# Changes to Year 3 courses – Case for Support

M Rovatsos

## 1. Summary of proposed changes

Changes requiring Board of Studies in **bold** print, others for information only.

<table>
<thead>
<tr>
<th>Course</th>
<th>Changes</th>
</tr>
</thead>
</table>
| **ABS** | • SCQF Credit Level: Discontinue level 9, open up Level 10  
• Normal Year Taken: Year 3 Undergraduate  
• Study Pattern: Tutorials – 8 hours  
• Available to Visiting Students: Yes  
• Pre-Requisite Courses: Informatics 2D or new AI Foundations course if approved  
• Co-Requisites: Remove ABS Level 9  
• Other Requirements: Update text to ‘Successful completion of Year 2 of an Informatics Single or Combined Degree, or equivalent by permission of the School’ |
| **ADS** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate  
• Minor changes to descriptor (attached) |
| **COMN** | • Move to Semester 1  
• Minor changes to descriptor (tbc) |
| **CT** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate  
• Minor changes to descriptor (attached) |
| **CCS** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate |
| **CAR** | • Minor changes to descriptor (attached) |
| **CS** | • Remove “mathematical ability” and Java requirement |
| **CT** | • Minor changes to descriptor (tbc) |
| **DBS** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate  
• Addition of tutorials  
• Minor changes to descriptor (attached) |
| **FNLP** | • Should include Relevant Curriculum QAA sections: Artificial Intelligence, Natural Language Computing |
| **LP** | • Minor changes to descriptor (tbc) |
| **LSI** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate |
| **OS** | • Study Pattern: Coursework assessed for credit – 25  
• Check where 'experience ....' Needs to be included (tbc) |
| **SEOC** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate  
• Other Requirements: "Students will be assumed to be able to read code examples in Java, and to be conceptually familiar with programming in a statically-typed object oriented language."  
• "Relevant QAA Computing Curriculum Sections: Software Engineering” |
| **ST** | • SCQF Credit Level: Amend to level 10  
• Normal Year Taken: Year 3 Undergraduate |
2. Case for Level 10 re-classification of current Level 9 courses
We have recently conducted a thorough review of all Year 3 courses in consultation with current Course Lecturers, Degree Programme Coordinators, Year Organisers, and Convenors of Examination Boards. This showed that a number of our Year 3 courses are should be more appropriately classified as Level 10 due to their level of difficulty, according to the following differences between the SCQF learning outcomes for the two levels:

1. **Detailed** knowledge and understanding *(rather than detailed only in some areas)* in one or more specialisms, some of which is informed by or at the forefront of a subject/discipline
2. **Critically** identify, define, conceptualise, and analyse complex/professional level *(rather than routine)* problems and issues
3. Use a **range of** the principal skills *(rather than a selection of these)*, practices and/or materials associated with a subject/discipline

A beneficial side effect of this change will be that these courses become available to Year 4 students. This will provide more flexibility in terms of combining courses across different years in terms of possible timetable clashes and semester load balancing. It will also enable students who have spent a year abroad during Year 3 to take courses not available abroad in Year 4, including courses that are prerequisite for other Level 10 and 11 courses.

We propose to amend the SCQF level for the following courses from 9 to 10 (all to remain advertised as “Normal Year Taken: Year 3 Undergraduate”). Each change is justified by making reference to the above sets of learning outcomes and comparison to other courses within the School:

**Algorithms and Data Structures**
This course requires advanced mathematical skills and deals with algorithms that are comparable to those taught in Level 10 courses such as, for example, Computer Algebra or Computer Graphics. In this course, students acquire an in-depth understanding of algorithmic thinking, the ability to analyse complex abstract problems, and a range of algorithm design and analysis techniques.

**Language Semantics and Implementation**
This course teaches advanced skills in using complex formal representations to model the semantics of programming languages and analyse their properties that are similar to those taught in Level 10 courses such as Communication and Concurrency, Performance Modelling, and Automated Reasoning. This is a theoretical course that teaches a range of highly abstract methods for describing and analysing programming languages which require a deep understanding of computation.

**Compiling Techniques**
This course combines a broad range of different techniques from computer science, from theoretical and conceptual models of programming languages and their analysis to computer systems architecture. It can easily be compared with Level 10 courses such as Distributed Systems in terms of difficulty. Compared to many other courses, the breadth of knowledge and understanding of computational techniques conveyed by the course significantly exceeds that of normal Level 9 courses. Practical work undertaken by students as part of the
course involves solving the complex task of mapping abstract languages to concrete computer hardware.

**Computational Cognitive Science**  
The higher level of difficulty of this course is mostly justified by its interdisciplinary nature: it combines an understanding of a broad range of literature from various non-Informatics disciplines with mathematical modelling and programming with systems that simulate cognitive architectures. Its learning outcomes are closer to a Level 10 profile due to the skills taught that enable students to critical review and compare various models of human cognition, combining computational thinking with empirical research on human and animal cognition.

**Database Systems**  
This course involves two types of skills: the ability to develop and understand abstract algebraic models of databases and their operations, and the ability to use database systems, programming and query languages. This combination is typical of many of our Level 10 courses, e.g. Embedded Systems, Distributed Systems, Parallel Architectures. The learning outcomes of the course involve an ability to design, build and use complex databases using a range of skills that are very different from standard programming skills taught in lower years.

**Software Engineering with Objects and Components**  
This course teaches object-oriented and component-based analysis and design and a variety of tools used in within this field. Its level of difficulty is roughly comparable to that of Level 10/11 software engineering course Software Architecture, Processes and Management. The abstraction and modelling capabilities required to develop coherent object-oriented and component-based software systems require advanced computational thinking skills.

**Software Testing**  
This course requires a level of maturity regarding the understanding of software systems comparable to that of Software Architecture, Processes and Management, a Level 10/11 software engineering course. Its syllabus involves understanding non-functional (such as human) aspects of software quality, as well as practical testing of real software systems. Its learning outcomes involve an ability to critically evaluate complex software systems based on rigorous testing methods.
Appendix A. Database Systems

Case for provision of tutorials and minor descriptor changes

Students have traditionally asked for tutorials for this course, as it introduces an entirely new area and implies a different way of thinking. It focuses on data modelling; the distinction between conceptual and physical data design; and declarative processing. This is in contrast to the implicit data models enforced by imperative and object-oriented languages students are more familiar with. Tutorials will help bridge the gap between the two ways of thinking and better prepare students for their coursework.

Additionally, note that if the course becomes a Level-10 course, then this means that it will be open to MSc students. It can therefore potentially be seen as a substitute to the now suspended Applied Databases, which dealt with largely similar topics and was one of the popular courses of the MSc programme when it was running. Coupled with the database specialism being a popular specialism of the MSc, then this means that the cohort may increase substantially. Moreover, it will be more diverse as some of the MSc students do not have a strong programming background. Tutorials will then be even more beneficial.

We would therefore request that the Board considers introducing tutorials for the course if there are available resources.

Updates to descriptor:

Summary of Intended Learning Outcomes:
1. Demonstrate knowledge of relational database schema design by employing the concepts of normal forms and entity-relationship diagrams.
2. Demonstrate the ability to declare, define, and access relational database schemas in relational database management systems using the data definition language subset of SQL.
3. Demonstrate knowledge of querying relational databases by using SQL and the ability to convert relational algebra queries to SQL.
4. Demonstrate knowledge of query evaluation by describing and implementing database processing algorithms.
5. Demonstrate the ability to programmatically map other data models to the relational one.
6. Demonstrate knowledge of concurrency control by describing transactional semantics.
7. Apply transactional semantics to reason about the correctness and consistency of database interaction among multiple users.
8. Demonstrate knowledge of recent advances in database systems by identifying the connection between traditional relational databases and other data models and/or query languages.
9. Demonstrate a basic knowledge of serial data expressed in XML.
10. Demonstrate understanding and the ability to implement tools for structuring and querying XML.

Assessment Information
Written Examination: 75
Assessed Assignments: 25
Oral Presentations: 0

Assessment
Three exercises, two small pen-and-paper ones (each worth 5% of the final mark) and a larger programming one (worth 15% of the final mark).
Appendix B. Computer Architecture

Changes to course descriptor

Course Description
Computer architecture is about optimising the design of computer hardware and software under constraints of time, cost and power consumption. Over the years, improvements in technology and advances in computer architecture have resulted in huge increases in computer performance. This course examines the fundamentals of high-performance computer architecture and looks at how the interface between hardware and software (architecture and compiler) influences performance.

Summary of Intended Learning Outcomes
1 - Demonstrate the ability to describe the structure and operational characteristics of a pipelined microprocessor.
2 - Demonstrate the ability to explain principles of: orthogonal instruction set design; pipeline hazards and interlocks; out of order execution; scoreboards and reservation stations and their use; branch prediction (both static and dynamic); and techniques (both software and hardware) for exploiting loop-level parallelism.
3 - Demonstrate the ability to quantitatively evaluate the performance of a combined processor and memory system with respect to cycles-per-instruction (CPI) and memory bandwidth requirements.
4 - Demonstrate the ability to explain the principle of memory locality, to show how a memory hierarchy exploits the various forms of locality, and to analyze the performance of a memory hierarchy.
5 - Demonstrate the ability to design, in outline, a memory hierarchy, and to specify reasonable parameters for each configuration point (capacity, associativity, block size, and write policies) at each level in the hierarchy.
6 - Demonstrate an understanding of the memory coherency issues involved when designing a multiprocessor system, and to explain the behaviour of a typical cache coherency protocol.

Study Pattern
Lectures 18
Tutorials 8
Timetabled Laboratories 0
Non-timetabled assessed assignments 25
Private Study/Other 49
Total 100

Reading list
Appendix C. Algorithms and Data Structures

**Standard (non-VUG) version**

**Pre-requisites:**
Students must have passed "Informatics 2B (INFR08009)" AND one of the following sets of Maths courses: (DMMR (INFR08023) AND "Probability with Applications (MATH08067)") OR the specialist 2nd year Maths courses for a BSc in Maths.

**Co-requisites and prohibited combinations:**
empty

**Other requirements:**
Delete old version for non-VUG. New text will be "Students who did not take Inf2B should get special permission from the course lecturer. Students who took a different collection of Maths courses should get permission from the Course lecturer."

**Assessment information:**
Delete sentence "If delivered in semester 1, ....", this is not relevant for the standard version of the course.

**Transferable skills:**
Add “Rigorous mathematical reasoning.”

**VUG version:**

**Pre-requisites:**
delete current text, as it is not relevant for visiting students. Instead copy over the paragraph starting "This course has the following mathematical ...." from "Other requirements".

**Add an item 7. also that says**
7. Simple sorting algorithms (MergeSort, Quicksort), Heaps and Priority Queues.

**Co-requisites and other requirements:**
empty

**Prohibited combinations:**
Add "Should not have previously taken any course which covered 1/3 of more of the syllabus."

**Assessment information:**
Change sentence "If delivered in semester 1, ...." to instead say "When delivered in semester 1, the visiting undergraduate students will have the written examination prior to the end of the calendar year. " Rest of assessment information stays the same.

**Transferable skills:**
Add “Rigorous mathematical reasoning.”
Appendix D. Compiling Techniques

Course description:
This course describes the phases of a modern programming language compiler with an emphasis on widely-used techniques. The course project will require students to implement a complete compiler for a simple educational programming language, targeting an abstract machine such as the JVM.

Summary of Intended Learning Outcomes
1 - Understanding of the challenges involved in compilation (semantic gap between input and output languages, compiler efficiency and code quality)
2 - Understanding of the phases involved in compilation, and knowledge of the techniques applied.
3 - Ability to understand design decisions in modern compilers and to justify these.
4 - Ability to develop and apply modifications to standard compilation techniques wherever this is necessary.
5 - Ability to analyse compilation tasks and to apply standard compilation techniques.
6 - Ability to implement standard compilation algorithms.
7 - Gain an understanding of the challenges involved in compilation for modern architectures and the approaches taken in modern compilers

Syllabus
* Introduction: structure of a compiler
* Lexical analysis: tokens, regular expressions, Lex
* Parsing: context-free grammars, predictive and LR parsing, Yacc
* Abstract syntax: semantic actions, abstract parse trees
* Semantic analysis: symbol tables, bindings, type-checking
* Stack frames: representation and abstraction
* Intermediate code: representation trees, translation
* Basic blocks and traces: canonical trees and conditional branches
* Instruction selection: algorithms for selection, RISC and CISC
* Liveness analysis: solution of dataflow equations
* Register allocation: colouring by simplification, coalescing
* Advanced Topics: automatic parallelisation, popular open-source compilers: GCC, LLVM

Relevant QAA Computing Curriculum Sections: Compilers and Syntax Directed Tools

Assessment Information:
Written Examination 75%
Assessed Assignments 25%
Oral Presentations 0%

Assessment
Two practical compiler exercises (Part 1 on the compiler front-end and IR generation, part 2 on semantic analysis and code generation)

If delivered in semester 1, this course will have an option for semester 1 only visiting undergraduate students, providing assessment prior to the end of the calendar year.