:	*5*	*hours*	**Private Study:**	*105*	*hours*

Subject Knowledge

Aims Computational intelligence is a field of artificial intelligence concerned with so-called heuristic algorithms, which aim to produce good solutions to problems in a reasonable amount of time. These algorithms are widely used for several real world applications, e.g., routing problems; assignment and scheduling problems; medical, biomedical and bioinformatics problems; forecasting problems; etc. More recently, they have also started to be used to help solving software engineering problems. In particular, due to the increased size and complexity of software systems, software engineering tasks such as software project planning, software testing and maintenance have become increasingly time consuming and error prone. Computational intelligence techniques can be used as decision support tools in order to produce higher quality software faster, helping to overcome the challenges posed by large and complex software systems. This module explains computational intelligence approaches that can be used for solving problems from several different domains. It also explores the synergies between computational intelligence and software engineering, explaining how computational intelligence approaches can be used to help solving software engineering problems.

Learning Outcomes By completion of this module, students should be able to: recognise which software engineering problems can be formulated as computational intelligence optimisation or machine learning problems; formulate such software engineering problems as optimisation or machine learning problems; demonstrate an understanding of the core techniques used in the computational intelligence approaches to solve such problems; communicate such core techniques to software engineers; build models able to support software engineering learning tasks; use optimisation algorithms to support software engineering optimisation problems; evaluate, analyse and critique computational intelligence approaches for software engineering. It is also expected that students will be able to use their knowledge to non-software engineering problems.

Methods Class sessions together with lecture slides; recommended book chapters, articles and research papers; web resources; worksheets.

Assessment Marked coursework and written examination.

Skills

Aims Students will learn how to: research current issues in computational intelligence for software engineering; write reports; solve problems; and use computational intelligence toolboxes.

Learning Outcomes Upon completion of this module, students should be able to: research a given topic using a variety of sources including books, current articles and research papers; write reports; and use computational intelligence toolboxes.

Methods Class sessions; lab sessions; articles and research papers; and web resources.

Assessment Marked coursework and written examination.

Explanation of Prerequisites Coursework and lab sessions involving programming will require CO1003 and CO1005, besides experience of programming in Java. The module will give a computational

intelligence perspective to some of the problems introduced in CO2006 and CO2015. Having taken these modules is not required for CO3091, but would provide useful background knowledge.

Course Description Computational intelligence is a field of artificial intelligence concerned with so-called heuristic algorithms, which aim to produce good solutions to problems in a reasonable amount of time. These algorithms are widely used for several real world applications, e.g., routing problems; assignment and scheduling problems; medical, biomedical and bioinformatics problems; forecasting problems; etc. More recently, they have also started to be used to help solving software engineering problems. In particular, software systems have become ever larger and more complex, making software engineering tasks such as software project planning, software testing and maintenance increasingly costly and error prone. Many of these tasks can be formulated as problems that can be solved with the help of computational intelligence. If properly designed, computational intelligence approaches can be used as decision support tools to increase software development cost-effectiveness and improve software quality.

This module teaches computational intelligence approaches that can be applied to problems from several different domains. It also explores the synergies between computational intelligence and software engineering. It explains (1) software engineering problems that can be solved with the help of computational intelligence, (2) when and why computational intelligence is important for solving these problems, (3) how to formulate these problems in order to use computational intelligence approaches to solve them, (4) what computational intelligence approaches can be used to solve these problems, (5) how these approaches work and (6) how to use these approaches.

In particular, this module explains both optimisation and machine learning approaches. Optimisation problems are concerned with finding a solution that achieves one or more pre-defined goals. For example, software test case generation can be seen as the problem of finding a set of test cases that maximise code coverage and minimise testing effort. Computational intelligence algorithms such as evolutionary algorithms inspired by Darwin's evolutionary theory can be used to solve this problem, saving the time to create a good set of test cases manually and helping to improve software quality.

Machine learning problems are concerned with creating models able to infer knowledge from data. For example, models can be created to learn the relationship between software code metrics and information on whether the corresponding piece of code contains bugs or not. These models can then be used to predict whether new pieces of code are likely to contain bugs, providing an insight into what pieces of code require more or less testing effort. Computational intelligence algorithms such as naive bayes can be used to solve this problem.

Detailed Syllabus

Examples of computational intelligence approaches covered by this module are:

- Optimisation approaches: evolutionary algorithms and multi-objective evolutionary algorithms.
- Machine learning approaches: k-nearest neighbour, decision trees and naive-bayes.

Examples of software engineering problems covered by this module are:

- Requirements engineering: requirements selection.
- Project management: software effort estimation and software project scheduling.
- Development and maintenance: code optimisation for non-functional requirements.
- Testing and debugging: software defect prediction and software test suite generation.

This module will also explain procedures that can be used to evaluate the suitability of computational intelligence approaches to a given problem.

Reading List

[B] Russel, S.; Norvig, P.; John F. Canny, *Artificial Intelligence – A Modern Approach, Section 4.1: Local Search Algorithms and Optimization Problems, 3rd Edition (New International Edition),* Pearson Education, 2014.

[B] Michel, B.; Mancoridis, S., *Using Heuristic Search Techniques To Extract Design Abstractions From Source Code,* Proceedings of the Genetic and Evolutionary Computation Conference, Pages 1375-1382, 2002. Read section 3 until the end of section 3.1.1.

[B] Menzies, T.; Kocaguneli, E.; Minku, L.; Peters, F. and Turhan, B., *Sharing Data and Models in Software Engineering, Chapters 7 (Data Mining and Software Engineering), 8 (Defect Prediction), 9 (Effort Estimation), 10.10 (Extensions [of Decision Trees] to Continuous Classes),* Morgan Kaufmann, 2014.

[B] Eiben, A.; Smith, J., *Introduction to Evolutionary Computing, Chapters 1–8,* Springer, 2003.

[B] Turhan, B.; Menzies, T.; Bener, A.; Di Stefano, J., *On the Relative Value of Cross-Company and Within-Company Data for Defect Prediction, Section 3.2 (Naive Bayes Classifier),* Empirical Software Engineeing 14:540578, 2009.

[B] Fraser, G.; Arcuri, A., *Whole Test Suite Generation,* IEEE Transactions on Software Engineering 39(2):276-291, 2013.

[B] Minku, L.; Sudholt, D.; Yao, X., *Improved Evolutionary Algorithm Design for the Project Scheduling Problem Based on Runtime Analysis,* IEEE Transactions on Software Engineering 40(1):83-102, 2014.

[B] Linares-Vasquez, M.; Bavota, G.; Cardenas, C.; Oliveto, R.; Di Penta, M.; Poshyvanyk, D., *Optimizing Energy Consumption of GUIs in Android Apps: A Multi-objective Approach,* ACM SIGSOFT Symposium on the Foundations of Software Engineering, p. 143-153, 2015.

[B] Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T., *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,* IEEE Transactions on Evolutionary Computation, 6(2):182-197, 2002.

[B] Chen, W.; Zhang, J., *Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler,* IEEE Transactions on Software Engineering 39(1):1-17, 2013.

[B] Rosen, C.; Grawi, B.; Shihab, E., *Commit Guru: Analytics and Risk Prediction of Software Commits,* Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, p. 966-969, 2015.

[C] Le Goues, C.; Nguyen, T.; Forrest, S.; Weimer, W., *GenProg: A Generic Method for Automatic Software Repair,* IEEE Transactions on Software Engineering 38(1):54-72, 2012.

[C] Shin Yoo, *Evolving Human Competitive Spectra-Based Fault Localisation Techniques,* Symposium on Search Based Software Engineering, Lecture Notes in Computer Science 7515:244-258, 2012.

[C] Harman, M.; Jones, B., *Search-based software engineering,* Information and Software Technology 43(14):833-839, 2001.

[C] Fraser, G.; Arcuri, A., *A Large Scale Evaluation of Automated Unit Test Generation Using EvoSuite,* ACM Transactions on Software Engineering and Methodology 24(2):8, 2014.

[C] Le Goues, C.; Dewey-Vogt, M.; Forrest, S.; Weimer, W., *A Systematic Study of Automated Program Repair: Fixing 55 out of 105 bugs for $8 Each,* International Conference on Software Engineering, p. 3-13, 2012.

[C] Kamei, Y.; Shihab, E.; Adams, B.; Hassan, A.; Mockus, A.; Sinha, A.; Ubayashi, N., *A Large-Scale Empirical Study of Just-In-Time Quality Assurance,* IEEE Transactions on Software Engineering 39(6):757-773, 2013.

[C] Eiben, A.; Smith, J., *Introduction to Evolutionary Computing,* Springer, 2003.

Resources Lecture slides; books, articles and research papers; web page; study guide; worksheets; lecture rooms with projectors; labs with projectors; open source computational intelligence toolboxes.

Module Evaluation Course questionnaires, course review.

CO3093 Big Data and Predictive Analytics

Credits: *20* **Convenor:** *Dr. E. Tadjouddine* **Semester:** *2nd*

Prerequisites:	*Essential: CO1005, CO1008*	*Desirable: CO2015, CO3091*	
Assessment:	*Coursework: 60%*	*Two hours exam in May/June: 40%*	
Lectures:	*20 hours*	**Class Tests:**	*2 hours*
Surgeries:	*10 hours*	**Private Study:**	*98 hours*
Laboratories:	*20 hours*		

Subject Knowledge

Aims This module is aimed at teaching tools and methods to interrogate data in order to carry out predictive analytics and to evaluate the resulting models.

Learning Outcomes Students should be able to: analyse possibly large amount of data; use a Map-Reduce approach in processing data; develop and back-test a predictive model; compare and contrast different types of predictive models; and evaluate a given model.

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

Assessment

Summative: Coursework assessments including class tests and traditional written problem-based examination.

Skills

Aims To teach students scientific writing and problem solving skills.

Learning Outcomes Students will be able to: appreciate niche applications wherein predictive analytics can be useful; write a technical report based on design and experiments and solve abstract and concrete problems involving possible vast amount of data (structured or unstructured).

Methods Lab sessions together with worksheets.

Assessment Summative: Coursework assessments including class tests and traditional written problem-based examination.

Explanation of Prerequisites In addition to the pre-recquisite modules above, the basics of calculus will be helpful. The ability to program in Python is desirable but this will be part of the Lab sessions.

Course Description

As we increasingly rely upon the online environment for our daily routines, we leave behind a vast amount of information about us. Commercial and public organisations can use this information to predict our behavior. This module aims to study methods and tools enabling us to identify variables of interest and their relationships from an existing data set in order to develop a statistical model that can predict values of variables of interest. This kind of analysis should give us an insight into individual preferences, and most importantly, what someone is likely to do in a given scenario. Some of the applications include credit bank approval, marketing, stock price predictions, demand forecasting or polical campaining.

In this course, we will also study the importance of good quality data and will rely upon open libraries such as scikit-learn (http://scikit-learn.org/stable/) to implement basic models with much less programming effort. We will also learn how to compare and contrast different models for the same data and objective. As a predictive analysis does not necessarily demand a huge amount of data, we will also

discuss the utility or misfortune of the so-called big data and how to process such a large amount of data efficiently by using a distributed approach as in the Apache Spark.

Detailed Syllabus

Python: the basics and relevant libraries such as numpy, scipy, and pandas and its visualisation tools will be reviewed typically during the first three Lab sessions.

Big data: its philosophy (seek, store, analyse, and act); structured and unstructured data; data sources, volume, value, and timeliness; and data cleaning and manipulation.

Basic probability and statistics: random variables, mean, variance, standard deviation; relationship between variables (correlation and regression); probability distributions including Poisson and Gaussian; sampling; hypothesis testing, confidence intervals and significance tests.

Using predictive models: setting up clear objectives, identifying predictor data and outcome data, and the decision making process. We will use the example of a credit scoring model.

Types of predictive models: regression and classification models including linear models, logistic regression, random forests, clustering, and finding similar items. These algorithms will be presented in the form of black-box and be used from within a software package, e.g., http://scikit-learn.org/stable/

Building up a predictive model: data collection, sampling, and the iterative process (understanding the data, modeling, and performance evaluation). We will run through an example e.g., stock price movements or marketing.

Processing big data: The Apache Spark tool. We will introduce the basics of how to efficiently process large data sets.

Reading List

[B] David M. Levine and David F. Stephan, *Even You Can Learn Statistics and Analytics: An Easy to Understand Guide to Statistics and Analytics (Third Edition);* *ISBN: 978-0133382662,* Pearson FT Press.

[B] Steven Finlay, *Predictive Analytics, Data Mining and Big Data (Business in the Digital Economy);* *ISBN: 9781137379276,* Palgrave Macmillan.

[B] Philipp K. Janert, *Data Analysis With Open Source Tools;* *ISBN: 9787564126742,* O'Reilly Media.

[B] Ashish Kumar, *Learning Predictive Analytics with Python;* *ISBN 978-1783983261,* PACKT Publishing.

[B] P.N. Tan and M. Steinbach and V. Kumar, *Introduction to Data Mining;* *ISBN: 9787111316701,* Pearson.

[B] Anand Rajaraman and Jeffrey D. Ullman, *Mining of Massive Datasets;* *ISBN: 9781207015357,* Cambridge University Press.

[B] Viktor Mayer-Schonberger and Kenneth Cukier, *Big data: A revolution that will transform how we live, work and think;* *ISBN: 9781848547926,* John Murray.

[B] Frank J. Ohlhorst, *Turning Big Data into Big Money;* *ISBN: 9781118147597,* Wiley.

[B] John M. Chambers, *Software for Data Analysis: Programming with R (Statistics and Computing);* *ISBN: 9781441926128,* Springer.

[B] Kevin Sheppard, *Introduction to Python for Econometrics, Statistics and Data Analysis (free eBook, August 2014);* *ISBN: N/A,* KevinSheppard.com.

[B] Charles Severance, *Python for Informatics: Exploring Information (free eBook, May 2014); ISBN: N/A,* pythonlearn.com.

Resources Course notes, web page, study guide, handouts, lecture, mock tests or examinations, rooms with two OHPs.

Module Evaluation Course questionnaires, course review.

CO3095 Software Measurement and Quality Assurance

Credits: *20* **Convenor:** *Dr N. Walkinshaw* **Semester:** *1st*

Prerequisites:	*Essential: CO1003, CO1005, CO1008, CO2006*	*Desirable: CO1019, CO2012*	
Assessment:	*Coursework, two class tests: 40%*	*Three hour exam in January: 60%*	
Lectures:	*32 hours*	**Class Tests:**	*2 hours*
Surgeries:	*10 hours*	**Private Study:**	*90 hours*
Laboratories:	*10 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; research papers; web resources.

Assessment Marked coursework, 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).

Methods Class sessions together with work sheets.

Assessment Marked coursework, written examination.

Explanation of Prerequisites Software quality is a broad concept, that encompasses several aspects of software development. On the one hand, software quality is concerned with the complexity of the software source code and its design. To learn about this, it is essential to have a prior understanding of Java and Object-Oriented design (CO1003, CO1005). However, there is also a broader picture; quality depends on the ability of developers to properly plan development, to estimate costs, and to follow suitable procedures to ensure that quality is maintained. Accordingly, the student will ideally have studied preliminary modules on requirements engineering (CO1008) and project management and professionalism (CO2012).

Course Description The course will introduce various concepts associated with quality and quality assurance. It will show how these do not only revolve around software as an entity, but also depend upon the underlying development procedures, and the ability to improve upon these as the software and its stakeholders evolve over time.

Detailed Syllabus

Quality Defining "Quality" with Software Quality Models. Different (often competing) perspectives of software quality.

Risk analysis and management Measurement Theory of measurement. Project size/cost estimation. Quality metrics, cost metrics and process metrics. Statistics: data collection and analysis.

Software inspections and testing An overview of approaches to assess the quality of a software system and its design. These include human-centric methods such as software inspections / code reviews, as well as software testing approaches. For the latter we will look at both white-box coverage-based testing approaches, as well as "black-box" functional testing techniques.

Tools and instrumentation Process improvement frameworks such as the Capability Maturity Model. Software quality assurance models and standards.

Reading List

[A] S. Kan, *Metrics and Models in Software Quality Engineering,* Addison Wesley.

[B] N. E. Fenton, *Software Metrics: A Rigorous Approach,* Chapman & Hall.

[B] I. Sommerville, *Software Engineering,* Addison Wesley.

Resources Course notes, Blackboard page, study guide, worksheets, lecture rooms with data-projector, past examination papers.

Module Evaluation Course questionnaires, course review.

CO3096 Compression Methods for Multimedia

Credits: *20* **Convenor:** *Prof. R. Raman* **Semester:** *2nd*

Prerequisites:	*Essential: CO1012 or CO1011*	*Desirable: CO1016,CO2016*	
Assessment:	*Coursework: 50%*	*Two hour exam in January: 50%*	
Lectures:	*30 hours*	**Problem Classes:**	*5 hours*
Surgeries:	*5 hours*	**Class Tests:**	*3 hours*
		Private Study:	*107 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 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 Introduction: Raw multimedia data representation, Transmission medium characteristics, Data compression, Adaptive and non-adaptive methods, Lossy and lossless compression, Introduction to information theory and Theoretical limits of compressibility. Compressing symbolic data: Run-length coding, Entropy coders: Huffman coding, arithmetic coding, Dictionary coders: LZ77, LZW, Other text compression methods: Block-sorting. Standard text compression utilities: compress, zip. Image compression: Monochrome, facsimile and grayscale compression, GIF compression, JPEG compression, Video compression: Frame-by-frame compression: M-JPEG. Inter-frame compression: MPEG. Audio compression: Speech coding: ADPCM; CD-quality audio: MPEG layer 3.

Reading List

[B] Ze-Nian Li and Mark S. Drew, *Fundamentals of Multimedia; ISBN: 0130618721,* Pearson Prentice Hall, 2004.

[B] Khalid Sayood, *Introduction to Data Compression; ISBN: 1558605584,* Morgan Kaufmann Publishers, 2006 (3rd edition).

[C] Roy Hoffman, *Data compression in digital systems; ISBN: 0412085518,* Chapman and Hall Digital Multimedia Standards Series, 1997.

[C] Andrew S. Tanenbaum, *Structured Computer Organization; ISBN: 0130204358,* Prentice Hall, 1999 (5th edition).

[C] Jean-loup Gailly, *The comp.compression FAQ,* www.faqs.org/faqs/compression-faq/.

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.

CO3098 Web Technologies

Credits: *20* **Convenor:** *Dr. Y Hong* **Semester:** *1st*

Prerequisites:	*Essential: CO1003, CO1005, CO1019, CO2006*				
Assessment:	*Coursework: 50%*		*Two hour exam in January: 50%*		
Lectures:	*14*	*hours*	**Class Tests:**	*1*	*hours*
Surgeries:	*16*	*hours*	**Private Study:**	*100*	*hours*
Laboratories:	*19*	*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 the following specific aspects: XML, JSON, Java Servlet, AJAX, Web MVC Frameworks and Web Services.

Learning Outcomes At the end of the course the student should be able to: Understand the architectural foundations for Web Technologies, understand XML/JSON and AJAX based techniques and use Java Servlet and JavaScript technolgy appropriately to create web applications and handle data, be aware of security and session handling issues, decrease server load and increase user-perceived performance of the web application, deploy and use Web service to bring together a variety of web data from multiple sources and combine them to create an integrated experience.

Methods Class sessions, flipped classroom, 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 the knowledges and skills for developing web applications.

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, Markup Language (HTML), Database (SQL) and Programming in Object Oriented Paradigms (Java) and Scripting Language (JavaScript) 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. Java Servlets, Java Script) – 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 are XML, JSON and their related technologies. XML Schema/ XSLT, server-side and client-side scripting languages (Java and JavaScript), AJAX, and Web Design (HTML5). The course will also look at Web Services and discussing how they

can be combined with other technologies for creating 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, overview of web frameworks

Current Web data standards: XML and JSON related technologies, such as XML Schema/XSLT, XML and JSON parsing, as well as programming supports for them

Java servlets: designing and deploying Java Servlets and Java Server Page (JSP), session handling with cookies, sessions with servlet session APIs

Client-server communication AJAX, develping single-page web application using JQuery and Bootstrap.

Web services: Building, consuming and deploying SOAP/RESTful Web Services, Data Mashup

Web frameworks: Web MVC Frameworks, AngularJS and Spring MVC

Reading List

[C] Moller and Schwartzbach, *An Introduction to XML and Web Technologies; ISBN: 0-321-26966-7,* Addison Wesley, 2006.

[C] Hunter and Crawford, *Java Servlet Programming (2nd ed); ISBN: 0596000405,* O'Reilly,2001.

[C] Deitel, Deitel and Nieto, *Internet and World Wide Web: How to Program (2nd edition); ISBN: 0131218557,* Prentice Hall, 2002.

[C] Jon Duckett, *JavaScript & JQuery: Interactive Front-end Web Development,* ISBN: 1118531647.

[C] Budi Kurniawan and Paul Deck, *Servlet, JSP and Spring MVC: A Tutorial,* ISBN: 1771970022.

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

Module Evaluation Course questionnaires, course review.

CO3099 Cryptography and Internet Security

Credits: *20* **Convenor:** *Dr. S. Fung and Dr. E. Tuosto* **Semester:** *2nd*

Prerequisites:	*Essential: CO1003, CO1005*	*Desirable: CO1012, CO2017*	
Assessment:	*Coursework: 40%*	*Three hour exam in May/June: 60%*	
Lectures:	*30 hours*		
Surgeries:	*10 hours*	**Private Study:**	*100 hours*
Laboratories:	*10 hours*		

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; write cryptographically secure network applications using Java's cryptographic libraries; 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 CO1005, 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).

Internet Security: Applications of cryptographic techniques in distributed applications. Access control in Java. Enforcement of security properties through cryptography. Pretty Good Privacy (PGP). Digital Signatures. Digital certificates. Kerberos. Secure Electronic Transaction (SET).

Reading List

[B] W. Stallings, *Cryptography and Network Security; ISBN: 0133354695,* Prentice Hall. 2013.

[B] J. Knudsen, *Java Cryptography; ISBN: 1565924029,* O'Reilly. 1998.

[B] S. Oaks, *Java Security; ISBN: 0596001576,* O'Reilly. 2001.

[B] B. Schneier, *Applied Cryptography; ISBN: 0471128457,* John Wiley and sons. 1996.

[B] S. Garfinkel, *PGP: Pretty Good Privacy; ISBN: 1565920988,* O'Reilly. 1994.

[B] S. Garfinkel with G. Spafford, *Web Security, Privacy & Commerce; ISBN: 0596000456,* O'Reilly. 2001.

Resources Course notes, web page, study guide, worksheets, handouts, past examination papers.

Module Evaluation Course questionnaires, course review.

CO3105 Advanced C++ Programming

Credits: *20* **Convenor:** *Dr N. Piterman* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures:	*14*	*hours*			
Tutorials:	*7*	*hours*	**Private Study:**	*115*	*hours*
Laboratories:	*14*	*hours*			

Subject Knowledge

Aims This module teaches advanced C++ programming.

Learning Outcomes Students should be able to: understand the components of a C++ program, the structures required to write advanced programs, and the ideas of object orientation.

Methods Class sessions, recommended textbook and worksheets.

Assessment Marked coursework.

Skills

Aims To develop design, analysis and problem solving skills.

Learning Outcomes Students will be able to apply their C++ skills to solve computing problems.

Methods Class sessions together with worksheets.

Assessment Marked coursework.

Explanation of Prerequisites It is assumed that students are already familiar with a programming language such as Fortran, Python 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 32 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 programmer wishing to develop efficient programs should be familiar with the use of its central features. In addition, object orientation, has become a central dogma in programming languages. This module is intended to give the student a basic grasp of the use of C++ and the underlying principles of object oriented programming.

The module covers important aspects of object oriented programming and object oriented design. Assessment is done by programming exercises giving students an opportunity to practice the taught principles and improving their programming skills.

Detailed Syllabus

- Introduction to C++.
- Strings and Streams.
- Classes, constructors and destructors.
- Pointers, arrays and references.
- Exceptions.

- Methods and operators.
- Overloading.
- Templates and inheritance.
- The Standard Template Library and its data structures.
- Functional C++.

Reading List

[A] John R. Hubbard, *Programming with C++, 2nd edition; Shaum's Outlines; ISBN 007135346-1,* McGraw Hill.

[C] Bjarne Stroustrup, *The C++ Programming Language, 4th edition; ISBN: 0321563840,* Addison-Wesley, 2013.

[C] Andrew Koenig and Barbara E. Moo, *Accelerated C++; ISBN: 0-201-70353-X,* Addison Wesley, 2001.

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

Module Evaluation Course questionnaires, course review.

CO3120 Computing with Management Project

Credits: *40* **Convenor:** *Dr. S. 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.

CO4015 MComp Computer Science Project

Credits: *30* **Convenor:** *Dr. S. 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.

CO4105 Advanced C++ Programming

Credits: *20* **Convenor:** *Dr N. Piterman* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures:	*14*	*hours*			
Tutorials:	*7*	*hours*	**Private Study:**	*115*	*hours*
Laboratories:	*14*	*hours*			

Subject Knowledge

Aims This module teaches advanced C++ programming.

Learning Outcomes Students should be able to: understand the components of a C++ program, the structures required to write advanced programs, and the ideas of object orientation.

Methods Class sessions, recommended textbook and worksheets.

Assessment Marked coursework.

Skills

Aims To develop design, analysis and problem solving skills.

Learning Outcomes Students will be able to apply their C++ skills to solve computing problems.

Methods Class sessions together with worksheets.

Assessment Marked coursework.

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 32 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 programmer wishing to develop efficient programs should be familiar with the use of its central features. In addition, object orientation, has become a central dogma in programming languages. This module is intended to give the student a basic grasp of the use of C++ and the underlying principles of object oriented programming.

The module covers important aspects of object oriented programming and object oriented design. Assessment is done by programming exercises giving students an opportunity to practice the taught principles and improving their programming skills.

Detailed Syllabus

- Introduction to C++.
- Strings and Streams.
- Classes, constructors and destructors.
- Pointers, arrays and references.
- Exceptions.

- Methods and operators.
- Overloading.
- Templates and inheritance.
- The Standard Template Library and its data structures.
- Functional C++.

Reading List

[A] John R. Hubbard, *Programming with C++, 2nd edition; Shaum's Outlines; ISBN 007135346-1,* McGraw Hill.

[C] Bjarne Stroustrup, *The C++ Programming Language, 4th edition; ISBN: 0321563840,* Addison-Wesley, 2013.

[C] Andrew Koenig and Barbara E. Moo, *Accelerated C++; ISBN: 0-201-70353-X,* Addison Wesley, 2001.

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

Module Evaluation Course questionnaires, course review.

CO4200 Algorithms for Bioinformatics

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

Prerequisites:	*Desirable: Java programming*		
Assessment:	*Coursework: 40%*	*1.5 hour exam in May/June: 60%*	
Lectures:	*24 hours*	**Problem Classes:**	*4 hours*
Laboratories:	*12 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 abe 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

[B] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, *Biological sequence analysis – Probabilistic models of proteins and nucleic acids,* Cambridge University Press, 1998.

[B] N.C. Jones, P.A. Pevzner, *An introduction to bioinformatics algorithms,* MIT Press, 2004.

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

Module Evaluation Course questionnaires, course review.

CO4203 Advanced C++ Programming

Credits: *15* **Convenor:** *Dr N. Piterman* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures:	*14*	*hours*			
Tutorials:	*7*	*hours*	**Private Study:**	*77.5*	*hours*
Laboratories:	*14*	*hours*			

Subject Knowledge

Aims This module teaches advanced C++ programming.

Learning Outcomes Students should be able to: understand the components of a C++ program, the structures required to write advanced programs, and the ideas of object orientation.

Methods Class sessions, recommended textbook and worksheets.

Assessment Marked coursework.

Skills

Aims To develop design, analysis and problem solving skills.

Learning Outcomes Students will be able to apply their C++ skills to solve computing problems.

Methods Class sessions together with worksheets.

Assessment Marked coursework.

Explanation of Prerequisites It is assumed that students are already familiar with a programming language such as Fortran, Python 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 32 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 programmer wishing to develop efficient programs should be familiar with the use of its central features. In addition, object orientation, has become a central dogma in programming languages. This module is intended to give the student a basic grasp of the use of C++ and the underlying principles of object oriented programming.

The module covers important aspects of object oriented programming and object oriented design. Assessment is done by programming exercises giving students an opportunity to practice the taught principles and improving their programming skills.

Detailed Syllabus

- Introduction to C++.
- Strings and Streams.
- Classes, constructors and destructors.
- Pointers, arrays and references.
- Exceptions.

- Methods and operators.
- Overloading.
- Templates and inheritance.
- The Standard Template Library and its data structures.
- Functional C++.

Reading List

[A] John R. Hubbard, *Programming with C++, 2nd edition; Shaum's Outlines; ISBN 007135346-1,* McGraw Hill.

[C] Bjarne Stroustrup, *The C++ Programming Language, 4th edition; ISBN: 0321563840,* Addison-Wesley, 2013.

[C] Andrew Koenig and Barbara E. Moo, *Accelerated C++; ISBN: 0-201-70353-X,* Addison Wesley, 2001.

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

Module Evaluation Course questionnaires, course review.

CO4205 Advanced System Design

Credits: *15* **Convenor:** *Dr. E. Tuosto* **Semester:** *1st*

Prerequisites: *none*

Assessment: *Coursework: 100%*

Lectures:	*24*	*hours*	**Problem Classes:**	*2*	*hours*
Surgeries:	*8*	*hours*	**Private Study:**	*72.5*	*hours*
Laboratories:	*8*	*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 software architectures in general and modelling techniques for distributed applications in particular.

Learning Outcomes At the end of the course, students should be able to: understand the basic concepts and role of software architectures, including the separation between computation and coordination concerns; understand the relations between models and implementations of distributed applications; apply techniques to support the modelling, testing, and programming of distributed applications.

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

Assessment Assessed classtests.

Skills

Aims To teach students abstraction and higher-level modelling skills, with special emphasis on architectural and coordination aspects 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 message-based distributed applications; modularise applications by identifying the dependencies that interactions have on distributed parties; model service orchestrations; model the protocols that coordinate service interactions.

Methods Class and practical 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. Familiarity with message sequence charts (or sequence diagrams) and with state machines.

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 distributed applications in particular.

Detailed Syllabus

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

Software Architectures Architectures of usage vs interaction; components versus connectors; architectural styles; nature and role of architectural description languages.

Modelling for Service-Oriented Architectures Models for capturing requirements on distributed applications: the roles distributed components. Conversational protocols and patterns for interactions. Choreography of complex services.

Reading List

[B] E. Tuosto, *Choreography for Distributed Applications,* Notes.

[B] M.Shaw and D.Garlan, *Software Architectures: Perspectives on an Emerging Discipline; ISBN: 0131829572,* Prentice Hall, 1996.

[C] Bass, Clements, Kasman, *Software Architecture in Practice; ISBN: 0321154959,* Addison Wesley, 1998.

Resources Study guide, lecture rooms with data projector, 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: 100%*	*Exam: 0%*	
Lectures:	*20 hours*		
Surgeries:	*7 hours*	**Private Study:**	*71.5 hours*
Laboratories:	*14 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 Three assignments.

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 Three assignments.

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 Deterioration: How and why software systems deteriorate.

Software Maintenance and Reengineering: What are the challenges involved in re-engineering a system.

Software Comprehension: Basic skills of manually analysing and comprehending the functionality of a complex, unfamiliar system.

Automated and Semi-Automated Software Analysis: Learning basic static and dynamic program analysis techniques.

Assessing Software Vulernabilities and Weaknesses: What are the sources of problems in a software system? Which areas need to be re-engineered most urgently?

Regression Testing: Essential testing skills to make sure that re-engineering does not introduce any new bugs.

Restructuring and Migration: Basic strategies to restructure and migrate a software system.

Reading List

[**A**] Neil Walkinshaw,, *Reverse Engineering Software Behaviour,* Elsevier Advances in Computers, vol 91, 2013.

[**A**] Oscar Nierstrasz, Stéphane Ducasse, Serge Demeyer,, *Object-Oriented Reengineering Patterns,* Morgan Kaufmann Publishers, 2003.

[**A**] Elliot Chikofsky and James Cross, *Reverse Engineering and Design Recovery: A Taxonomy,* IEEE Software 7(1):13-17, 1990.

Resources

Course notes.

Module Evaluation

Course questionnaires, course review.

CO4207 Generative Development

Credits: *15* **Convenor:** *Dr R Chitchyan* **Semester:** *2nd*

Prerequisites:	*Desirable: UML, Java, Eclipse*				
Assessment:	*Coursework: 100%*				
Lectures:	*24*	*hours*			
Surgeries:	*8*	*hours*	**Private Study:**	*64.5*	*hours*
Laboratories:	*16*	*hours*			

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: demonstrate a knowledge of the main approaches for automating software development; explain concepts of aspect-oriented programming and apply them; explain concepts of software product line development and apply them.

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

Assessment Assessed individual and group coursework assignments, in-class tests.

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: 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.

Methods Class sessions together with worksheets and practical programming experience.

Assessment Assessed individual and group coursework assignments, in-class tests.

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

[B] Thomas Stahl, Markus Völter, *Model-Driven Software Development,* John Wiley and Sons. 2006.

[B] Perdita Stevens, *Using UML : Software Engineering With Objects and Components,* Addison Wesley. Second edition. 2005.

[B] Jos Warmer, Anneke Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA,* Second Edition. Addison-Wesley. 2003.

[B] Joseph D. Gradecki, Nicholas Lesiecki, *Mastering AspectJ: Aspect-Oriented Programming in Java,* Wiley, 2003.

[C] Frank Budinsky; David Steinberg; Ed Merks; Raymond Ellersick; Timothy J. Grose, *Eclipse Modeling Framework: A Developer's Guide,* Addison Wesley Professional. 2003.

[C] Martin Fowler, *UML Distilled,* Third Edition. Addison-Wesley. 2004.

[C] K. Czarnecki, U. Eisenecker, *Generative Programming,* Addison-Wesley, 2000.

[C] David S. Frankel, *Model Driven Architecture,* Wiley, 2003.

[C] B. Bruegge, A. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems,* Prentice Hall, 2003.

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

Module Evaluation Course questionnaires, course review.

CO4209 Software Reliability

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

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

[B] Peled, Doron, *Software Reliability Methods,* Springer-Verlag, 2001.

[C] Holzmann, Gerard, *Spin Model Checker: Primer and Reference Manual,* Addison Wesley, 2003.

[C] McMillan, Ken, *Symbolic model checking,* Kluwer Academic Publishers, 1993.

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

Module Evaluation Course questionnaires, course review.

CO4210 Personal and Group Skills

Credits: *15* **Convenor:** *Prof T Erlebach, Prof A Kurz* **Semester:** *1st and 2nd*

Prerequisites:	*none*		
Assessment:	*Group discussions: 30%* *Oral presentation: 20%*	*Collective essay: 50%*	
Lectures:	*6 hours*	**Other:**	*4 hours*
Seminars:	*8 hours*	**Private Study:**	*94.5 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;
- demonstrate knowledge of 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, 4,000 word 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.

CO4211 Discrete Event Systems

Credits: *15* **Convenor:** *Dr M Hoffmann* **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:	*Two hour exam in May/June: 100%*				
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;

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

Assessment 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 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.

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:

- The Petri Net Model of Discrete Event Systems.
- 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.
- Analysis of Markov Chains: Convergence and Transition Probabilities.
- Introduction to Queueing Theory.
- $M/M/1$, $M/M/1/N$ and $M/M/m$ queues.

Reading List

[B] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems,* Springer, 2008.

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

Module Evaluation Module questionnaires, course review.

CO4212 Game Theory in Computer Science

Credits: *15* **Convenor:** *Dr R van Stee* **Semester:** *2nd*

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

[A] Bernhard von Stengel, *Notes on Game Theory Basics,* Available from the module webpage.

[A] Rahul Savani, *Notes on Routing Games,* Available from the module webpage.

[A] Noam Nisan, Tim Roughgarden, Eva Tardos, Vijay V. Vazirani (Eds.), *Algorithmic Game Theory,* Cambridge University Press, 2007. We are likely to cover parts of Chapters 1, 2, 17 and 18. Online: http://www.cambridge.org/journals/nisan/downloads/nisan_non-printable.pdf.

[C] Martin J. Osbourne, *An Introduction to Game Theory,* Oxford University Press.

[C] Ken Binmore, *Playing for Real,* Oxford University Press.

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

Module Evaluation Course questionnaires, course review.

CO4214 Service-Oriented Architectures

Credits: *15* **Convenor:** *Prof. R. Heckel* **Semester:** *2nd*

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

[**A**] Francisco Curbera, Frank Leymann, Tony Storey, Donald Ferguson, Sanjiva Weerawarana, *Web Services Platform Architecture: Soap, WSDL, WS-Policy, WS-Addressing, WS-Bpel, WS-Reliable Messaging and More,* Prentice Hall. 2005.

[**A**] Tom Pender, *UML Bible,* Wiley Publishing Inc, 2003.

[**A**] Martin Fowler, Kendall Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language,* Object Technology S., 2003.

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

Module Evaluation Course questionnaires, course review.

CO4215 Advanced Web Technologies

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

Prerequisites:	*none*		
Assessment:	*Coursework: 40%*	*Two hour exam in January: 60%*	
Lectures:	*17 hours*		
Surgeries:	*3 hours*	**Private Study:**	*74.5 hours*
Laboratories:	*18 hours*		

Subject Knowledge

Aims The aim of this course is to teach the students the concepts, technologies and techniques for creating large-scale destributed software system using service oriented computing and cloud applications.

Learning Outcomes At the end of the course the student should be able to:

- define the fundamental ideas and standards underlying Web Service Technology;
- define the fundamental principles for cloud applications;
- discuss concepts at the frontier of industrial practice and emerging standards;
- differentiate the major frameworks allowing to develop web services and cloud applications and assess their suitability for specific usage scenarios;
- explain the link between the concepts of services and business processes and discuss and critique related standards;
- develop busines processes using the Workflow foundation.
- develop and deploy web services and cloud applications using appropriate Microsoft technologies.

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 its predominant implementation as Web Services are at the forefront of industrial practice in software engineering. There are two major technologies supporting WS 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. Finally, service applications need to be executed in a scalable fashion and cloud computing provides one possible deployment architecture. We will cosnider the main business drivers and advantages for adopting cloud computing and study details of some cloud computing platforms with practical exercises based on the Azure platform.

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 and cloud computing.

Reading List

[B] Papazoglou, *Web Services: Principles and Technology (2nd edition); ISBN: 978-0-273-73216-7,* Prentice Hall, 2012.

[B] Alonso, Casati, Kuno and Machiraju, *Web Services: Concepts, Architectures and Applications; ISBN: 3540440089,* Springer, 2004.

[B] Cerami, *Web Services Essentials; ISBN: 0596002246,* O'Reilly,2002.

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.

CO4216 Semantic Web

Credits: *15* **Convenor:** *Dr. Yi Hong* **Semester:** *2nd*

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

[A] Grigoris Antoniou, Frank Van Harmelen, *A Semantic Web Primer,* MIT Press, 2008.

[A] Pascal Hitzler, Markus Krotzsch, Sebastian Rudolph, *Foundations of Semantic Web Technologies,* CRC Press, 2009.

[B] Dean Allemang, James Hendler, *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL,* Morgan Kauffmann, ISBN-10: 0-12-373556-4.

[B] Geroimenko, Vladimir; Chen, Chaomei (Eds.) 2nd ed., 2006, XIV, 248 p. 108 illus., Hardcover ISBN: 978-1-85233-976-0, *Visualizing the Semantic Web XML-based Internet and Information Visualization,* Springer-Verlag London Ltd; 2Rev Ed edition (Oct 2005).

[B] Michael C. Daconta, Leo J. Obrst, Kevin T. Smith, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management: A Guide to the Future of XML, Web Services and Knowledge Management,* John Wiley & Sons (20 Jun 2003).

[B] S Powers, *Practical RDF (Paperback) ,* OReilly (1 Aug 2003).

[B] Thomas B. Passin, *Explorer's Guide to the Semantic Web (Paperback),* Manning Publications (8 Jul 2004).

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

Module Evaluation Course questionnaires, course review.

CO4217 Agile Cloud Automation

Credits: *15* **Convenor:** *Dr A. Boronat* **Semester:** *1st*

Prerequisites: *Desirable: UML, Java, Eclipse*

Assessment: *Coursework: 100%*

Lectures:	*8*	*hours*	**Class Tests:**	*2*	*hours*
Surgeries:	*8*	*hours*	**Private Study:**	*86.5*	*hours*
Laboratories:	*8*	*hours*			

Subject Knowledge

Aims This module introduces students to concepts, techniques and technologies for developing cloud-based systems using domain-specific languages (DSLs), NoSQL technology and agile practices. In particular, it will familiarise students with NoSQL principles, with techniques for specifying DSLs and modelling environments, and with designing and developing interoperable infrastructure for heterogeneous software ecosystems using agile practices.

Learning Outcomes At the end of this module, successful students will be able to: demonstrate understanding of NoSQL principles and technology; discuss issues and solution approaches for questions of scalability and consistency; explain agile principles and practices for developing cloud systems; demonstrate a systematic understanding of the specification of DSLs using formal meta-languages; explain and employ the relation between their surface characterization and semantics for verification purposes; apply model transformations for the effective design and implementation of meta-data environments in heterogeneous software ecosystems; defend a critical awareness of state-of-the-art model-driven principles, standards and practices and their relevance to software engineering and to cloud-based computing.

Methods Lectures, surgeries, laboratory sessions, recommended reading, worksheets, web support.

Assessment Formative coursework; assessed coursework; mini project; short technical essay.

Skills

Aims To teach students modelling and problem-solving skills.

Learning Outcomes At the end of this module, successful students will be able to: solve abstract and concrete problems (both routine seen, and simple unseen), including modelling aspects; write concise, clear statements of technical knowledge.

Methods Lectures, surgeries, laboratory sessions, recommended reading, worksheets, web support.

Assessment Formative coursework; assessed coursework; mini project; short technical essay.

Explanation of Prerequisites Basic knowledge of UML and some programming experience with Eclipse and Java will be helpful.

Course Description Cloud-based software system development involves a wide range of languages and notations, from high-level business modelling languages to low-level scripting languages, with many different abstraction facilities. Domain-specific languages (DSLs) help close this implementation gap by combining the most advantageous aspects of both business experts and developers, potentially improving software interoperability and quality, and increasing productivity and ROI in software development processes. As a result, there is a wide choice of languages both for implementing business logic − such as the number of DSLs built atop JavaScript that are born on a weekly basis − and for persisting big data in a scalable manner − such as the wide range of persistence platforms under the NoSQL umbrella. Hence, becoming polyglot in a cloud-based age is a invaluable skill for managers and developers.

In *Agile Cloud Automation*, we will discuss an outline of approaches and technologies for the agile

development of cloud-based systems.

Detailed Syllabus In detail, the module will cover the following topics (to be updated):

Introduction to principles, standards and practices regarding the specification of DSLs using agile practices.

Introduction to NoSQL principles, including consistency and scalability concerns, using MongoDB.

Specification of the infrastructure of a DSL by using OMG standards, including MOF and UML profiles.

Specification of the superstructure of a DSL by using formal languages and compiler generation technology.

Model transformations and their role in heterogeneous software ecosystems and in big data scenarios.

Reading List

[A] Marco Brambilla, Jordi Cabot, and Manuel Wimmer, *Model-Driven Software Engineering in Practice,*Morgan & Claypool Publishers, 2012, ISBN-13 9781608458820.

[A] Pramod J. Sadalage, and Martin Fowler, *NoSQL distilled: a brief guide to the emerging world of polygot persistence,*Addison-Wesley, 2012, ISBN-13 9780321826626.

[B] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman., *Compilers : principles, techniques, and tools,* Pearson, second edition, 2006, ISBN-13 9780321491695, 9780321547989.

[B] Martin Fowler, *Domain-specific languages,* Addison-Wesley, 2011, ISBN-13 9780132107549.

[B] Markus Voelter and Sebastian Benz and Christian Dietrich and Birgit Engelmann and Mats Helander and Lennart C. L. Kats and Eelco Visser and Guido Wachsmuth, *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages,* dslbook.org, 2013, ISBN-13 9781481218580.

[C] Frank Budinsky; David Steinberg; Ed Merks; Raymond Ellersick; Timothy J. Grose, *Eclipse Modeling Framework: A Developer's Guide,* Addison Wesley Professional, 2009, ISBN-13 9780321331885.

Resources Course notes, module web page, study guide, worksheets, lecture rooms with data projectors, the Eclipse platform (Modeling Package), laboratories with PCs and demonstrators.

Module Evaluation Course questionnaires, course review.

CO4218 Financial Services Information Systems

Credits: *15* **Convenor:** *Dr. G. Koutsoukos* **Semester:** *2nd*

Prerequisites:	*none*		
Assessment:	*Coursework: 40%*	*Two hour exam in May/June: 60%*	
Lectures:	*25 hours*	**Private Study:**	*79.5 hours*
Surgeries:	*8 hours*		

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

[B] Essvale Corporation Limited, *Business Knowledge for IT in Retail Banking: The Complete Handbook for IT Professionals,* ISBN: 0955412420, 2007..

[B] Essvale Corporation Limited, *Business Knowledge for IT in Investment Banking: The Complete Handbook for IT Professionals,* ISBN: 1906096309, 2008..

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.

CO4219 Internet and Cloud Computing

Credits: *15* **Convenor:** *Prof T Erlebach* **Semester:** *1st*

Prerequisites:	*Desirable: Java programming*				
Assessment:	*Coursework: 40%*		*Two hour exam in January: 60%*		
Lectures:	*28*	*hours*	**Problem Classes:**	*3*	*hours*
Surgeries:	*6*	*hours*	**Class Tests:**	*1*	*hours*
Laboratories:	*2*	*hours*	**Private Study:**	*72.5*	*hours*

Subject Knowledge

Aims This module teaches the basic concepts and mechanisms underlying the Internet and the principles of Cloud Computing.

Learning Outcomes At the end of the module students will be able to: discuss the layered architecture, routing mechanisms and main protocols used in the Internet; demonstrate understanding of the principles, components and architecture of cloud computing; discuss issues and solution approaches for questions of privacy and security in the context of cloud computing; demonstrate understanding of mechanisms for enhancing fault-tolerance in cloud computing; discuss scalable approaches to distributed computing on large amounts of data, especially the MapReduce programming model, including application areas such as data analytics, data mining, and information retrieval.

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 problem solving skills relevant to networking and cloud computing.

Learning Outcomes Students will be able to: perform calculations to predict performance metrics of data transfers in different network scenarios; design and implement scalable computations in Java using the Hadoop framework.

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 in Java will be helpful.

Course Description Internet-based communication has become essential for our society and economy. Furthermore, the advent of cloud computing has revolutionised the way in which data is stored and processed. The first part of the module covers basic concepts of networking such as the Internet architecture, the sliding window protocol for reliable transfer, routing in the Internet, and end-to-end protocols. Students also learn to carry out calculations to predict network performance. The second part of the module focuses on cloud computing. It covers the principles of cloud computing, the MapReduce programming model for large-scale data processing, the implementation of MapReduce programs using Hadoop, and security considerations relevant to cloud computing.

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.

Overview of Internet architecture and mechanisms for routing and internetworking.

Principles, architecture of cloud computing, including aspects of fault-tolerance, privacy and security.

Scalable distributed computing using the MapReduce programming model, and implementation in Java using the Hadoop framework.

Reading List

[B] Larry Peterson and Bruce S. Davie, *Computer Networks: A Systems Approach,* 4th Edition, Morgan Kaufmann, 2007, ISBN 1-558-60832-X.

[B] Andrew S. Tanenbaum, *Computer Networks,* 4th Edition, Pearson, 2003, ISBN 0-13-038488-7.

[B] William Stallings, *Data and Computer Communications,* 7th Edition, Pearson, 2004, ISBN 0-13-183311-1.

[B] Jimmy Lin and Chris Dyer, *Data-Intensive Text Processing with MapReduce,* Morgan & Claypool, 2010, ISBN 978-1-608-45342-9.

[B] Tom White, *Hadoop: The Definitive Guide,* 3rd Edition, O'Reilly, 2012, ISBN 978-1-449-31152-0.

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.

CO7002 Analysis and Design of Algorithms

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

Prerequisites:	*none*				
Assessment:	*Coursework: 40%*		*Three hour exam in May/June: 60%*		
Lectures:	*30*	*hours*	**Problem Classes:**	*9*	*hours*
Surgeries:	*10*	*hours*	**Class Tests:**	*1*	*hours*
			Private Study:	*62.5*	*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 analysing 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

[B] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms, 3rd edition; ISBN: 978-0-262-53305-8,* MIT Press, 2009.

[B] S. Dasgupta, C. H. Papadimitriou and U. Vazirani, *Algorithms,* McGraw-Hill, 2007.

[B] J. Kleinberg and E. Tardos, *Algorithm Design,* Addison-Wesley, 2006.

[B] S. Skiena, *The Algorithm Design Manual, 2nd edition,* Springer, 2008.

Resources Course notes, web page, study guide, worksheets, past examination papers.

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 January: 60%*	
Lectures:	*38 hours*	**Class Tests:**	*2 hours*
Surgeries:	*10 hours*	**Private Study:**	*62.5 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 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, both by hand and with the aid of the Concurrency Workbench, using the techniques of equational reasoning and bisimulation, including bisimulation games.

Methods Class 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 sessions, course notes and text books, class 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

[A] R. Milner, *Concurrency and Communication; ISBN: 0131150073,* Prentice-Hall 1989.

[B] C.A.R. Hoare, *Communicating Sequential Processes; ISBN: 0131532898,* Prentice-Hall 1985.

[B] J.C. Baeten and W.P. Weijland, *Process Algebra; ISBN: 0521400430,* Cambridge University Press 1990.

[B] C. Fencott, *Formal Methods for Concurrency,* Thomson Computer Press 1996.

[B] A.W. Roscoe, *The Theory and Practice of Concurrency; ISBN: 0136744095,* Prentice-Hall 1997.

[B] S. Schneider, *Concurrent and Real-time Systems; ISBN: 0471623733,* Wiley 2000.

[B] W. Fokkink, *Introduction of Process Algebra; ISBN: 354066579X,* Springer 2000.

[B] C. Stirling, *Modal and Temporal Properties of Processes; ISBN: 0387987177,* Springer 2001.

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

Module Evaluation Course questionnaires, course review.

CO7090 Distributed Systems and Applications

Credits: *15* **Convenor:** *Yi Hong* **Semester:** *2nd*

Prerequisites:	*Essential: General knowledge on object-oriented programming, basic knowledge of Java, basic knowledge of software architectures*				
Assessment:	*Coursework: 40%*		*Three hour exam in May/June: 60%*		
Lectures:	*17*	*hours*	**Class Tests:**	*1*	*hours*
Surgeries:	*7*	*hours*	**Private Study:**	*73.5*	*hours*
Laboratories:	*14*	*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; Enterprise Java Beans; Distributed coordination.

Reading List

[B] A. S. Tanenbaum and M. Van Steen,, *Distributed Systems: Principles and Paradigms,* Prentice Hall, Inc..

[B] Rick Leander, *Building Application Servers,* Cambridge University Press, 2000.

[B] Daniel Serain, *Middleware and Enterprise Application Integration,* Springer, 2002.

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

Module Evaluation Course questionnaires, course review.

CO7094 System Modelling

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

Prerequisites: *none*

Assessment: *Three hour exam in January: 100%*

Lectures:	*20*	*hours*	**Problem Classes:**		*hours*
Surgeries:	*10*	*hours*	**Private Study:**	*100*	*hours*
Laboratories:	*20*	*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

- J. M. Spivey, *The Z Notation: a reference manual*, Prentice Hall, 2001: `http://spivey.oriel.ox.ac.uk/mike/zrm/`
- Jim Woodcock and Jim Davies, *Using Z : Specification, Refinement, and Proof*, Prentice Hall, 1996: `http://www.usingz.com/text/zedbook.pdf`
- Graeme Smith. *The Object-Z Specification Language.* Kluwer Academic Publishers, 2000.
- Roger Duke and Gordon Rose, *Formal object-oriented specification using Object-Z*, 2000, MacMillan Press.
- John Derrick and Eerke Boiten, *Refinement in Z and Object-Z*, Springer 2001.

Material on CSP

- C.A.R. Hoare, *Communicating Sequential Processes (CSP)*, Prentice Hall, 1985: `http://www.usingcsp.com/ (2004)`
- Bill Roscoe, *The theory and practice of concurrency*, Prentice Hall, 1997 (lightly revised 2005): `http://www.usingz.com/text/zedbook.pdf`
- M.G. Hinchey and S.A. Jarvis, *Concurrent Systems: formal developments in CSP*,McGraw-Hill, 1995.

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.

CO7095 Computer Systems

Credits: *15* **Convenor:** *Dr N. Walkinshaw* **Semester:** *1st*

Prerequisites:	*Essential: Knowledge of software requirements and design procedures, knowledge of Object-Oriented programming concepts*		
Assessment:	*Coursework, two class tests: 40%*	*Three hour exam in January: 60%*	
Lectures:	*32 hours*		
Surgeries:	*10 hours*	**Private Study:**	*90 hours*
Laboratories:	*10 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; research papers; web resources.

Assessment Marked coursework, 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).

Methods Class sessions together with work sheets.

Assessment Marked coursework, written examination.

Explanation of Prerequisites

Software quality is a broad concept, that encompasses several aspects of software development. On the one hand, software quality is concerned with the complexity of the software source code and its design. To learn about this, it is essential to have a prior understanding of Java and Object-Oriented design (CO1003, CO1005). However, there is also a broader picture; quality depends on the ability of developers to properly plan development, to estimate costs, and to follow suitable procedures to ensure that quality is maintained. Accordingly, the student will ideally have studied preliminary modules on requirements engineering (CO1008) and project management and professionalism (CO2012).

Course Description

The course will introduce various concepts associated with quality and quality assurance. It will show how these do not only revolve around software as an entity, but also depend upon the underlying development procedures, and the ability to improve upon these as the software and its stakeholders evolve over time.

Detailed Syllabus

Quality Defining "Quality" with Software Quality Models. Different (often competing) perspectives of software quality.

Risk analysis and management Measurement Theory of measurement. Project size/cost estimation. Quality metrics, cost metrics and process metrics. Statistics: data collection and analysis.

Software inspections and testing An overview of approaches to assess the quality of a software system and its design. These include human-centric methods such as software inspections / code reviews, as well as software testing approaches. For the latter we will look at both white-box coverage-based testing approaches, as well as "black-box" functional testing techniques.

Tools and instrumentation Process improvement frameworks such as the Capability Maturity Model. Software quality assurance models and standards.

Reading List

[A] S. Kan, *Metrics and Models in Software Quality Engineering,* Addison Wesley.

[B] N. E. Fenton, *Software Metrics: A Rigorous Approach,* Chapman & Hall.

[B] I. Sommerville, *Software Engineering,* Addison Wesley.

Resources Course notes, Blackboard page, study guide, worksheets, lecture rooms with 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: 50%*	*Two hour exam in May/June: 50%*	
Lectures:	*30 hours*	**Problem Classes:**	*5 hours*
Surgeries:	*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 Introduction: Raw multimedia data representation, Transmission medium characteristics, Data compression, Adaptive and non-adaptive methods, Lossy and lossless compression, Introduction to information theory and Theoretical limits of compressibility. Compressing symbolic data: Run-length coding, Entropy coders: Huffman coding, arithmetic coding, Dictionary coders: LZ77, LZW, Other text compression methods: Block-sorting. Standard text compression utilities: compress, zip. Image compression: Monochrome, facsimile and grayscale compression, GIF compression, JPEG compression, Video compression: Frame-by-frame compression: M-JPEG. Inter-frame compression: MPEG.

Reading List

[B] Ze-Nian Li and Mark S. Drew, *Fundamentals of Multimedia,* Pearson Prentice Hall, 2004.

[B] Khalid Sayood, *Introduction to Data Compression,* Morgan Kaufmann Publishers, 2000 (2nd edition).

[C] Roy Hoffman, *Data compression in digital systems,* Chapman and Hall Digital Multimedia Standards Series, 1997.

[C] Andrew S. Tanenbaum, *Structured Computer Organization,* Prentice Hall, 1999 (4th edition).

[C] Jean-loup Gailly, *The comp.compression FAQ,* www.faqs.org/faqs/compression-faq/.

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: *20* **Convenor:** *Dr. Y Hong* **Semester:** *1st*

Prerequisites:	*Essential: HTML, Java Programming*				
Assessment:	*Coursework: 50%*		*Two hour exam in January: 50%*		
Lectures:	*14*	*hours*	**Class Tests:**	*1*	*hours*
Surgeries:	*16*	*hours*	**Private Study:**	*100*	*hours*
Laboratories:	*19*	*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 the following specific aspects: XML, JSON, Java Servlet, AJAX, Web MVC Frameworks and Web Services.

Learning Outcomes At the end of the course the student should be able to: Understand the architectural foundations for Web Technologies, understand XML/JSON and AJAX based techniques and use Java Servlet and JavaScript technolgy appropriately to create web applications and handle data, be aware of security and session handling issues, decrease server load and increase user-perceived performance of the web application, deploy and use Web service to bring together a variety of web data from multiple sources and combine them to create an integrated experience.

Methods Class sessions, flipped classroom, 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 the knowledges and skills for developing web applications.

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, Markup Language (HTML), Database (SQL) and Programming in Object Oriented Paradigms (Java) and Scripting Language (JavaScript) 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. Java Servlets, Java Script) – 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 are XML, JSON and their related technologies. XML Schema/ XSLT, server-side and client-side scripting languages (Java and JavaScript), AJAX, and Web Design (HTML5). The course will also look at Web Services and discussing how they

can be combined with other technologies for creating 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, overview of web frameworks

Current Web data standards: XML and JSON related technologies, such as XML Schema/XSLT, XML and JSON parsing, as well as programming supports for them

Java servlets: designing and deploying Java Servlets and Java Server Page (JSP), session handling with cookies, sessions with servlet session APIs

Client-server communication AJAX, develping single-page web application using JQuery and Bootstrap.

Web services: Building, consuming and deploying SOAP/RESTful Web Services, Data Mashup

Web frameworks: Web MVC Frameworks, AngularJS and Spring MVC

Reading List

[C] Moller and Schwartzbach, *An Introduction to XML and Web Technologies; ISBN: 0-321-26966-7,* Addison Wesley, 2006.

[C] Hunter and Crawford, *Java Servlet Programming (2nd ed); ISBN: 0596000405,* O'Reilly,2001.

[C] Deitel, Deitel and Nieto, *Internet and World Wide Web: How to Program (2nd edition); ISBN: 0131218557,* Prentice Hall, 2002.

[C] Jon Duckett, *JavaScript & JQuery: Interactive Front-end Web Development,* ISBN: 1118531647.

[C] Budi Kurniawan and Paul Deck, *Servlet, JSP and Spring MVC: A Tutorial,* ISBN: 1771970022.

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

Module Evaluation Course questionnaires, course review.

CO7099 Cryptography and Internet Security

Credits: *15* **Convenor:** *Dr. S. Fung and Dr. E. Tuosto* **Semester:** *2nd*

Prerequisites:	*none*		
Assessment:	*Coursework: 40%*	*Three hour exam in May/June: 60%*	
Lectures:	*30 hours*		
Surgeries:	*10 hours*	**Private Study:**	*62.5 hours*
Laboratories:	*10 hours*		

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; write cryptographically secure network applications using Java's cryptographic libraries; 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).

Internet Security: Applications of cryptographic techniques in distributed applications. Access control in Java. Enforcement of security properties through cryptography. Pretty Good Privacy (PGP). Digital Signatures. Digital certificates. Kerberos. Secure Electronic Transaction (SET).

Reading List

[B] W. Stallings, *Cryptography and Network Security; ISBN: 0133354695,* Prentice Hall. 2013.

[B] J. Knudsen, *Java Cryptography; ISBN: 1565924029,* O'Reilly. 1998.

[B] S. Oaks, *Java Security; ISBN: 0596001576,* O'Reilly. 2001.

[B] B. Schneier, *Applied Cryptography; ISBN: 0471128457,* John Wiley and Sons. 1996.

[B] S. Garfinkel, *PGP: Pretty Good Privacy; ISBN: 1565920988,* O'Reilly. 1994.

[B] S. Garfinkel with G. Spafford, *Web Security, Privacy & Commerce; ISBN: 0596000456,* O'Reilly. 2001.

Resources Course notes, web page, study guide, worksheets, handouts, past examination papers.

Module Evaluation Course questionnaires, course review.

CO7100 Algorithms for Bioinformatics

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

Prerequisites:	*Desirable: Java programming*		
Assessment:	*Coursework: 50%*	*1.5 hour exam in May/June: 50%*	
Lectures:	*24 hours*	**Problem Classes:**	*4 hours*
Laboratories:	*12 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 abe 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

[B] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, *Biological sequence analysis – Probabilistic models of proteins and nucleic acids,* Cambridge University Press, 1998.

[B] N.C. Jones, P.A. Pevzner, *An introduction to bioinformatics algorithms,* MIT Press, 2004.

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

Module Evaluation Course questionnaires, course review.