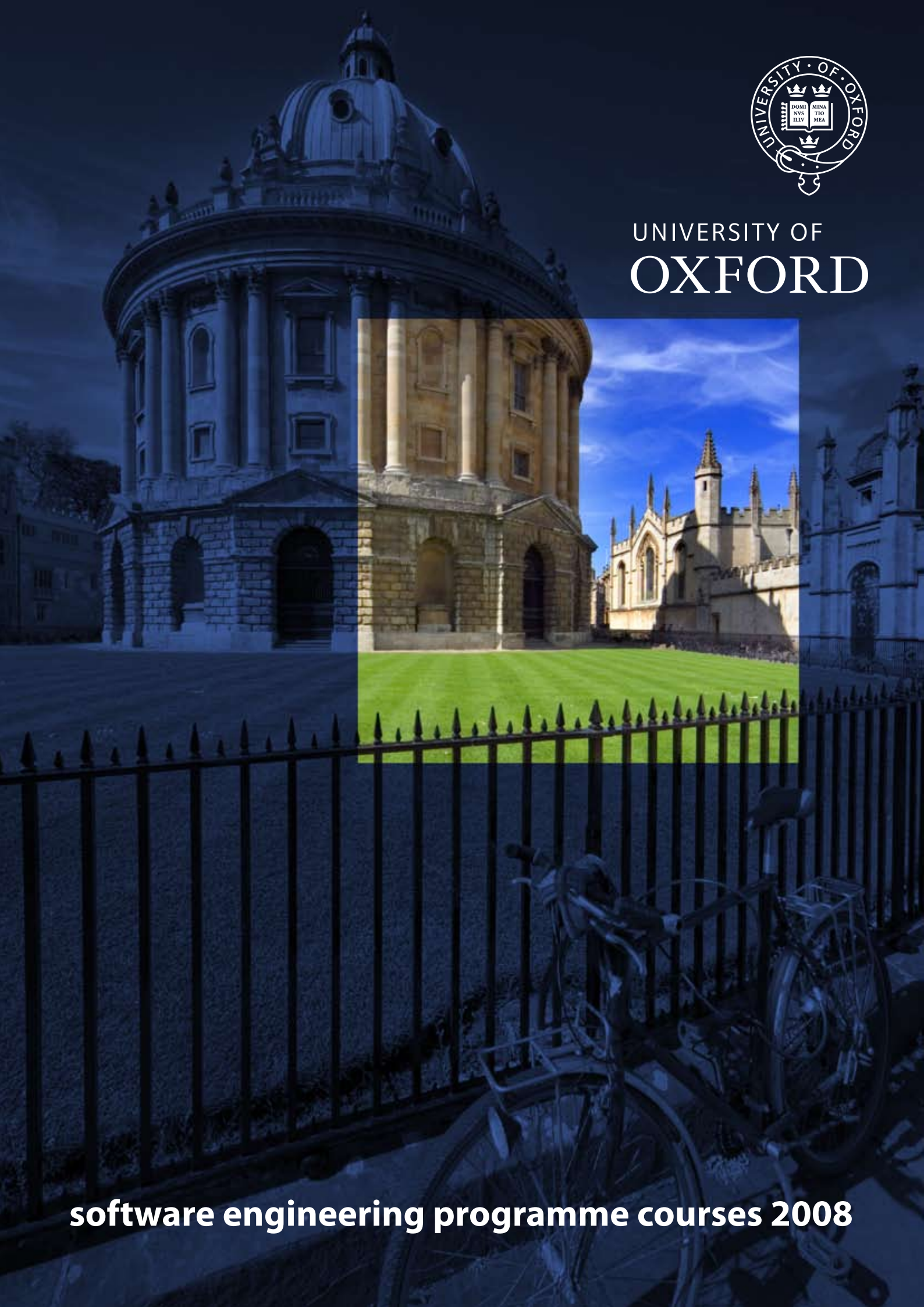


UNIVERSITY OF
OXFORD

software engineering programme courses 2008

Software Engineering at Oxford

Welcome to the *Software Engineering Programme*, a centre for advanced education and applied research at the University of Oxford. Established in 1993, the Programme exists to make the connection between software engineering theory and practice, and to make the expertise of the University available to those who wish to study while continuing in full-time employment.

Software Engineering

'Software Engineering' means the application of scientific and engineering principles to the development of software systems: principles of design, analysis, and management. The application of these principles makes it easier to develop software that meets its requirements, even when these requirements change; to complete the development on time, and within budget; and to produce something of lasting value, by being easy to maintain, re-use, and re-deploy.

An opportunity to learn

The Programme teaches the principles of modern software engineering, together with the tools, methods, and techniques that support their application. It offers working professionals an opportunity to learn more about the technological advances that are changing their lives, through a programme of part-time study at Master's level at one of the world's leading universities.

Courses and qualifications

The Programme offers a number of postgraduate qualifications — Postgraduate Certificates, a Postgraduate Diploma and a Master of Science. These qualifications can be obtained through attendance on courses in a variety of software engineering subjects, such as requirements engineering, security risk and threat analysis, service-oriented architectures, design patterns, software testing, formal techniques, and development management. The courses can also be taken separately, without registering for a qualification, as short programmes of advanced professional training.

"The separation of practical and theoretical work is artificial and injurious."
(Christopher Strachey)

Teaching quality

Each course is delivered by an expert in the subject, and includes an intensive teaching week of classes, lectures, and practical work, followed by a written assignment. This mode of study is particularly effective for those with significant professional or personal commitments. Courses take place at the Programme's purpose-built facilities in the centre of Oxford, and class sizes are kept small to allow for a high degree of interaction.

Intended audience

The Programme is intended for those with an awareness of software development issues: architects, programmers, managers, and informed customers. The entry requirements for the postgraduate qualifications reflect this: industrial experience is valued as highly as previous education. There are no formal entry requirements for the individual courses, although prospective attendees are asked to confirm that they satisfy the informal requirements for any course that they wish to attend.

Research activity

The Programme is also a centre for research activity. The teaching staff are involved in a number of national and international projects in the areas of: large-scale data integration and sharing, particularly regarding authorization; cancer clinical trials informatics; applications of trusted infrastructure technologies in distributed systems; languages and tools for object model transformation; virtual research environments; techniques for generic programming; and model-driven software engineering.

Postgraduate qualifications

All of the courses offered by the Programme are taught at Master's level: the five postgraduate qualifications differ only in focus and extent. All successful applicants for postgraduate study are registered initially for a Postgraduate Certificate. Students may ask to change their registration at any time; such a request will be granted provided that the student has made good progress towards the new qualification.

Postgraduate Certificate in Software Engineering (PGCSE)

This qualification requires attendance on courses in any four of the subjects offered by the Programme, and the successful completion of the corresponding written assignments. The courses and assignments must be completed within two years of the date of admission.

Postgraduate Certificate in Object Technology (PGCOT)

Like the PGCSE, this qualification requires attendance on courses in four subjects. The difference is that at least three of these subjects must be chosen from the Object Technology theme; the fourth subject may be any of those offered.

Postgraduate Certificate in Computer Security (PGCCS)

Like the PGCSE and PGCOT, this qualification requires attendance on courses in four subjects. At least three of these subjects must be chosen from the Computer Security theme; the fourth subject may be any of those offered.

Postgraduate Diploma in Software Engineering (PGDipSE)

This qualification requires courses in eight subjects. The subjects can be any of those offered by the Programme. The courses and assignments must be completed within three years of admission.

MSc in Software Engineering (MScSE)

This qualification requires courses in ten subjects; it also entails the completion of a project and dissertation, involving participation in two project weeks. The subjects can be freely chosen from the schedule. The courses and assignments should be completed within four years of admission; an additional year is available, if needed, in which to complete the project and dissertation.

Admission

Applications for part-time, postgraduate study are invited from anyone with sufficient experience or proven ability in the field of Software Engineering.

A typical applicant might have an undergraduate degree in a related subject, and at least two years' experience of software development in an industrial context. However, relevant experience may compensate for a lack of formal qualifications, or vice versa — concerned candidates should contact the Programme Office for further information.

The open nature of the entry requirements means that a formal interview is an essential part of the admissions process. In advance of the interview, applicants are asked to present two references from people who are familiar with their work or study achievements and — if appropriate — to secure the support of their employer.

Successful applicants will become registered students of the University with effect from the beginning of the next University term. New students who have already attended courses on the Programme can use those courses as credit towards their qualification, provided that the course dates are within one year of the date of admission.

Previous study

A student on the Programme may be able to use a course taken elsewhere as credit, provided that: the course was taught and assessed at the same level; the subject fits within the Programme curriculum; and the course is not used as credit towards any other qualification. For the Postgraduate Certificates, no more than one course taken elsewhere may be used; for the Postgraduate Diploma or the MSc, no more than two.

Accreditation

The MSc in Software Engineering is accredited by the British Computer Society: graduates from the Programme can use the MSc to gain exemption from the Professional Graduate Diploma and PGD Project. The individual courses can be used as credit in the IEE, IAP and BCS continuing professional development schemes, and have a CATS value of 15 points for credit transfer between postgraduate programmes.

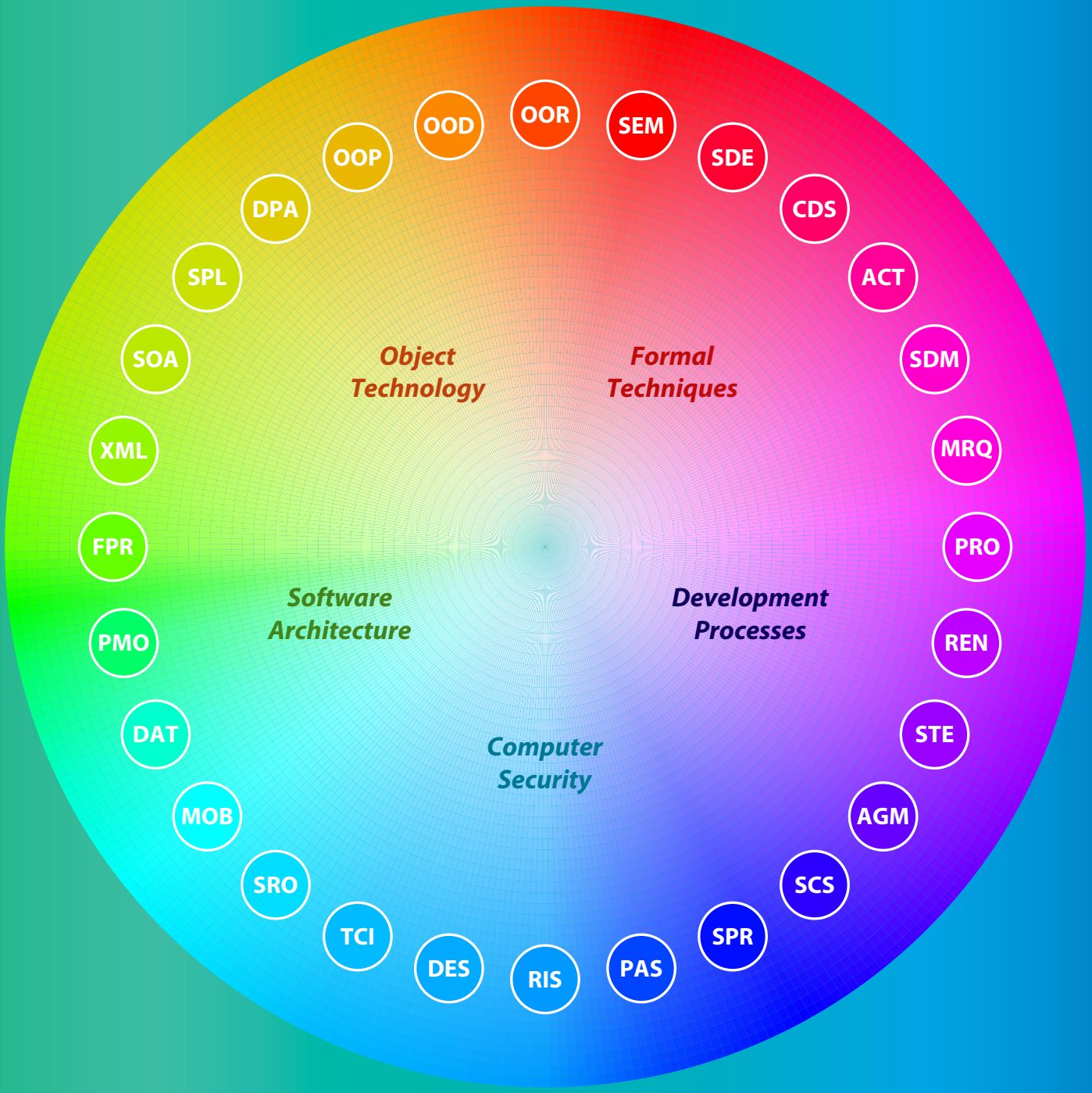
Examination

A student who has completed the required number of courses and written assignments may enter for examination as soon as the minimum period of study has elapsed: for a Postgraduate Certificate or Postgraduate Diploma, this means one year after first admission; for the MSc, at least two years after first admission, and at least one year after transferring registration to the MSc.

Each written assignment is treated as part of an examination at the University of Oxford. Assignment submissions and assessment reports are reviewed by a board of examiners — all subject specialists — before a grade is assigned, and a copy is returned to the student.

When a student enters the examination for a postgraduate qualification, the examiners consider their performance at the next formal meeting; there is a meeting at the end of each University term. They may also wish to interview the student, to ask about the work that they have submitted and the courses that they have attended.

There is a straightforward progression mechanism from the Postgraduate Certificate through the Postgraduate Diploma to the MSc: a student who has completed a lower award will be able to upgrade it, using the courses and assignments that they have completed as credit towards a higher award. Should a student fail to satisfy the examiners, they would be given an additional year in which to undertake further study.



Courses

Courses in each subject are offered according to demand; most subjects are taught at least twice a year. Class sizes are limited, so advanced booking is essential. Outline descriptions of the subjects are presented later in this brochure; more detailed descriptions can be found on the Programme website.

Course dates in 2008

14 Jan - 18 Jan	*SPR*	Security Principles
21 Jan - 25 Jan	*CDS*	Concurrency and Distr. Systems
21 Jan - 25 Jan	*OOR*	Object Orientation
18 Feb - 22 Feb	*XML*	Extensible Markup Language
25 Feb - 29 Feb	*PSE*	Practical Software Engineering
03 Mar - 07 Mar	*AGM*	Agile Methods
03 Mar - 07 Mar	*STE*	Software Testing
10 Mar - 14 Mar	*ACT*	Advanced Concurrency Tools
10 Mar - 14 Mar	*SDM*	Software Development Mgmt.
31 Mar - 04 Apr	*FPR*	Functional Programming
07 Apr - 11 Apr	*DES*	Design for Security
14 Apr - 18 Apr	*OOD*	Object-Oriented Design
21 Apr - 25 Apr	*OOR*	Object Orientation
21 Apr - 25 Apr	*SEM*	Software Engineering Maths
28 Apr - 02 May	*SPR*	Security Principles
12 May - 16 May	*DAT*	Database Design
12 May - 16 May	*REN*	Requirements Engineering
19 May - 23 May	*MRQ*	Managing Risk and Quality
19 May - 23 May	*PAS*	People and Security
02 Jun - 06 Jun	*XML*	Extensible Markup Language
09 Jun - 13 Jun	*RIS*	Security Risk Analysis & Mgmt.

16 Jun - 20 Jun	*SPL*	Software Product Lines
16 Jun - 20 Jun	*SDE*	Specification and Design
23 Jun - 27 Jun	*OOP*	Object-Oriented Programming
23 Jun - 27 Jun	*PSE*	Practical Software Engineering
07 Jul - 11 Jul	*PRO*	Process Quality & Improvement
14 Jul - 18 Jul	*OOD*	Object-Oriented Design
21 Jul - 25 Jul	*DPA*	Design Patterns
01 Sep - 05 Sep	*SDM*	Software Development Mgmt.
08 Sep - 12 Sep	*PW*	Project Week
08 Sep - 12 Sep	*PMO*	Performance Modelling
22 Sep - 26 Sep	*MOB*	Mobile and Sensor Networks
29 Sep - 03 Oct	*SPR*	Security Principles
29 Sep - 03 Oct	*SEM*	Software Engineering Maths
06 Oct - 10 Oct	*AGM*	Agile Methods
06 Oct - 10 Oct	*TCI*	Trusted Comp. Infrastructure
13 Oct - 17 Oct	*OOP*	Object-Oriented Programming
13 Oct - 17 Oct	*REN*	Requirements Engineering
20 Oct - 24 Oct	*DAT*	Database Design
27 Oct - 31 Oct	*XML*	Extensible Markup Language
03 Nov - 07 Nov	*MRQ*	Managing Risk and Quality
03 Nov - 07 Nov	*SCS*	Safety Critical Systems
10 Nov - 14 Nov	*STE*	Software Testing
17 Nov - 21 Nov	*CDS*	Concurrency and Distr. Systems
24 Nov - 28 Nov	*SOA*	Service-Oriented Architecture
01 Dec - 05 Dec	*DES*	Design for Security
01 Dec - 05 Dec	*OOD*	Object-Oriented Design
08 Dec - 12 Dec	*PSE*	Practical Software Engineering
08 Dec - 12 Dec	*SRO*	Secure & Robust Programming

Dates are correct at time of press, but may change; check the website for up to date information.

Formal Techniques

The Programme offers four courses on the theme of formal techniques: practical applications of mathematics, designed specifically for the task of modelling software systems. The courses cover: mathematical description and reasoning; static models of component states; dynamic models of component interactions; and analysis tools for process verification and validation.

Software Engineering Mathematics (SEM)

An important characteristic of a specification is the ability to reason about the objects it describes, and thus about the system it models. If that specification is written mathematically, the reasoning can take the form of calculation: we can use simple, logical methods — and tool support — to check that a specification is consistent, and to predict the consequences of our design decisions. This course is an introduction to mathematical description, using basic set theory and logic. It serves as an excellent foundation for other courses in the Formal Techniques theme.

Specification and Design (SDE)

We deal with complexity by adding structure and organisation to our specifications, identifying patterns, and finding appropriate abstractions. This course shows how to structure and organise specifications, using the pattern, or 'schema', language of the Z notation to produce descriptions of component functionality that are concise, precise, and comprehensible. It shows also how these descriptions can be analysed to check for consistency of requirements, to validate invariants or assertions, and to determine the preconditions of operations.

"The job of formal methods is to elucidate the assumptions upon which formal correctness depends."
(Tony Hoare)

The course assumes familiarity with the concepts and notations presented in *Software Engineering Mathematics (SEM)*. Participants who have not previously attended *SEM* are asked to confirm that they have sufficient experience before attending *SDE*.

Concurrency and Distributed Systems (CDS)

The consequences of design decisions are particularly hard to predict when a system consists of several concurrently-executing components, or when there is a complex pattern of interaction between a system and its environment. This course presents a powerful technique for describing the intended behaviour of concurrent systems, and for reasoning about the patterns of interaction that may emerge.

The technique is based upon the language of CSP (Communicating Sequential Processes): an economical, precise notation with practical theories of correctness, calculation, and refinement. No previous knowledge is assumed, although familiarity with the basic concepts of sets, sequences, and logic would be useful.

Advanced Concurrency Tools (ACT)

It is impossible to prove the correctness of a complex design without automated reasoning. One of the most powerful forms of automated reasoning is model-checking, in which every configuration of a given design is automatically explored and validated. This course presents a set of practical techniques for the analysis of patterns of interaction, and shows how the application of these techniques can be supported by model-checking tools.

The course assumes familiarity with the concepts and notations presented in *Concurrency and Distributed Systems (CDS)*. Participants who have not previously attended *CDS* are asked to confirm that they have sufficient experience before attending *ACT*.

Object Technology

Object orientation is a widely-used approach to solving the problems of software construction. It involves the encapsulation of functionality and associated data as objects, and the classification of those objects into a hierarchy of classes. This makes it easier to control complexity, and encourages the development of flexible, robust, and re-usable components. The Programme offers five different courses on the theme of object technology, addressing: essential concepts; programming principles; design techniques; patterns of structure and behaviour; and modularity across a family of software systems.

Object Orientation (OOR)

This is a general course in object orientation, offering both an introduction for beginners and a wider perspective for experienced users of object technology. It takes a 'big picture' approach to programming and design, rather than dwelling on language-specific details, and serves as a framework for the other courses in the Object Technology theme.

"A language that doesn't affect the way you think about programming isn't worth knowing."
(Alan Perlis)

No previous experience with object-oriented languages or tools is necessary, although some familiarity with basic programming concepts is expected. Programme staff will be happy to provide advice and preliminary study material to those who wish to attend the course, but who may lack sufficient programming experience.

Object-Oriented Programming (OOP)

This course teaches the concepts and principles of object-oriented programming. The language used is Java, although most of the material covered will apply equally well to any other object-oriented language: objects, methods, interfaces, messages, and events.

The course assumes familiarity with object-oriented concepts, and some knowledge of the notations of the Unified Modeling Language (UML). Participants who have not previously attended *Object Orientation (OOR)* are asked to confirm that they have sufficient experience before attending *OOP*.

Object-Oriented Design (OOD)

This course teaches standard techniques for the specification of software. The course is based around a carefully-chosen subset of the UML (Unified Modeling Language), and places the techniques in a formal software engineering context.

The course assumes familiarity with object-oriented concepts — such as classes, inheritance, and polymorphism — and some basic knowledge of the Java language. Participants who have not previously attended *Object Orientation (OOR)* are asked to confirm that they have sufficient experience before attending *OOD*.

Design Patterns (DPA)

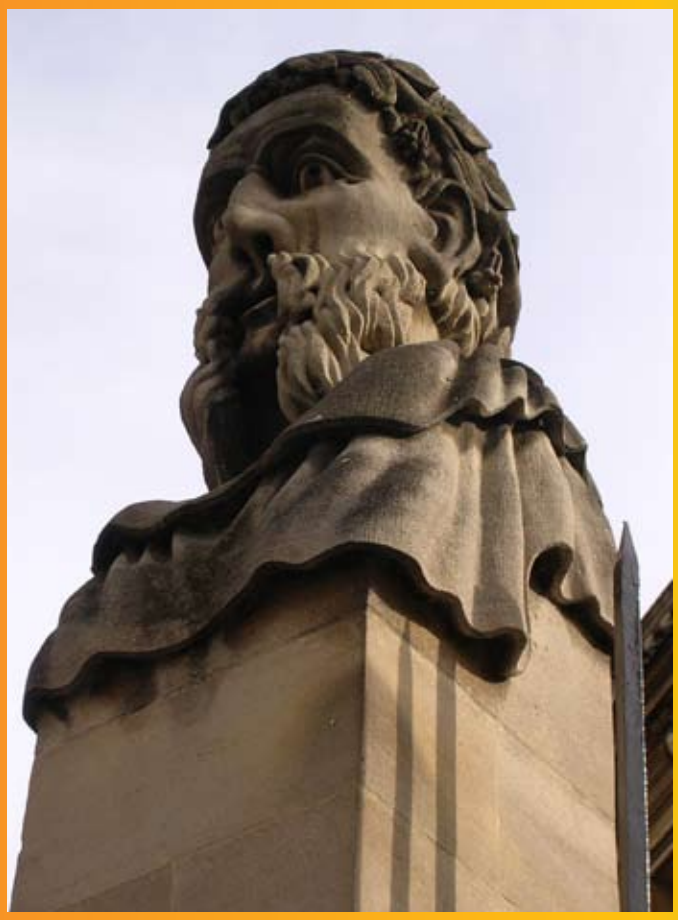
This is an advanced course in the structure and behaviour of object-oriented systems, covering both design and programming. It is based around the notion of a *design pattern*: an abstraction of a proven solution to a recurring problem in a specific context in system design.

This course will assume considerable experience in object-oriented design and programming; the courses *Object-Oriented Design (OOD)* and *Object-Oriented Programming (OOP)* are suitable preparation.

Software Product Lines (SPL)

Software Product Lines are an increasingly important trend in software development, systematically identifying and exploiting commonalities and variations among similar software systems. They have the potential to promote asset reuse throughout the software life cycle, reduce time to market, and ease product customization.

This course presents a thorough and comprehensive coverage of state-of-the-art practices in SPLs, from analysis and design to implementation. It assumes knowledge of Java and UML comparable to *Object-Oriented Programming (OOP)* and *Object Orientation (OOR)*, respectively.



Software Architecture

Modern software systems operate in a complex environment, interacting and interoperating with a wide variety of services and processes over which the designer of a single component may have little control. The successful development of such systems requires: an understanding of system architecture; languages and tools for the representation and interchange of data, metadata, and computation; and effective techniques for reasoning about performance.

The Programme offers six courses on the theme of software architecture, addressing: programs as data; metalanguages for information interchange; service-oriented architectures; relational database technology; mobile networks; and network performance modelling.

Functional Programming (FPR)

In functional programming, computations are modelled as expressions rather than actions. This offers significant opportunities for parameterisation and modularisation, beyond those available in conventional, imperative programming. It also results in the production of programs that are clearer, simpler, and often surprisingly concise. The techniques taught in this course will be useful in any language — particularly for the definition of transformations on structured data.

Programming experience in a traditional programming language is expected. In addition, some familiarity with basic mathematical concepts such as functions, predicates, and lists will be helpful; these concepts are taught from first principles in *Software Engineering Mathematics (SEM)*.

"A system is composed of components; a component is something you understand."
(Howard Aiken)

Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a language designed for the definition of document structures, and the production of structured documents. It can be used to define application-specific representations that are easy to process and transform, facilitating the interchange of information between different systems and components. This course teaches the essentials of the language, document validation, the creation of stylesheets, and the use of core XML technologies to solve software engineering problems.

Service-Oriented Architecture (SOA)

The current consensus on best practice for building component-based distributed applications is to use a *service-oriented architecture*. Services are encapsulated behind carefully-designed simple interfaces, and the realities of heterogeneity, decentralization and fault tolerance are embraced rather than ignored. The course provides an understanding of the strengths and weaknesses of service orientation, informed by an ability to implement and deploy simple web services using a suitable development platform.

The course assumes familiarity with the XML language, and with the basic concepts of object-oriented programming, to a level equivalent to the courses *Extensible Markup Language (XML)* and *Object-Oriented Programming (OOP)* respectively.

Database Design (DAT)

Relational database technology is the dominant approach to information storage, with products that offer an unmatched combination of abstraction and performance. To use these products effectively, however, requires an understanding of the underlying principles and concepts: relational modelling, normalisation, query optimisation, transactions, and distribution. This course covers these fundamentals, rather than providing a working knowledge of specific products and proprietary solutions.

This course assumes some familiarity with basic mathematical concepts such as sets, predicates, and relations. These concepts are taught from first principles in *Software Engineering Mathematics (SEM)*.

Mobile and Sensor Networks (MOB)

Recent advances in wireless mobile and sensor technologies have changed computing, enabling application scenarios in which large numbers of pervasive computing devices are connected to a wireless networking infrastructure in an ad hoc manner. This course covers communication protocols for mobile ad hoc networks, and provides an overview of distributed data management techniques for resource-constrained sensor networks.

Performance Modelling (PMO)

This course shows how simple techniques from continuous mathematics can be used to predict the behaviour of networks and systems. It presents basic formulae and theorems from the theories of probability and random processes, and shows how these may be applied in software and communications performance modelling. It explains how to predict congestion, calculate expected loads, and develop strategies for integrated services.

Familiarity with basic notions of probability are assumed, along with an ability to manipulate linear equations and familiarity with exponential functions. Basic calculus would be helpful but is not essential.

Computer Security

As computing systems become more essential to our daily lives, it becomes ever more important that the services they provide are available whenever we need them. We must also be able to rely on the integrity of the systems, and thus the information that they hold and provide. What is more, our society and our economy depend upon certain pieces of information being held in confidence. The Programme offers six courses on the theme of computer security, covering different aspects of availability, integrity, and confidentiality. There is also a related course on safety-critical systems in the Development Processes theme.

Security Principles (SPR)

This course combines a treatment of the fundamental principles of cryptography and security protocols with a practical treatment of current best practice. It explains the need for computer security, and the scope of the available technical solutions; presents techniques for evaluating security solutions; and provides an overview of the current leading technologies and standards in the security arena.

People and Security (PAS)

A very high proportion of failures in security can be attributed to human weakness, misunderstanding, misinformation, or failure to grasp the importance of the processes individuals are expected to follow. This course draws on work from human-computer interaction, and more widely from psychology, relating these issues back to hard technical implementation decisions.

"Security is not a product; it itself is a process."
(Bruce Schneier)

Familiarity with basic security principles and standard mechanisms, as covered in *Security Principles (SPR)*, is assumed.

Security Risk Analysis and Management (RIS)

Security is a property of an entire system in context, rather than of a software product, so a thorough understanding of system security risk analysis is necessary for a successful project. This course introduces the basic concepts and techniques of security risk analysis, and explains how to manage security risks through the project lifecycle.

Participants should have a basic understanding of topics in security, as provided by the *Security Principles (SPR)* course.

Trusted Computing Infrastructure (TCI)

A secure system is the product of numerous layers that operate together to provide in-depth protection. This course looks at the various platforms upon which a secure system operates, with an emphasis on practical and repeatable means of implementing these platforms securely. Topics covered include buffer overflows, cryptographic libraries, sand-boxing, virtualisation, trusted computing, and database security, building towards a toolkit of sound principles for secure systems implementation.

Participants should have a basic understanding of topics in security, as provided by the *Security Principles (SPR)* course.

Design for Security (DES)

Capability in the design of systems which will meet security goals is an increasingly important skill. This course explores how cost-effective solutions to security needs can be achieved by following well-established architectural practices and detailed security principles. Central to these considerations is the need for requirements to be met with established solutions, and how a balance can be struck between security and other system requirements.

Participants should have a basic understanding of topics in security, as provided by the *Security Principles (SPR)* course.

Secure and Robust Programming (SRO)

Many system failures and security vulnerabilities arise at the programming level. These can often be attributed to inadequate handling of exceptional situations, poor understanding of the details of the programming language in use, incomplete descriptions of the interfaces between components, and insufficient care in the treatment of concurrency and threading issues. This course addresses those problems from a programming perspective, with the aim of improving the practitioner's capability in writing and reviewing code.

Because of the discrete mathematics needed to understand and apply these ideas, prior attendance on the *Software Engineering Mathematics* course is advisable. Participants should also have a good, detailed understanding of programming, to the level offered by the *Object-Oriented Programming*.

Bodleian Library, University of Oxford

Development Processes

There is more to software development than the purely technical aspects of system design and implementation, programming languages and data formats. Successful development projects depend as much on human, process and managerial considerations as on technical ones.

The Programme offers seven courses on the theme of development processes, addressing this wider view of software development: managing people, projects, plans and processes; lightweight development processes; managing risk and quality; standardised processes such as the Capability Maturity Model; eliciting system requirements from people; testing techniques for measuring the success of a system design or implementation; and industry standards for safety-critical development.

Software Development Management (SDM)

Management is an essential element of successful software development. This course gives an introduction to the fundamental aspects of project and people management that are required to successfully manage a software development project.

"Adding manpower to a late software project makes it later."
(Fred Brooks)

While no experience of project management or team leadership is necessary, some experience of working in a project based software environment is highly desirable.

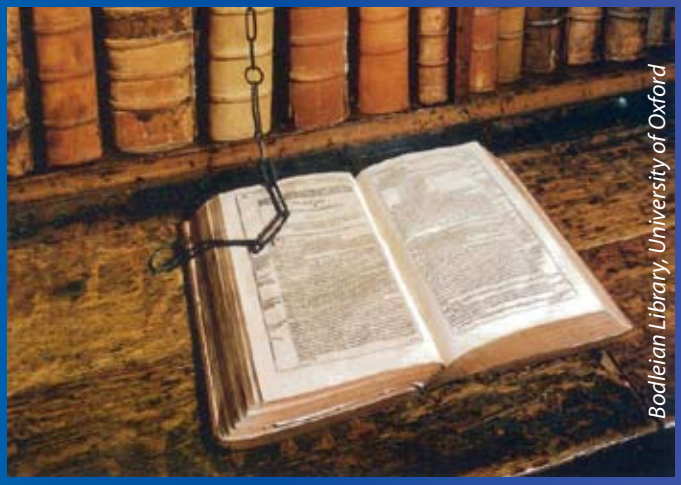
Agile Methods (AGM)

Agile methods are challenging conventional wisdom regarding systems development processes and practices, effectively 'putting process on a diet' and investing in people and teams. This course enables today's software development professional to understand the heart of agility, and covers both the theory and practice of agile methods such as XP, Scrum, Crystal, FDD, Lean and DSDM.

Management of Risk and Quality (MRQ)

Too many project planning approaches concentrate on just the estimating and network aspects of planning. This is of little value if the project is given the wrong shape or the wrong activities are chosen in the first place. The *Strada method* taught in this course builds the project from an analysis of the specific risks to be faced. It then uses an analysis of the specific quality requirements to fill out the detail. The two perspectives of risk and quality prove sufficient to give the basis for a reliable and robust plan.

Experience of working in a software development environment, preferably with management responsibility, is desirable.


Bodleian Library, University of Oxford

Process Quality and Improvement (PRO)

Every software development organization needs to be focused on the delivery of quality. The software engineering discipline responds by calling for a managed process for the construction and testing of software, and for the improvement of that process. This course explains the necessary concepts within the frameworks provided by three important international standards.

Experience of working in a software development environment is desirable. It would be beneficial (but not necessary) to have some experience of software project management.

Requirements Engineering (REN)

Establishing firm and precise requirements is an essential component of successful software development. Although many aspects of the requirements process involve the resolution of purely technical issues, these are often the least problematic; successful analysis requires expertise and investigations of a broader nature, addressing the human context of current and future work practices. This course covers a range of methods from 'hard' semi-formal approaches, to 'softer' people-oriented ones.

Previous knowledge of requirements is not necessary, but experience in some aspect of software design is desirable.

Software Testing (STE)

This course presents realistic, pragmatic steps for rigorous and organized software testing. It clarifies testing terminology and covers the different types of testing performed at each phase of the software lifecycle, together with the issues involved in these types of testing. The course discusses how tests can be derived from requirements and specifications, design artifacts, or the source code, and introduces appropriate testing tools with hands-on exercises.

There are no formal prerequisites for this course, but familiarity with programming in an imperative or object-oriented language will be very beneficial.

Safety Critical Systems (SCS)

Computers are often placed in control situations within safety-critical systems. Safety is an emergent property of whole systems; software may play only a small part. This course will enable the systems engineer to determine whether a safe system can be built, and what requirements must be placed on software in order to keep risk at an acceptable level.

Background

The Programme is the result of a collaboration, established in 1992, between Oxford University Computing Laboratory (OUCL) and Oxford University Department for Continuing Education (OUDCE). OUCL has an international reputation for linking mathematical theory to industrial practice. OUDCE has a distinguished history in promoting life-long learning, including delivering high-quality education to working professionals.

The discipline of software engineering has changed considerably since the early 90s: although the principles remain the same, the context in which they are applied is evolving. As a result, the Programme is in a state of constant development, placing greater emphasis on different principles, and finding new ways to relate them to industrial practice.

Programme staff

There are eighteen core members of staff, all of whom work exclusively for the Programme: *Alessandra Cavarra*, Lecturer; *Andrew Cooper,* Teaching Assistant; *Edward Crichton*, Researcher; *Jim Davies*, Professor and Director; *Melissa Endacott*, Administrator; *Ivan Flechais*, Lecturer; *Jeremy Gibbons*, Reader and Deputy Director; *Ralf Hinze*, Lecturer; *Jackie Jordan*, Manager; *Eric Kerfoot*, Teaching Assistant; *Roberto Lopez-Herrejon,* Teaching Assistant; *Andrew Martin*, Lecturer and Deputy Director; *Steve McKeever*, Lecturer; *Shirley Sardar*, Administrator; *Clint Sieunarine*, Teaching Assistant; *Andrew Simpson*, Lecturer; *Niki Trigoni*, Lecturer; *Chen-Wei Wang*, Teaching Assistant.

Subject specialists

There are also a number of subject specialists who teach courses in their particular areas of expertise: *Paul Beaven*, Independent Consultant; *Rob Collins*, Director, Entelechia Ltd; *Marina Jirotka*, University Lecturer, Oxford; *Nigel Kermode*, Independent Consultant; *Robert Leese*, Director, Smith Institute; *Angela Martin*, Independent Consultant; *Bill Roscoe*, Professor, Oxford; *Angela Sasse*, Professor, London; *Mark Slaymaker*, Researcher, Oxford; *Pete Verey*, Programme Consultant, Oxford.

Organisations

The Programme benefits greatly, in terms of advice and sponsorship, from a number of leading companies and organisations: EPSRC, IBM, Motorola, Marconi, QinetiQ, SEEDA, the Smith Institute, and Symbian. Representatives from most of these organisations sit on the advisory panel for the Programme, helping to ensure that its teaching reflects and anticipates industrial needs.

The students come from a wide range of organisations: software companies; healthcare providers; finance houses; government agencies; bioinformatics start-ups; management consultants; power companies; car manufacturers. A significant number are self-employed, or work for small enterprises. They share an awareness of software engineering practice, and a desire to learn more about the underlying principles.

Kellogg College

A student requesting a change to MSc registration will be required to complete a college application form. If the request is appropriate, it will be forwarded to the University's Graduate Studies Office, who will deal with the college in question. If the application is approved, then the student will be required to matriculate, normally during the next University term. The student then becomes a member of their chosen college.

There are many different colleges, each with its own style and tradition, and one of them is closely associated with part-time study. It was named in honour of William Keith Kellogg, industrialist and philanthropist, in recognition of the support given to adult and continuing education by the W. K. Kellogg Foundation.

Kellogg has approximately ninety fellows (including all of the Programme's University Lecturers) and approximately four hundred registered students. Most of the Programme's students who progress to the MSc apply to Kellogg. The College organises regular events, and acts as an additional point of contact between MSc students and the University.

Taking part

To reserve a place on a course without first becoming a registered student, you should complete the course booking form. If a place is available, you will receive a confirmation from the Programme Office. For courses in certain subjects — *ACT*, *DES*, *DPA*, *OOD*, *OOP*, *PAS*, *RIS*, *SDE*, *SOA*, *SPL*, *SRO* and *TCI* — you may be asked to confirm that you have sufficient experience before your reservation can be approved.

Students who are registered for a postgraduate qualification do not need to submit booking forms for individual courses: they can make reservation requests via the Programme website.

Postgraduate study

To apply to study for a postgraduate qualification, you should follow the procedure on the Programme website, or request an information pack from the Programme Office. Either route provides instructions on what supporting documentation you should supply.

Only those applicants with an appropriate combination of education and experience will be called to interview. Applicants should not make travel plans, or accept contingent financial (or other) commitments, unless a formal, written offer of a place has been made. The allocation of a place on an individual course does not imply admission to a postgraduate qualification.

Fees

There is a standard fee of £1350 for each course attended, payable in advance. This includes course materials, and lunches during the teaching week, but not accommodation. The fee applies whether or not the attendee is working towards a postgraduate qualification, and regardless of nationality and residency. The Postgraduate Certificate will typically entail four course attendance fees, the Postgraduate Diploma eight, and the MSc ten.

Students may cancel attendance, provided that the cancellation is received well enough in advance. Cancellations at short notice may not receive a full refund of the course attendance fee.

The examination and assessment of the course assignment is included in the course attendance fee. Should a student be granted permission to take the assignment for a later course in the same subject, an additional examination fee of £100 will apply.

There is an annual award fee of £2500 for any of the postgraduate qualifications; this is payable for one year for a Postgraduate Certificate, two years for a Postgraduate Diploma, and four years for an MSc. Students who are citizens of a member state of the European Community, and have been ordinarily resident in the European Economic Area for the past three years, may qualify for Home/EU status, and a reduction of this annual award fee to £1250.

All payments must be completed before any postgraduate qualification will be awarded by the University. Payments made towards one qualification will be counted towards the total amount due should the student subsequently register for another.

The above rates apply for course attendance and student registration between October 2007 and September 2008; the fees will change in October 2008. Award fees are based upon the date of admission. Attendance fees are based upon the date of the course, and may increase during the period of study, typically in line with the rate of inflation in the UK.

Contact

Enquiries should be addressed to the Programme Office:

Software Engineering Programme
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
+44 1865 283525 (phone), 283531 (fax)

The most effective means of contact is email:

info@softeng.ox.ac.uk

Further information about the Programme, together with course booking and application forms, can be found on the Programme website:

www.softeng.ox.ac.uk/about