Appendix A:

Course Outlines for New Courses in Program

Course Title: Discrete Structures in Computer Science

Pre-Requisites: None

Year and Semester: Year 1, Semester 1

- **Course Description and Content Outline**(by topic)**:**

Provides a foundation in finite mathematics which is required for advanced computer science courses. For example an ability to derive and comprehend a formal proof is essential in formal specification and cryptography. Set theory concepts are used in software engineering and image processing.

A. Functions, relations and sets
    a. Introduction to functions (surjections, injections, inverses, composition)
    b. Growth of functions, 'big O' notation, 'big Theta' notation
    c. Introduction to sets (Venn diagrams, complements, Cartesian products, power sets)
    d. Introduction to relations (reflexivity, symmetry, transitivity, equivalence relations)
    e. Cardinality and countability

B. Basic logic
    a. Predicate and prepositional logic
    b. Logical connectives
    c. Truth tables
    d. Validity
    e. Normal forms

C. Proof techniques
    a. Notions of implications, converse, inverse, contrapositive, negation and contradiction
    b. The structure of formals proofs
    c. Direct proofs
    d. Types of proof (counterexample, contraposition, contradiction)
    e. Mathematical induction
    f. Recursive mathematical definitions

D. Counting
    a. Counting arguments
    b. Permutations and combinations, pigeonhole principle
    c. Elementary recurrence relations

E. Graphs and trees
    a. Trees
    b. Directed and undirected graphs
    c. Spanning trees
    d. Traversal strategies

F. Discrete probability
    a. Finite probability space, probability measure, events
    b. Conditional probability, Bayes' rule

- **Methods of Delivery:** 3 hours of lecture per week and a 2-hour weekly workshop incorporating a web-based tutorial.

- **Student Evaluation (typical)**:

    - Assignments solving sample problems (30%).
    - Mid-term examination (25%).
    - Final examination (45%)

- **Resources to be purchased/provided by students:** Textbook, reference sources and internet access (see Textbook requirements below).

- **Textbook requirements:**

    - Gersting, J.L., 2002, Mathematical Structures for Computer Science (5^{th} ed.), W.H.Freeman & Co (ISBN: 0716743582)

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: Illustrate by example the basic terminology and operations associated with functions, relations and sets.
Outcome 2: Relate practical examples to the appropriate set, function or relation model.
Outcome 3: Manipulate formal methods of symbolic prepositional and predicate logic.
Outcome 4: Demonstrate knowledge of formal logic proofs and logical reasoning through problem solving.
Outcome 5: Outline basic proofs for theorems using the techniques of proof by contradiction and mathematical induction.
Outcome 6: Relate the ideas of mathematical induction to recursion and recursively defined structures.
Outcome 7: Compute permutations and combinations of a set.
Outcome 8: Solve a variety of simple recurrence equations.
Outcome 9: Illustrate by example the basic terminology of graph theory.

Outcome 10: Model problems in application areas using graphs and trees.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

The lecture component of the class will be augmented by collaborative/active learning: the class will include some individual and group activities.

Information about Course Designer/ Developer
Course designed by Dr. Michael Wirth (Dept. Computing and Information Science, University of Guelph)

List faculty eligible to teach the course: Faculty to be hired

Are there any plans to teach all or portions of this course on-line? A course website will play an integral role in the delivery of resources for this course: syllabus, schedule, assignments, solutions to homework and exams, handouts, supplementary notes, etc.

Faculty Qualifications to teach/supervise the course: Postgraduate degree with experience in discrete structures.

Classroom requirements: Technology-enhanced classroom with networking, internet and data projection capabilities.

Equipment requirements: None

Course Title: Computer Architecture I

Pre-Requisites: Discrete Structures in Computer Science, Principles of Computer Science (corequisite)

Year and Semester: Year 2, Semester 1

- **Course Description and Content Outline**(by topic)**:**

  Description

  This course introduces the basic ideas of computer organization and underlying digital logic that implements a computer system. Starting from representation of information, the course looks at logic elements used for storing and processing information. The course also discusses how the information storage and processing elements are hooked together to function as a computer system.

  Course outline

  - Data representation: Number systems: binary, decimal, and hexadecimal number systems; bits, bytes, and words; numeric data representation ; fixed- and floating-point systems; signed and two's-complement representations; representation of nonnumeric data (character codes, graphical data); representation of records and arrays
  - Digital logic: Fundamental building blocks (logic gates, flip-flops, counters, registers, programmable logic devices; logic expressions, minimization, sum of product and product of sums forms; register transfer notation; physical considerations (gate delays, fan-in, fan-out)
  - Assembly level organization: Basic organization of the von Neumann machine; control unit; instruction fetch, decode, and execution; instruction sets and types (data manipulation, control, I/O); assembly/machine language programming; instruction formats; addressing modes; subroutine call and return mechanisms; I/O and interrupts
  - Memory systems: Storage systems and their technology; coding, data compression, and data integrity; memory hierarchy; main memory organization and operations; latency, cycle time, bandwidth, and interleaving; cache memories (address mapping, block size, replacement and store policy); virtual memory (page table, TLB); fault handling and reliability
  - Interfacing and communication: I/O fundamentals: handshaking, buffering, programmed I/O, interrupt-driven I/O; interrupt structures: vectored and prioritized, interrupt acknowledgment; external storage, physical organization, and drives; buses: bus protocols, arbitration, direct-memory access (DMA); introduction to networks; multimedia support; raid architectures

- **Methods of Delivery:**
  3 hours of lecture per week plus a 2 hour weekly lab

- **Student Evaluation**:
  Graded lab problem sets, 2 exams (midterm and final), assignments (at least 5)

- **Resources to be purchased/provided by students:**

  Textbook, reference sources and internet access (see textbook requirements below).

- **Textbook requirements:** (illustrative texts and other course materials)

The following is a list of representative textbooks appropriate for this course. The list is not exhaustive and the course instructor is encouraged to select a text appropriate for the focus they wish to place on the course

- Hamacher, V. Carl., Computer Organization (5th ed.), McGraw-Hill, c2002., ISBN: 0072320869
- Stallings, William., Computer Organization and Architecture: Designing for Performance (6th ed.), Saddle River, N.J., Prentice Hall, c2003., ISBN: 0130351199
- Steen, Maarten van., Computer and Network Organization: An Introduction, London ; New York, Prentice Hall, 1995., ISBN: 007025883X, McGraw-Hill computer science series.
- Tanenbaum, Andrew S., Structured Computer Organization, (4th ed.), Upper Saddle River, N.J., Prentice Hall, c1999., ISBN: 0130959901
- Waldon, John, Introduction to RISC Assembly Language Programming, c1998, ISBN:0201398281

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

  Outcome 1: Write and debug simple programs using assembly code

  Outcome 2: Understand and explain the principles underlying the design and development of computer systems for a variety of purposes

  Outcome 3: Be able to design and debug digital logic

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

A key issue in relation to this course is motivation. It is important to try to heighten the motivation of both students and faculty into seeing hardware design as an increasingly interesting, relevant, and challenging area. One approach is to include a significant laboratory component with the course that gives students the opportunity to build their own computer system or at least parts of a computer system. In doing so, they will come to appreciate the underlying issues at a much greater level of detail and get an "inside" view of a computer system. In addition, the students will experience a sense of accomplishment in the hardware area similar to what most students describe when they complete a significant software project.

Information about Course Designer/Developer

Course designed by Prof. Alexander Ferworn (Associate Professor of Computer Science, Ryerson University) and Dilip K. Banerji, Professor, Department of Computing & Information Science, University of Guelph.

List faculty eligible to teach the course

Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course web site will be a key resource available to participants. Suggested contents should include (but are not limited to)

- A Course Management Form,
- Labs (whether text-based or online),
- Lecture Notes, and
- Other resources such as links to relevant web sites.

Faculty Qualifications to teach/supervise the course:

A Postgraduate degree with expertise in Computer Architecture is recommended

Classroom requirements:

A technology-enhanced classroom with laptop connections, data projector, and Internet access should be made available

Equipment requirements:

A hardware lab consisting of individual work areas equipped with;

- workstations running hardware simulation software, or
- workbenches containing appropriate computing hardware components.

The creative use of simple processing platforms (such as embedded controllers, microcontrollers or microprocessors) is highly encouraged. A lab equipped with Field Programmable Gate Arrays (FPGAs) and supporting software from vendors (Xilinx, Altera) can be used both for this course and Computer Architecture II.

Course Title: Software Systems Development and Integration

Pre-Requisites: Principles of Computer Science

Year and Semester: Year 2, Semester 2

- **Course Description and Content Outline**(by topic)**:**

File systems and structures:
- File Structures: linked structures written to disk, binary files, records.
- Secondary storage devices: disk structure, performance issues.

Regular Expressions:
- Syntax, purpose, tools which use them, differences between implementations.

Scripting Languages:
- Modes of execution: compiled, interpreted, and both.
- Purpose: text processing, rapid development, integration between existing components.
- Characteristics: dynamic typing, complex data structures (lists, associative arrays), memory management.
- Syntax of a modern scripting language such as Python.

Software Tools and Philosophy:
- Source code control: purpose, operations, repository locking strategies.
- Build management: benefits of compiling independent components, dependencies.
- Compilers: purpose, common errors, compile and linking stages.
- Libraries: benefits, construction, associated files (headers and libraries), static and dynamic (shared).

Graphical User Interface Design:
- Design: organization and grouping or components, the goal of interface design, modal operations, components for dialogs with the user.
- Components: common elements found in interfaces.

Relational Databases and SQL:
- Relational Model: database organization, structure of tables.
- Type of execution: interactive or embedded.
- Language Syntax: common operations to create, destroy, and manipulate tables.

Coding Standards:
- Standards organizations: those relevant to software, the standards which they produce.
- Scope: extent which standards can enforce.

Performance:
- Performance: bottlenecks, methods to improve performance, when to optimize.
- Portability: issues when porting to a different language or different platform, strategies for writing portable code.

Debugging and Testing:
- Strategies for isolating errors, coverage and boundary testing, pre and post conditions, return values.

- **Methods of Delivery:** 3 hours of lecture per week.

- **Student Evaluation (typical)**:

Four programming assignments which provide exposure to the material presented during the lecture, a midterm and a final exam. Assignments are cumulative and intended to produce a single large

application once they are completed and integrated. This places a great importance on the completion all assignments.

- **Resources to be purchased/provided by students:** The textbooks.

- **Textbook requirements:** Due to the breadth of the course there is not a single text which covers all of the course material. The software tools and system level information is available in Beginning Linux Programming, by Richard Stones and Neil Matthew, from Wrox Press Ltd. For a scripting language reference Programming Python by Mark Lutz, published by O'Reilly and Associates Inc.

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:
  Outcome 1: Develop and use software libraries.
  Outcome 2: Identify types of software tools and the applications for which they are most appropriate.
  Outcome 3: Use common development tools including compilers, build utilities, and source control.
  Outcome 4: Build a graphical interface and be familiar with the common components found in interfaces.
  Outcome 5: Create and perform simple interactions using a relational database.
  Outcome 6: Integrate several parts of a system which have been developed using different programming languages and tools.
  Outcome 7: Identify the primary standards organizations and the standards related to computing which they maintain.
  Outcome 8: Describe the issues which are relevant to software portability and performance tuning.
  Outcome 9: Develop software which constructs and manages a complex data structure (such as a tree or linked structure) on disk.
  Outcome 10: Write and interpret regular expressions.

- **Rationale:** The learning outcomes are primarily supported by the programming assignments. The assignments are intended to reinforce the lecture material by providing hands-on experience with the technology and techniques which are described in the lectures. Assignments generally involve experience with data structures on disk, relational databases, graphical interface development, constructing a library, and scripting languages. The integration of all of the assignments into a single application by the end of the course provides experience in developing a large, non-trivial application and emphasizes the importance of proper design.

Information about Course Designer/ Developer
Course designed by David Calvert (Assistant Professor, Computing and Information Science, University of Guelph).

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line? The course web site will provide lecture notes. Sample code will also be provided which demonstrates the use of the more complex libraries or tools which will be used in the assignments.

Faculty Qualifications to teach/supervise the course: A strong working knowledge in software development, tools, and integration. A Postgraduate degree in Computing Science.

Classroom requirements: Classroom with a laptop projector.

Equipment requirements: Students will require access to computers running a Unix derivative (such as Linux) operating system which have the software libraries and tools used in the course installed on them. A database server running a relational database is required (examples of which include Mysql or Postgress). Access to the internet is also required to allow students to search the online documentation for the software tools.

Course Title: Operating Systems and Networking

Pre-Requisites: Principles of Computer Science and Computer Architecture I

Year and Semester: Year 3, Semester 1

- **Course Description and Content Outline** (by topic)**:**

This course will cover a variety of system topics related to the operating system, with emphasis on components that are unique to its role as the interface layer between computer hardware and application software. The course will explore techniques for sharing the processor, memory, secondary storage and networking between programs. The basics of networking will also be introduced, particularly at the higher protocol levels.

- Basic structure of a process: address space, registers, program counter and stack
- Asynchronous activity: hardware and software interrupts, service routines
- The implementation of multiprogramming: co-routines, the context switch, process status
- Sharing of the processor (multiprogramming): Concurrent processes, the problem of shared variables, race conditions, critical sections and techniques for partial synchronization, such as mutual exclusion, semaphores and strict message passing
- Performance implications of multiprogramming: processor scheduling, queueing
- Management of main memory: MMU hardware, virtual memory, segmentation and paging algorithms, their performance, thrashing
- Handling of secondary storage: block and byte I/O, internal buffer handling, file system structures
- Operating System Kernel: Internal structure, system call interface
- Client/server model for distributed computing, remote procedure calls
- Operation of local area networks, internet protocols, communication between processes (TCP, UDP, Java socket API)

- **Methods of Delivery:** 3 hours of lecture per week plus a 1 hour weekly tutorial/seminar.

- **Student Evaluation**:

Assignment Section (30%): 4 individual assignments

Examination Section (70%)
- Midterm: 30%
- Final Examination: 40%

- **Resources to be purchased/provided by students:** textbooks

- **Textbook requirements:**
  - *Operating System Concepts*, by Abraham Silberschatz and Peter Galvin, John Wiley and Sons

- **Learning Outcomes.**
  Students who successfully complete the course have reliably demonstrated the following abilities:

  Outcome 1: Understanding of the structure of an operating system along with some familiarity of system programming interfaces.
  Outcome 2: Appreciation for the infrastructure that supports and enables distributed computing.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Outcome 1 will be achieved through the assignments that will reinforce the material from the lectures and the textbook.

Outcome 2 will be developed through lectures and seminar sessions that include tutorials on specific aspects of networking, particularly the protocols that support the Internet.

Information about Course Designer/ Developer
Course designed by Deborah Stacey, Computing and Information Science, University of Guelph from course material designed by Rick Macklem, Andrew Hamilton-Wright and Tom Wilson, Computing and Information Science, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course website will be a key resource for the students of this course.

Faculty Qualifications to teach/supervise the course:

The faculty member should have a postgraduate degree in Computer Science, with knowledge of the basic principles of operating systems (particularly UNIX) and networking (particularly Internet protocols).

Classroom requirements: Some of the seminar hours must be in an environment where the class can have access to UNIX. A technology-enhanced classroom is also a requirement.

Equipment requirements:

Appropriate workstations and software must be available to the class for their individual assignments. Software includes a program development environment for predominately C programming.

Course Title: Database Systems and Concepts

Pre-Requisites: Principles of Computer Science, Discrete Structures in Computer Science

Year and Semester: Year 2, Semester 2

- **Course Description and Content Outline** (by topic)**:**

  The aim of the course is to provide students with an overview of database management system architectures and environments, an understanding of basic database design and implementation techniques, and practical experience of designing and building a relational database.

Syllabus:

- Information models and systems: History and motivation for information systems; information storage and retrieval; information management applications; information capture and representation; analysis and indexing; search, retrieval, linking, navigation; information privacy, integrity, security, and preservation; scalability, efficiency, and effectiveness
- Database systems: History and motivation for database systems; components of database systems; DBMS functions; database architecture and data independence
- Data modeling: Data modeling; conceptual models; object-oriented model; relational data model
- Relational databases: Mapping conceptual schema to a relational schema; entity and referential integrity; relational algebra and relational calculus
- Database query languages: Overview of database languages; SQL; query optimization; 4th-generation environments; embedding non-procedural queries in a procedural language; introduction to Object Query Language
- Relational database design: Database design; functional dependency; normal forms; multivalued dependency; join dependency; representation theory
- Transaction processing: Transactions; failure and recovery; concurrency control
- Distributed databases: Distributed data storage; distributed query processing; distributed transaction model; concurrency control; homogeneous and heterogeneous solutions; client-server
- Physical database design: Storage and file structure; indexed files; hashed files; signature files; b-trees; files with dense index; files with variable length records; database efficiency and tuning

- **Methods of Delivery:**

  3 hours of lecture per week, 1 lab per week.

- **Student Evaluation (typical)**:

2 graded assignments, graded lab problem sets, 2 exams (midterm and final)

- **Resources to be purchased/provided by students:**

  Textbook, reference sources and internet access (see Textbook requirements below).

  - **Textbook requirements:**

    The following is a list of representative text books appropriate for this course. The list is not exhaustive and the course instructor is encouraged to select a text appropriate for the focus they wish to place on the course.

    - Connolly, Thomas M., Database systems : a practical approach to design, implementation, and management, 3rd ed., Harlow, England ; New York : Addison-Wesley, 2002., ISBN: 0201708574
    - Date, C. J., An introduction to database systems 7th ed., Reading, Mass., Addison-Wesley, c2000., ISBN: 0201385902
    - Garcia-Molina, Hector., Database systems : the complete book, Upper Saddle River, NJ : Prentice Hall ; Toronto : Pearson Education, 2002., ISBN: 0130319953
    - Harrington, Jan L.,Relational database design clearly explained 2nd ed., New York : Morgan Kaufmann Publishers, c2002., ISBN: 1558608206
    - Kroenke, David., Database processing : fundamentals, design & implementation 8th ed., Upper Saddle River, NJ : Prentice Hall, c2002., ISBN: 0130648396
    - Silberschatz, Abraham., Database system concepts 4th ed., Boston : McGraw-Hill, c2002., ISBN: 0072283637

- **Learning Outcomes.**

  Students who successfully complete the course have reliably demonstrated the ability to:

  Outcome 1: Describe what a database is.
  Outcome 2: Recognize different forms of organizational structures within databases
  Outcome 3: Describe the characteristics of different types of database organizations.

Outcome 4: Apply suitable techniques to be able to design and implement database systems.
Outcome 5: Be able to discuss/explain the importance of data, and the difference between file management and databases.
Outcome 6: Be able to discuss/explain the principals of database design.
Outcome 7: Be able to discuss, explain and apply the relational model and mappings from conceptual designs, in particular normalisation.
Outcome 8: Be able to discuss/explain physical and performance related design considerations.
Outcome 9: Be able to discuss/explain transaction processing.

Rationale: (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Given the extremely wide distribution of database technology and the applicability of database techniques to a wide variety of applications, it is appropriate that an introductory course present critical elements of the field in a manner that both emphasises the course components that will lead to the desired outcomes and allows the flexibility to concentrate of areas of interest.

The use of lectures to deliver the main content of the course followed by appropriate lab content will provide students with an appropriate balance between theoretical knowledge and practical experience.

Information about Course Designer/ Developer

Course designed by Prof. Alexander Ferworn (Associate Professor of Computer Science, Ryerson University).

List faculty eligible to teach the course

Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course web site will be a key resource available to participants. Suggested contents should include (but are not limited to)

- A Course Management Form,
- Labs (weather text-based or online),
- Lecture Notes, and
- Other resources such as links to relevant web sites.

Faculty Qualifications to teach/supervise the course:

A Postgraduate degree with expertise in databases is recommended.

Classroom requirements:

A technology-enhanced classroom with laptop connections, data projector, and Internet access should be made available.

Equipment requirements:

An appropriate set of database software tools must be made available to participants.

Course Title: System Analysis and Design in Applications

Pre-Requisites: Principles of Computer Science

Year and Semester: Year 3, Semester 1

- **Course Description and Content Outline**(by topic)**:**

Software Engineering Principles:

Introduction to software engineering, models of software evolution - life cycle and process;
computer systems engineering, software quality, software reviews, software testing, stakeholders.

Requirements Analysis and Specifications:

Requirements elicitation, requirements specification, software prototyping, informal and formal specification models, requirements validation, types of requirements, application-specific requirements.

Design:

Introduction to software design, Object-Oriented Design, Structured Design, Data Driven design, Data Flow design, design document contents, graphical interface and database design issues.

Development:

Project construction, document and code reviews, scheduled presentations of progress and demonstration of milestones, testing strategy.

- **Methods of Delivery:** 3 hours of lecture per week, 2 hours of lab per week. Lab times are primarily used for project demonstrations.

- **Student Evaluation (typical)**:

Several quizzes, a final exam, and a term project. The project requires the students to work within a group to produce a requirements document, a design document, a schedule of deliverables, and several presentations throughout the semester which demonstrate progress during the development. The quizzes are derived from assigned readings from the Sommerville and Sawyer book and are designed to demonstrate knowledge of the various aspects of the requirements process.

- **Resources to be purchased/provided by students:** The textbooks.

- **Textbook requirements:** Code Complete by Steve McConnell and published by Microsoft Press describes the practical elements of software development. Requirements Engineering: A good practice guide by Ian Sommerville and Pete Sawyer and published by John Wiley and Sons is a comprehensive description of the issues involved in requirements analysis.

  Instructors may wish to use Software Engineering by Ian Sommerville and published by Addison-Wesley and Software Requirements: Objects, Functions, and States, by Alan M. Davis and published by Prentice Hall for some of the lecture material. Software Project Survival Guide by Steve McConnell and published by Microsoft Press describes the project development process.

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:
  Outcome 1: Identify the stages in the software development life cycle.
  Outcome 2: Elicit requirements from a user.
  Outcome 3: Prepare both requirements and design documents.
  Outcome 4: Describe the different design techniques.
  Outcome 5: Productively interact within a group which is working on a software development project.
  Outcome 6: Perform a group review of a requirements or design document or for a section of code.
  Outcome 7: Schedule deliverables for a development project.
  Outcome 8: Translate a list of requirements into a design using either Top-Down or Bottom-Up design.
  Outcome 9: Develop a testing strategy.
  Outcome 10: Perform a demonstration of their project which clearly illustrates its achievements and limitations.

- **Rationale:** The students will experience the software development activities described in the lecture material through their participation in the group project. The project provides direct experience with all of the steps in the software development process. Working within a group provides the students with an important experience which mirrors the real-world development process.

Information about Course Designer/ Developer
Course designed by David Calvert (Assistant Professor, Computing and Information Science, University of Guelph)

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line? The course web site will provide the lecture notes.

Faculty Qualifications to teach/supervise the course: A postgraduate degree in Computer Science is desirable. Experience in managing software development projects is desirable.

Classroom requirements: Technology-enhanced classroom

Equipment requirements: Students will need access to computers with a development environment installed. Due to the group nature of the project, a dedicated lab is recommended to allow the students to work as a group during the development phase and not disturb students in other courses. The term projects often involve exposure to different technologies and can require some specialized software and dedicated server machines.

Course Title: Computer Architecture II

Pre-Requisites: Computer Architecture I, Operating Systems and Networking

Year and Semester: Year 3, semester 2

- **Course Description and Content Outline**(by topic)**:**

Description
This course presents advanced topics in computer architecture and builds on the concepts presented in Computer Architecture I. The emphasis is on analyzing how the various components or subsystems of a computer system interact with and affect each other. This will enable the student to see how performance of computer systems is enhanced by tweaking their architecture.
Outline

- Functional organization: data paths; control unit: hardwired control vs. micro-programmed control.
- Classification of architectures: SIMD, MIMD, VLIW, EPIC; systolic architecture
- Instruction set design: Tradeoffs in instruction set design, RISC vs CISC
- Performance modelling and analysis of computer systems, price/performance tradeoffs
- Interconnection networks, shared memory systems, memory models and consistency
- Single and multi-level cache modelling and their effect on performance, cache coherence
- Pipelining: Arithmetic pipelines, instruction pipelines, pipeline performance, pipeline hazards and how to deal with them, out of order instruction execution
- Performance enhancements: RISC architecture; branch prediction; prefetching; super scalar architecture
- Multiprocessor and alternative architectures
- Contemporary architectures: Hand-held devices; embedded systems; trends in processor architectures

- **Methods of Delivery:**
  3 hours of lecture per week plus a 2-hour weekly lab

- **Student Evaluation**:
  Graded lab problem sets or a project, 2 exams (midterm and final), assignments

- **Resources to be purchased/provided by students:**
  Textbook, reference sources and internet access (see Textbook requirements below).

- **Textbook requirements:** (illustrative texts and other course materials)

The following is a list of representative textbooks appropriate for this course. The list is not exhaustive and the course instructor is encouraged to select a text appropriate for the focus they wish to place on the course

- Hayes, John P., Computer Architecture and Organization (3rd ed.), Boston, Mass., WCB/McGraw-Hill, c1998., McGraw-Hill computer science series.
- Hennessy and Patterson, Computer Architecture: A Quantitative Approach(3rd ed), San Mateo, Calif., Morgan Kaufmann, 2002
- Hwang, Kai, Advanced Computer Architecture, McGraw Hill, 1993 (Later edition possibly exists)

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:
  Outcome 1: Appreciate the architectural tradeoffs and their effect on performance
  Outcome 2: How to model a computer system and its subcomponents for performance analysis
  Outcome 3: Understand the synergy between architecture and software
  Outcome 4: Analyze a given or proposed architecture for its performance

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Differences in the architecture of a computer system lead to significant differences in performance and functionality, giving rise to an extraordinary range of computing devices, from hand-held computers to large-scale, high-performance machines. This course and its method of delivery addresses the various options involved in designing a computer system, the range of design considerations, and the trade-offs involved in the design process.

Software tools can play an important role in this course, particularly when funding for a hardware laboratory is not available. These tools include, for example, instruction set simulators, software that will simulate cache performance, benchmark systems that will evaluate performance, and so on. If an FPGA based lab is set up for this course and CA I, then these devices can be used to actually implement some architectures.

Information about Course Designer/ Developer

Course designed by Prof. Alexander Ferworn (Associate Professor of Computer Science, Ryerson University) and modified by Dilip K. Banerji, Professor, Dept of Computing & Information Science, University of Guelph

List faculty eligible to teach the course

Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course web site will be a key resource available to participants. Suggested contents should include (but are not limited to)

- A Course Management Form,
- Labs (whether text-based or online),
- Lecture Notes, and
- Other resources such as links to relevant web sites

Faculty Qualifications to teach/supervise the course:

A Postgraduate degree with expertise in Computer Architecture is recommended**.**

Classroom requirements:

A technology-enhanced classroom with laptop connections, data projector, and Internet access should be made available

Equipment requirements:

A lab consisting of individual work areas equipped with;

- workstations running architecture simulation software, or
- FPGA based workbenches to implement architectures for experimentation and evaluation

Course Title: Software Engineering

Pre-Requisites: Systems Analysis and Design in Applications

Year and Semester: Year 3, Semester 2

- **Course Description and Content Outline** (by topic)**:**

This course is an examination of the software engineering process and the production of reliable software systems. It includes an advanced look at techniques for the design and development of complex software. Since this is an advanced software engineering course, the material will consist of those topics and techniques that are at the leading-edge of current thinking about object-oriented analysis, design and modeling, software architectures and development paradigms, software reviews, software quality, software engineering, ethics, maintenance and formal specifications. Topics include:

Software Quality and Software Quality Assurance
UML (Unified Modelling Language)
Design Patterns
Software Metrics and Testing
Project Management including Reviews
Secure Software (Problems, Techniques)
Formal Specifications and the Z Language
Introduction to Ethics
Introduction to Extreme Programming (Techniques such as Test First and Pair Programming)

- **Methods of Delivery:** 3 hours of lecture per week plus a 2 hour weekly tutorial/seminar.

- **Student Evaluation (typical)**:

Assignment Section (30%): 3 or 4 individual assignments in any of the following areas:
- Extreme Programming / Pair Programming
- Testing and Metrics
- Design Patterns
- Formal Specifications in Z

Project Section (40%)
- Team project (Requirements. Specification, Design, Implementation, Testing, Demonstration)

Examination Section (30%)
- Midterm: 10%
- Final Examination: 20%

- **Resources to be purchased/provided by students:** textbooks

- **Textbook requirements:** There is no single text that covers the material of this course in sufficient detail and depth. The following books are examples of books that provide the level of detail necessary for this course:
    - Applying UML and Patterns by Craig Larmen, 2nd edition
    - An Introduction to Formal Specifications and Z by Potter, Sinclair and Till, 2nd edition
    - Building Secure Software by Viega and McGraw, 2001
    - Extreme Programming Pocket Guide by Chromatic, 2003

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the following abilities:
  Outcome 1: Mature approach to eliciting requirements.
  Outcome 2: Mature approach to writing detail specifications for a large system.
  Outcome 3: Advanced object-oriented design skills.
  Outcome 4: Knowledge of design patterns and their uses.
  Outcome 5: Knowledge of and ability to use UML.

Outcome 6: Introduction to formal specifications and practise in using a particular formal language.
Outcome 7: Introduction to different ways of approaching the design of software including the experience of using a current leading-edge paradigm in both tutorial exercises and in the team project.
Outcome 8: Knowledge of the use and analysis of software metrics, SQA, and testing.
Outcome 9: Introduction to the complexities involved in producing secure software systems.
Outcome 10: Ability and experience in intensive teamwork situations.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

This course involves a project to develop a large, complex computing system in a team environment. This team project reinforces the utility of developing the skills listed in Outcomes 1, 2, 3, 7, and 10. The experience of working as a development team while taking a project from requirements to specification to design to implementation to testing and to delivery cannot be taught by books alone.

The seminars/tutorials are excellent avenues for running short exercises in advanced skills such as pair programming, OO design, and design pattern appreciation. The limited time of the seminars intensifies the learning experience and provides instruction on techniques that can be used in the team project.

Information about Course Designer/ Developer
Course designed by Deborah Stacey, Professor of Computing and Information Science, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course website will be a key resource for the students of this course.

Faculty Qualifications to teach/supervise the course:

The faculty member should have a postgraduate degree in Computer Science, with knowledge of basic software engineering principles. Specific knowledge of UML, design patterns, formal methods, and software metrics is also needed. The ability and experience to manage and evaluate large team projects would be an asset.

Classroom requirements: Some of the seminar hours must be in an environment where all the class can have access to workstations.

Equipment requirements:

Appropriate workstations and software must be available to the class for their individual assignments and team project. Software includes an object-oriented language compiler, program development environment, and necessary software for the team project (this may change from offering to offering).

Course Title: Analysis and Design of Algorithms

Pre-Requisites: Discrete Structures in Computer Science,
Fundamentals of Programming
Principles of Computer Science

Year and Semester: Year 3, Semester 1

- **Course Description and Content Outline:** This course exposes students to the fundamental techniques for designing efficient computer algorithms, proving their correctness, and analysing their complexity.

  Topics will include, but will not be limited to the following.
  1. Introduction to the area
  2. Basic Algorithm Analysis
     a) some classics: search, sort, Fibonacci, GCD, determinants;
     b) analysis techniques; efficiency and order:
  3. Greedy Algorithms
     a) graph oriented: spanning trees, path traversal, scheduling;
     b) other classics, including matching, knapsack, min cover.
  4. Divide and Conquer Algorithms
  5. Dynamic Programming Algorithms
  6. Search Space and Tree Traversal
     (a) Backtracking Algorithms
     (b) Branch 'n Bound Algorithms
  7. Computational Complexity (NP completeness)
  8. Heuristics -- how to handle really hard problems

- **Methods of Delivery:** 3 hours of lecture per week plus a two-hour weekly tutorial/seminar.

- **Student Evaluation (typical)**:
  Assignment Section 30%: 4 to 5 individual and group assignments.
  Examination Section (70%)
  - Quizzes: 20%
  - Midterm: 20%
  - Final Examination: 30%

- **Resources to be purchased/provided by students:** textbooks

- **Textbook requirements:**

  - Foundations of Algorithms using C++ Pseudocode by Richard Neapolitan & Kumarss Naimpour. (3rd Edition). Jones and Bartlett, 2004.
  - Introduction to Algorithms (2nd Ed.), Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. McGraw Hill, 2001. (Reference).

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:
  Outcome 1: analyse algorithms built on different paradigms, such as recursive algorithms, backtracking algorithms, branch and bound algorithms;
  Outcome 2: recognize the different types of recurrence relations, including divide-and-conquer recurrences, linear recurrences and homogeneous linear recurrences with constant coefficients, and methods of solving each type of recurrence.
  Outcome 3: solve recurrence relations, and use them to determine the complexity of recursive algorithms;

Outcome 4: determine the basic algorithmic paradigms that should be used to solve various problems, e.g. divide-and-conquer, dynamic programming, backtracking, branch and bound, or greedy algorithms;
Outcome 5: design algorithms, given problems, and be capable of implementing them in structured high-level computer languages;
Outcome 6: determine whether a problem has a polynomial solution, is intractable, or is undecidable.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

  These outcomes will be developed through lectures and seminar sessions that include demonstrations of the various algorithmic paradigms. Outcomes 1, 2, 3, and 6 are to be acquired through lectures, and seminars. In the latter, students have a hands-on experience in solving problems and analyzing solutions. Assignments are given to enhance this knowledge, and after designing of the algorithms, students are expected to implement them in a given programming language.

  Outcomes 4 and 5 are acquired through lectures and assignments. In some of the assignments groups of 2 or 3 students are required to work together to write complex programs.

Information about Course Designer/ Developer
Course designed by Dr. Charlie Obimbo of Computing and Information Science, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course website will be a key resource for the students of this course.

Faculty Qualifications to teach/supervise the course:

The faculty member should have knowledge of the basic principles of analysis and design of Computer Programs, a strong background in theoretical computer science, and its applications.

The essence of this course is to **design** algorithms and analyze, and this comes better when taught at both conceptual and implementation levels. This can be done well by one who is strong in both areas.

Classroom requirements: Smart classrooms would be desirable, so as to enable the demonstration of programs. (These would be classrooms with LCD projectors and screens for display. White boards are convenient, but not absolutely necessary).

Equipment requirements:
Appropriate workstations and software must be available to the class for their individual assignments and team project. Software includes high-level language compilers (C, C++ and Java), and a program development environment.

Course Title: Compilers

Pre-Requisites: Computer Architecture II, Operating Systems

- *experience with a group project course is helpful*
- *Elements of Theory of Computation is a useful co-requisite, however a sufficient introduction to regular expressions, deterministic finite automata and context free grammars can be provided as part of this course making an explicit pre-/co-requisite unnecessary unless it is desired.*

Year and Semester: Year 4, Semester 1

Course Description and Content Outline (by topic)**:**

This course provides a detailed study of the compilation process for a procedural language. Students will develop an understanding of compiler design and put these principles into practice through the construction of a fully functioning compiler for a small procedural language using widely available tools for compiler construction and a general-purpose programming language.

Topics to be covered include:

- lexical analysis (regular expressions, DFAs, tokenizing algorithms, scanner generators)
- parsing (context free grammars, LL, LR, LALR parsing, parser generators)
- syntax analysis (syntax directed translation, abstract syntax trees, detecting and reporting syntax errors)
- semantic analysis (type environments, symbol tables)
- run-time environments (stack frame layout, scoping resolution, translation strategies)
- representation and manipulation of intermediate code (quadruples, 3-address code, trees)
- liveness analysis and register allocation (graph colouring, register allocation strategies)
- code generation
- topics (time permitting): compiler optimizations, object oriented compilation, etc.

- **Methods of Delivery:** 3 hours of lecture / 1 hour of tutorial/seminar per week

- **Student Evaluation (typical)**:

Assignments (2 x 10%)

- lexical analysis design/implementation
- recursive descent parsing design/implementation

Project (40%)

- group-based project to implement a compiler given specific source and target languages (varied each year).
  - Source languages are best specified by taking some existing language and stripping out components or semantics you don't want (e.g. C with no pointers and only integer, char and floating point types). This allows the source language to be varied year-to-year which is critical.
  - Target languages are best expressed in an assembly language. One option is to use the assembly language native to machines in the student labs which can be assembled and run natively. Another option is to use a simulator such as SPIM (a MIPS R2000/R3000 simulator). SPIM is particularly attractive as it is freely available for multiple platforms and implements a full graphical interface to the virtual processor allowing students to view registers and memory during program execution.
- Checkpoint 1 (5-10%)
  - lexical analysis and parsing to the point of building abstract syntax trees
- Checkpoint 2 (5-10%)
  - semantic analysis and symbol tables to the point of building intermediate code
- Final submission (20-30%)
  - complete compilation from source language to target language

- Weighting of checkpoints and final submission are left to the instructor's discretion depending on the emphasis placed on completeness at each step.

Quizzes (4 x 5%)

- small diagnostic quizzes written in-class ensure students keep abreast of course material (can be replaced with a 20% midterm at instructor's discretion)

Final Exam (30%)

- comprehensive exam emphasizes understanding of major topics and principles

- **Resources to be purchased/provided by students:** textbooks

- **Textbook requirements:**

Required

Alfred V. Aho, Ravi Sethi and Jeffery D. Ullman, *Compilers: Principles, Techniques and Tools*. Addison Wesley, 1988 (ISBN 0-201-10088-6).

Recommended

John R. Levine, Tony Mason and Doug Brown, *lex & yacc (2e)* . O'Reilly & Associates, 1992 (ISBN 1-56592-000-7).

NOTE: I find Aho, Sethi and Ullman an ideal choice of text as it is highly independent of source and target language issues making it suitable for ongoing instruction where such details will vary. The book is weak covering compiler optimizations and non-procedural languages (although such material is rarely missed in undergraduate compiler course). If the instructor wishes to delve more deeply into these topics then supplemental material will be required, or another text should be selected.

Suggested Alternative:

D. Grune, H. Bal, C. Jacobs, and K. Langendoen, *Modern Compiler Design* , Wiley, 2000.

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1:	Understand the compilation process in depth
Outcome 2:	Design and build a basic compiler for a procedural language
Outcome 3:	Formally specify basic languages using regular expression and context free grammars
Outcome 4:	Approach the process of parsing in a sophisticated manner (both within the context of a compiler and in the more general sense)
Outcome 5:	Use development tools such as lex and yacc at an advanced level
Outcome 6:	Design, build and manipulate complex data structures in an applied context
Outcome 7:	Understand and implement a wide variety of algorithms of varying sophistication
Outcome 8:	Use modularity to enable the design and construction of otherwise complex systems
Outcome 9:	Work productively in a group setting for software development

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Lectures, together with assigned readings from the textbook, serve as the primary source of conceptual information and understanding (contributing to Outcome 1, 3, 7 and 8). These principles are put into practice through the construction of a fully functioning compiler for a small procedural language using widely available tools for compiler construction (e.g. lex and yacc) and a general purpose programming language (Outcome 2-9).

Tutorials provide directed implementation guidance in the context of the implementation language(s) for the course, making use of the source and target compilation languages in examples. Tutorials provide the bridge between the conceptual material, represented by the lectures and text, and the implementation of these concepts for the project (Outcome 1, 2, 4, 5, 6 and 7).

Assignments are designed to reinforce various concepts that are typically abstracted by tools during compiler construction in order to provide the student with a complete picture of the compiler design and implementation process (Outcome 3, 4, and 7).

Information about Course Designer/ Developer

Course designed by: **David McCaughan**, *High Performance Computing Consultant*, SHARCNET.ca, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A well organized course web-site with links to resources and additional reference material is i invaluable in any course.

Faculty Qualifications to teach/supervise the course:

The instructor should be have knowledge of intermediate compiler construction techniques for procedural programming languages including: scanner/parser construction using tools such as lex and yacc; algorithms and data structures for manipulating trees and lists; a good grasp of assembly language fundamentals. Previous Experience implementing a compiler (even as a student) is helpful. Specific knowledge regarding the chosen implementation language(s) (C, C++ and/or Java) is also an asset. Any expertise with more advanced issues (object oriented compilers, advanced compiler optimizations, etc.) can greatly enrich the course however are not necessary.

Classroom requirements:

Lectures and seminar can be held in any suitable classroom.

Equipment requirements:

Appropriate workstations and software must be available to the class for their individual assignments and term project. Software includes suitable compilers for the language of implementation, an assembler or simulator to process the output from student compilers, scanner and parser generators appropriate to the language of implementation (e.g. lex and yacc (C), JFlex and CUP (Java))

Sofware Resources:

SPIM (MIPS R2000/R3000 simulator): http://www.cs.wisc.edu/~larus/spim.html

JFex (Java scanner generator): http://www.jflex.de

CUP (Java parser gen): http://www.cs.princeton.edu/~appel/modern/java/CUP/

Course Title: Elements of the Theory of Computation

Pre-Requisites: Analysis and Design of Algorithms

Year and Semester: Year 4, elective

- **Course Description and Content Outline**(by topic)**:**

Provides and develops an understanding of what problems are inherently computable and what problems are tractable or feasible. Topics will include Church's thesis, recursively enumerable sets, Godel's incompleteness theorem as well as complexity results related to Turing machines and P vs NP hardness.

A. Computability and Noncomputability
B. RE sets and Church's thesis
C. Godel's Theorem
D. The Turing machine model
E. P and NP complexity
F. NP completeness
G. The travelling salesman problem and it's variations

- **Methods of Delivery: 3 hours of lecture and 1 hour of tutorial per week**

- **Student Evaluation**: 4-6 assignments (worth 40%), a term test (worth 20%) and a final exam (worth 40%)

- **Resources to be purchased/provided by students:** Textbook

- **Textbook requirements:** (illustrative texts and other course materials)

Possible texts for the course include:
M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness.
M. Sipser, Introduction to the Theory of Computation.

- **Learning Outcomes.**
- Students who successfully complete the course have reliably demonstrated the ability to:

  Outcome 1: Classify problems based on the difficulty involved in solving the problem with a finite algorithm.

  Outcome 2: Understand the standard model(s) of computation and how insensitive the problem classification is to this model.

  Outcome 3: Understand the inherent time and space required to to solve a given problem

  Outcome 4: Understand and appreciate the need for `approximate algorithms'.

  Outcome 5: Understand and be able to compare the theoretical performance of different algorithms.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Case studies of well known problems such as the travelling salesman problem, the four colour problem and linear programming will be presented and discussed in the tutorials. These examples will illustrate where new insights and improved understanding of computation is possible. The focus of the course is to understand the implications of the theoretical results rather than to develop proofs of new results.

Information about Course Designer/ Developer

The course was designed by Wayne Enright, Professor of Computer Science at the University of Toronto.

Assignments are designed to reinforce various concepts that are typically abstracted by tools during compiler construction in order to provide the student with a complete picture of the compiler design and implementation process (Outcome 3, 4, and 7).

Information about Course Designer/ Developer

Course designed by: **David McCaughan**, *High Performance Computing Consultant*, SHARCNET.ca, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A well organized course web-site with links to resources and additional reference material is i invaluable in any course.

Faculty Qualifications to teach/supervise the course:

The instructor should be have knowledge of intermediate compiler construction techniques for procedural programming languages including: scanner/parser construction using tools such as lex and yacc; algorithms and data structures for manipulating trees and lists; a good grasp of assembly language fundamentals. Previous Experience implementing a compiler (even as a student) is helpful. Specific knowledge regarding the chosen implementation language(s) (C, C++ and/or Java) is also an asset. Any expertise with more advanced issues (object oriented compilers, advanced compiler optimizations, etc.) can greatly enrich the course however are not necessary.

Classroom requirements:

Lectures and seminar can be held in any suitable classroom.

Equipment requirements:

Appropriate workstations and software must be available to the class for their individual assignments and term project. Software includes suitable compilers for the language of implementation, an assembler or simulator to process the output from student compilers, scanner and parser generators appropriate to the language of implementation (e.g. lex and yacc (C), JFlex and CUP (Java))

Sofware Resources:

SPIM (MIPS R2000/R3000 simulator): http://www.cs.wisc.edu/~larus/spim.html

JFex (Java scanner generator): http://www.jflex.de

CUP (Java parser gen): http://www.cs.princeton.edu/~appel/modern/java/CUP/

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?
A course website with expanded course notes and sample problems of different levels of complexity is expected but no web-based course delivery is expected.

Faculty Qualifications to teach/supervise the course:
A PhD or MSc in Computer Science

Classroom requirements:
A standard electronic classroom is adequate.

Equipment requirements:
None

Course Title: Ethics, Law and the Social Impact of Computing

Pre-Requisites: Software Engineering

Year and Semester: Year 4, Semester 2

- **Course Description and Content Outline** (by topic)**:**

This course is an examination of the impact that computing has on society and the impact that society has on computing. The development of laws and social mechanisms has not kept pace with the rapid development and deployment of computing and computing devices in our society. The ethics to deal with this situation exist but are not widely studied by students of computing. Current issues, developments and trends in computing ethics and law will be examined. The impact that computing has on society will be examined in light of the need for professional ethics and appropriate laws and regulatory agencies. Topics include:

Ethics:
- definition of ethics
- issues in privacy, security, access, intellectual property, internationalization
- professional codes of ethics
- case studies

Law:
- intellectual property – its definition, history and future
- copyrighting software
- patenting software
- data privacy and copyrighting
- open source and copylefting
- international issues
- relevant commercial law
- relevant criminal law
- case studies

Social Impacts:
- ubiquitous nature of computers
- the definition of privacy (and its loss) in modern society – impact of GPS/locator devices and monitoring and use and restrictions on uses of encryption
- threats to society from hacking, cyberattacks (e.g. viruses, denial of service), spam
- safety issues: computer control of industrial processes, transportation, medical devices, etc.
- e-business and its impact on business and society
- e-government and its influence on public empowerment
- e-elections and its impact on democracy

- **Methods of Delivery:** 3 hours of lecture per week plus a 2 hour weekly tutorial/seminar.

- **Student Evaluation (typical)**:

- Assignments (30%)
- Major Research Paper (30%)
- Examination Section (40%)
  - Midterm: 15%
  - Final Examination: 25%

- **Resources to be purchased/provided by students:** textbooks

- **Textbook requirements:** There is no single text that covers the material of this course in sufficient detail and depth. The following books are examples of books that provide the level of detail necessary for this course:
  - Computer Ethics by Forester and Morrison, 1990

- Essentials of Canadian Law: Computer Law by George S. Takach, 1998

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the following abilities:
  Outcome 1: Mature appreciation for the need for a professional code of ethics for computing professionals.
  Outcome 2: Mature appreciation for the complex issues involved with the ethical development and deployment of computing systems.
  Outcome 3: An introduction to the law as it pertains to the computing profession.
  Outcome 4: Knowledge of the issue of intellectual property and computing.
  Outcome 5: The concern and compassion to explore fully the impact of what they do in their profession on the lives of others in our society and in societies around the world (particularly in the developing world).

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

This course involves a major research paper. This paper will allow the student to explore specific issues that interest the student in a deep and meaningful way. Since this is a fourth-year course, the research paper will also prepare the student for the reading and writing of professional documents and/or graduate research documents.

The seminars/tutorials are excellent avenues for exploring case studies. These case studies will give the students an opportunity to engage in discussion and debate on current ethical, legal and social issues.

Information about Course Designer/ Developer
Course designed by Deborah Stacey, Computing and Information Science, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course website will be a key resource for the students of this course.

Faculty Qualifications to teach/supervise the course:

The faculty member should have knowledge of computing ethics and law.

Classroom requirements: The seminar should be in a room conducive to group discussions.

Equipment requirements: None

Course Title: Artificial Intelligence

Pre-Requisites: Principles of Computer Science, Discrete Structures in Computer Science

Year and Semester: Year 4, Semester 1 or 2 (elective)

Course Description and Content Outline(by topic)**:**

The course introduces students to the fundamental concepts and techniques of artificial intelligence (AI).

Syllabus:

- Fundamental issues in intelligent systems: History of artificial intelligence; philosophical questions; fundamental definitions; philosophical questions; modeling the world; the role of heuristics
- Search and constraint satisfaction: Problem spaces; brute-force search; best-first search; two-player games; constraint satisfaction
- Knowledge representation and reasoning: Review of propositional and predicate logic; resolution and theorem proving; non-monotonic inference; probabilistic reasoning; Bayes theorem
- Advanced search: Genetic algorithms; simulated annealing; local search
- Advanced knowledge representation and reasoning: Structured representation; non-monotonic reasoning; reasoning on action and change; temporal and spatial reasoning; uncertainty; knowledge representation for diagnosis, qualitative representation
- Agents: Definition of agents; successful applications and state-of-the-art agent-based systems; software agents, personal assistants, and information access; multi-agent systems
- Machine learning and neural networks: Definition and examples of machine learning; supervised learning; unsupervised learning; reinforcement learning; introduction to neural networks
- AI planning systems: Definition and examples of planning systems; planning as search; operator-based planning; propositional planning

Methods of Delivery:

3 hours of lecture per week.

Student Evaluation (typical):

Term paper, graded problem sets, 2 exams (midterm and final)

Resources to be purchased/provided by students:

Textbook, reference sources and internet access (see Textbook requirements below).

Textbook requirements:

The following is a list of representative text books appropriate for this course. The list is not exhaustive and the course instructor is encouraged to select a text appropriate for the focus they wish to place on the course. Many different approaches are possible and equally valid. The selection of a course text should reflect this premise.

- Baral, Chitta., Knowledge Representation, Reasoning, and Declarative Problem Solving, Cambridge, UK : Cambridge University Press, 2003., ISBN: 0521818028
- Copeland, Jack, Artificial intelligence: A Philosophical Introduction, Oxford ; Cambridge, Mass. : Blackwell, 1993., ISBN: 063118385X
- Desouza, Kevin C., Managing Knowledge with Artificial Intelligence: An Introduction with Guidelines for Nonspecialists, Westport, CT : Quorum Books, 2002., ISBN: 1567204910
- Ferber, Jacques., Multi-agent Systems : An Introduction to Distributed Artificial Intelligence, Harlow ; New York : Addison-Wesley, 1999, ISBN: 0201360489
- Grand, Steve., Creation : Life and How to Make It, Cambridge, MA : Harvard University Press, 2001., ISBN: 0674006542
- Hibbard, Bill., Super-intelligent Machines, New York : Kluwer Academic/Plenum Publishers, c2002., ISBN: 0306473887
- Luger, George F., Artificial Intelligence: Structures and Strategies for Complex Problem Solving (4th ed.), Harlow, UK. ; New York, Pearson Education, 2002., ISBN: 0201648660
- Russell, Stuart J., Artificial intelligence : A Modern Approach (2nd ed.), Upper Saddle River, N.J., Pearson Education, c2003., ISBN: 0137903952

Learning Outcomes.

Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: Identify the major historical trends in AI.
Outcome 2: Discuss current themes in AI.
Outcome 3: Discuss computing problem spaces with reference to AI techniques.
Outcome 4: Differentiate between AI techniques and understand their benefits and limitations.

Outcome 5: Discern the applicability of AI techniques to different computing problems.

Rationale: (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

The development of Artificial Intelligence within the computer science discipline has been fraught with both promise and profound disappointment. From the early days of Alan Turing many have sought mechanisms that mimic natural intelligence. Despite the failure to deliver on various predictions, many AI subjects have become main-stream computing techniques.

In order to appreciate the significance of discoveries in AI a student must be exposed to the inevitable reality of how discoveries were made and how they affect computer science today.

A lecture approach has been selected as it lends itself to the presentation of the "story" of AI. Like computer science itself, the field has matured, provided promising results that can be applied in other fields. It is suggested that a narrative approach lends itself well to communicating many of the introductory aspects of AI.

Information about Course Designer/ Developer

Course designed by Prof. Alexander Ferworn (Associate Professor of Computer Science, Ryerson University).

List faculty eligible to teach the course

Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course web site will be a key course component.

Faculty Qualifications to teach/supervise the course:

A Postgraduate degree with expertise in Artificial Intelligence.

Classroom requirements:

Technology-enhanced classroom with laptop connections, data projector, internet access

Equipment requirements:

None

Course Title: Human-Computer Interaction

Pre-Requisites: Software Engineering

Year and Semester: Year 4, Semester 2

- **Course Description and Content Outline**(by topic)**:**

This course is intended as an introduction to human-computer interaction (HCI), with emphasis being placed on understanding human behaviour with interactive objects, general knowledge of HCI design issues, and a human-centred approach to software design. The course will stress the design of usable interfaces including the consideration of cognitive factors and social contexts within which computer systems are used. Students will receive an introduction to HCI while applying this theory to a design project.

Topics discussed during this class include the following:

A. Introduction to HCI
    a. Motivation for human-centred design and evaluation.
    b. Human performance models: perception, movement and cognition.
    c. Design principles and usability heuristics
    d. Introduction to usability testing

B. Visual Interface Design
    a. Principles of Graphical User Interface GUI design
    b. GUI toolkits
    c. Choosing interaction styles and techniques
    d. HCI aspects of widgets
    e. HCI aspects of screen design: layout, colour, fonts.
    f. Handling human failure
    g. Beyond simple screen design: visualization, representation, metaphor
    h. Multimodal interaction: graphics, sound and haptics

C. Human Centred Software Design and Evaluation
    a. Overview of the design process
    b. Functionality and usability: task analysis, interviews, surveys
    c. Prototyping techniques and tools (storyboards, GUI builders)
    d. Goals for evaluation
    e. Evaluation without users: walkthroughs, guidelines and standards
    f. Evaluation with users: usability testing, interviews, surveys, experiments.

- **Methods of Delivery:** 2 hours of lecture per week (60%) and a 2 hour weekly interactive seminar/workshop (40%).

- **Student Evaluation (typical)**:

    - Small assignments applying practical skills to sample problems (20%).
        - e.g. User interface evaluation and redesign.
    - Examination on theoretical and practical knowledge of human-computer interaction (40%).
    - Group project applying HCI design criteria to a real-world application (40%)
        - Project proposal, design document, presentation.

- **Resources to be purchased/provided by students:** Textbook, reference sources and internet access (see Textbook requirements below).

- **Textbook requirements:** (illustrative texts and other course materials)

    - Mandel, T., 1997, The Elements of User Interface Design, John Wiley & Sons, ISBN: 0471162671
    - Cato, J., 2001, User-Centered Web Design, Addison-Wesley, ISBN: 0201398605
    - Nielsen, J., 1999, Designing Web Usability, New Riders, ISBN: 156205810X
    - Johnson, J., 2000, GUI Bloopers, Morgan Kaufmann, ISBN: 1558605827

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

  Outcome 1: Discuss the reasons for human-centred design.
  Outcome 2: Summarize the basic science of psychological and social interaction.
  Outcome 3: Develop a conceptual vocabulary for analyzing human interaction with software: e.g. conceptual model, feedback.
  Outcome 4: Identify the principles associated with effectual visual interface design.
  Outcome 5: Explain the basic types and features of human-centred design.
  Outcome 6: State three usability requirements.
  Outcome 7: Discuss the pros and cons of development with paper and software prototypes.
  Outcome 8: Explain good design principles for GUI components.
  Outcome 9: Design, prototype and evaluate a simple visual interface.

Outcome 10: Discuss evaluation criteria and conduct a usability test.

- **Rationale:**

It is expected that both the lecture and seminar/workshop will incorporate aspects of interactive collaborative learning.

Information about Course Designer/ Developer
Course designed by Dr. Michael Wirth (Dept. Computing and Information Science, University of Guelph)

List faculty eligible to teach the course: Faculty to be hired

Are there any plans to teach all or portions of this course on-line? A course website will play an integral role in the delivery of resources for this course: syllabus, schedule, assignments, solutions to assignments, handouts, supplementary notes, etc.

Faculty Qualifications to teach/supervise the course: Postgraduate degree with experience in human computer interaction.

Classroom requirements: Technology-enhanced classroom with networking, internet and data projection capabilities.

Equipment requirements: None

Course Title: High Performance Computing

Pre-Requisites: Software Engineering, Operating Systems and Networking, Simulation and Modelling

Year and Semester: Year 4, Semester 1 or 2

- **Course Description and Content Outline** (by topic)**:** This course allows the student to explore issues in high performance computing specifically in the areas of parallel software design and programming. The major paradigms of parallel architectures and parallel complexity will be covered. Topics covered include:
    - Current trends in High Performance Computing (grid computing, etc.)
    - Parallel programming models
    - Parallel programming with MPI
    - Designing parallel systems
    - Efficiency and debugging
    - Performance analysis and profiling
    - Parallel complexity theory
    - Applications in Scientific Computing

- **Methods of Delivery:** 3 hours of lecture per week plus a 1 hour weekly tutorial/seminar.

- **Student Evaluation**:

  Assignment Section (30%): 3 individual assignments in any of the following areas:
  Project Section (40%)
    - Individual project demonstrating the student's ability to use a high performance computing environment to develop a scientific computing application.

  Examination Section (30%)
    - Midterm: 10%
    - Final Examination: 20%

- **Resources to be purchased/provided by students:** textbooks, account on the SHARCNET HPC facility (no cost)

- **Textbook requirements:**
    - High Performance Cluster Computing by Rajkumar Buyya, Prentice Hall, 1999
    - MPI – The Complete Reference (Volume 1, 2nd edition) by Snir, Otto, Huss-Lederman, Walker and Dongarra, MIT Press, 1999.

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the following abilities:
  Outcome 1: Mature approach to the design of parallel systems.
  Outcome 2: Advanced knowledge of the MPI API.
  Outcome 3: Knowledge of the breadth of HPC technologies and paradigms in current use.
  Outcome 4: Knowledge of leading-edge technologies in the domain of grid computing.
  Outcome 5: Introduction to different ways of approaching the design of parallel software including the experience of using a current leading-edge paradigm in both tutorial exercises and in the project.
  Outcome 6: Experience in using the HPC equipment in the SHARCNET HPC consortium of South-Western Ontario.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Outcomes 1, 2, 3 and 4 will be developed through the lectures. The lecture material will prepare the student for the assignments and the project as will individual consultation between the students and the instructor over the choice of term project. This consultation will be augmented by personnel from the SHARCNET HPC consortium who will aid the instructor in the delivery of the course.

Outcomes 5 and 6 will be developed through the use of the HPC equipment available through SHARCNET. Students of this course will be able to develop on state-of-the-art HPC equipment and will be exposed to the practical issues involved in working on HPC systems This experience is not generally available to most computing science and/or science undergraduates in Canada.

Information about Course Designer/ Developer
Course designed by Deborah Stacey, Professor of Computing and Information Science, University of Guelph and David McCaughan, SHARCNET@Guelph.

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course website will be a key resource for the students of this course.

Faculty Qualifications to teach/supervise the course:

The faculty member should have knowledge of the principles of parallel computation and be familiar with the SHARCNET HPC facilities. Practical experience with MPI programming would also be an asset.

Classroom requirements: Some of the seminar hours must be in a computing environment where all of the class can have access to workstations.

Equipment requirements:

Appropriate workstations and software must be available to the class for their individual assignments and project. The students must be able to use the lab equipment to ssh to the SHARCNET facilities.

Course Title: Distributed Computing

Pre-Requisites: Software Engineering and Operating Systems and Networking

Year and Semester: Year 4, Semester 1 or 2

- **Course Description and Content Outline** (by topic)**:** This course exposes the student to the major paradigms of distributed computing, from sockets to client/server to web services and grid computing. Topics covered include:
    - Distributed Computing Paradigms and Models
    - Distributed Databases and Storage Issues
    - Security (including encryption, certificates, attacks, authentication, authorization, digital signatures, firewalls, access control lists, capability access)
    - Internet issues: name services, DNS
    - Web Services
    - Grid computing: Globus, OGSA
    - Peer to Peer
    - Project Management Issues in Distributed Computing
    - Testing and Performance Issues
    - Design Issues including in depth coverage of techniques such as sockets, threads, Java RMI, Corba, Tomcat, servelets, and Globus.

- **Methods of Delivery:** 3 hours of lecture per week plus a 2 hour weekly tutorial/seminar.

- **Student Evaluation**:

  Assignment Section (30%): 3 or 4 individual assignments in any of the following areas:
    - Client/Server using sockets and threads first and then converting to another technology
    - Corba
    - Web Services
    - Globus

  Project Section (40%)
    - Team project using a specific distributed paradigm and technology.

  Examination Section (30%)
    - Midterm: 10%
    - Final Examination: 20%

- **Resources to be purchased/provided by students:** textbooks

- **Textbook requirements:**
    - Distributed Computing with Java by Qusay H. Mahmoud, Manning, 2000
    - Distributed Systems: Concepts and Design, 3rd edition, Addison-Wesley, 2001

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the following abilities:
  Outcome 1: Mature approach to the design of distributed systems..
  Outcome 2: Advanced knowledge of Java distributed programming techniques.
  Outcome 3: Knowledge of the breadth of distributed technologies and paradigms in current use.
  Outcome 4: Knowledge of security issues.
  Outcome 5: Knowledge of leading-edge technologies in the domain of web services and grid computing.
  Outcome 6: Introduction to different ways of approaching the design of distributed software including the experience of using a current leading-edge paradigm in both tutorial exercises and in the team project.

Outcome 7: Knowledge of the use and analysis of software metrics and testing with respect to distributed systems.
Outcome 8: Introduction to the complexities involved in producing secure software systems.
Outcome 9: Ability and experience in serious teamwork situations.

- **Rationale:** (explanation of how learning outcomes will be enhanced by the method(s) of delivery)

Outcomes 1, 6, 8, and 9 will be developed through the use of a team project. The students will have already experienced team projects in their software engineering courses and so this course will advance these skills with regards to a particular type of computing system.

Outcomes 2, 3, 4, 5 and 7 will be developed through lectures and seminar sessions that include tutorials on specific software tools and environments and APIs, and specially designed exercises in the design and use of techniques for the development of distributed systems.

Information about Course Designer/ Developer
Course designed by Deborah Stacey, Computing and Information Science, University of Guelph

List faculty eligible to teach the course Faculty to be hired

Are there any plans to teach all or portions of this course on-line?

A course website will be a key resource for the students of this course.

Faculty Qualifications to teach/supervise the course:

The faculty member should have knowledge of the basic principles of distributed computing with regards to system and software aspects. Specific knowledge of Java's distributed facilities is also needed. The ability and experience to manage and evaluate large team projects would be an asset.

Classroom requirements: Some of the seminar hours must be in a computing environment where all of the class can have access to workstations.

Equipment requirements:

Appropriate workstations and software must be available to the class for their individual assignments and team project. Software includes an object-oriented language compiler, program development environment, and necessary software for team project (this may change from offering to offering). The lab environment should facilitate distributed system development (servers, database servers, firewall software, etc).

Appendix B:

Course Outlines for Existing UOIT Courses in Program

Course Outlines for Existing UOIT Computer Science Courses

COURSE TITLE: **FUNDAMENTALS OF PROGRAMMING (CSCI1600U)**
Pre-requisite: Scientific Computing Tools
Year and Semester: Year 1, Semester 2

- **Course Description and Content Outline:** This course provides a basic introduction to computer programming using the C programming language. Topics include basic computer hardware and software concepts, problem analysis, design of algorithms and programs, the basic principles of object-oriented languages.

- **Delivery Mode and Teaching Method (s):** This one semester course will be delivered as 3 hours of classroom lectures and 2 hrs tutorial weekly.

- **Student Evaluation**: Students will be evaluated using a combination of tests, assignments, and a final examination. The exact weighting of the various components will be determined by the professor hired to teach the course and presented to the students in the first week of classes.

- **Resources to be purchased by students:** to be identified by professor(s) hired to teach the course

- **Textbook requirements:** to be identified by professor(s) hired to teach the course

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:
  Outcome 1: explain the main hardware components of computers.
  Outcome 2: describe how the various hardware devices and software programs interact to perform the basic functions of a personal computer
  Outcome 3: explain how application programs manipulate data to produce the desired results.
  Outcome 4: understand how computers communicate with each other, including local and wide area networks, the Internet and Intranets
  Outcome 5: understand the notion of an algorithm as a problem solving technique
  Outcome 6: describe the main characteristics and benefits of structured programming
  Outcome 7: write and debug programs to achieve specified outcomes
  Outcome 8: understand the basic concepts of object-oriented programming

- **Rationale:** Lectures are designed to ensure that students understand the concepts and techniques of computer programming using an object-oriented approach.

Information about Course Designer/ Developer
Course designed by Wm. Smith

List faculty eligible to teach the course: Dr. G. Lewis (School of Science)

Are there any plans to teach all or portions of this course on-line? No

Faculty Qualifications to teach/supervise the course: Ph.D. (preferred) or postgraduate degree in Mathematics or Computing Science

Classroom requirements: technology-enhanced classroom with laptop connections, data projector, internet access

Equipment requirements: Access to an appropriate software environment

COURSE TITLE: PRINCIPLES OF COMPUTER SCIENCE (CSCI2010U)
Pre-requisite: Fundamentals of Programming
Year and Semester: Year 2, semester 1

- **Course Description and Content Outline:** This course introduces students to general computer programming principles and the analysis of algorithms and data structures. Topics include problem analysis, design of algorithms and programs, the selection of data types, decision-making, and program correctness. The course uses an object-oriented programming language such as Java or C++. Applications to business, science and engineering are illustrated. The focus is on the effective choice of algorithm and data structure and the use of a disciplined programming style which permits programs to be understood and read by others.

- **Delivery Mode and Teaching Method (s):** This one semester course will be delivered using three hours of lectures and one hour of tutorials/laboratory sessions per week.

- **Student Evaluation:** Assignments/projects will be given every two to three weeks in the form of knowledge testing questions, analysis and design skill testing problems, and software projects (analysis, design and programming) to test the student's ability to integrate various aspects of knowledge that are taught in the course and apply them to solving business, science and engineering needs. Assessment of these assignments will form part of the final mark for the course. Tests which may have an open-book component, will evaluate the student's progress in grasping and applying the material. The final examination will generally be closed-book, testing the analytical, design, problem-solving and domain-application skills of students. Presentations may form a part of the final mark.

.

- **Resources to be purchased by students:** to be identified by professor(s) hired to teach the course

- **Textbook requirements:** to be identified by professor(s) hired to teach the course

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: understand and implement nontrivial programs using suitably chosen data structures and algorithms
Outcome 2: analyze business, science or engineering problems, design solutions in terms of data structures and algorithms, and realize solutions by writing, testing and documenting computer programs
Outcome 3: choose among alternatives for data structures and algorithms, estimate their time and space complexity, and analyze time-space tradeoffs
Outcome 4: use pre-defined routines, and write and execute procedures and functions
Outcome 5: evaluate the correctness of programs, and test for boundary conditions.
Outcome 6: understand and use object-oriented concepts, including objects, classes, inheritance, methods, encapsulations and exceptions.
Outcome 7: relate programming techniques to addressing physical constraints, and meeting organizational needs.

- **Rationale:** Lectures are designed to ensure that students understand the concepts and techniques of programming languages and their application to solving problems. The tutorials and/or laboratory sessions are aimed at providing a forum for solving problems, gaining hands-on experience, and relating software techniques to the meeting of scientific needs.

- **Information about Course Designer/ Developer:** Dr. Wm. Smith, Dean of Science

- **List faculty eligible to teach the course:** Dr. G. Lewis (School of Science)

PRINCIPLES OF COMPUTER SCIENCE continued…

- **Are there any plans to teach all or portions of this course on-line?** No
- **Faculty Qualifications to teach/supervise the course:** Ph.D. in Computer Science and experience with programming languages in an academic, research or industrial setting.

COURSE TITLE: **COMPUTATIONAL SCIENCE I (MATH2070U)**
Pre-requisite: Calculus II **(**MATH 1020U), Linear Algebra (MATH2050U)
Year and Semester: Year 2, semester 2

- **Course Description and Content Outline:** This course provides an overview of and practical experience in utilizing algorithms for solving numerical problems arising in the applied sciences. Topics covered include computer arithmetic, solution of a single nonlinear equation, interpolation, numerical integration, solution of differential equations, and systems of linear equations. Students will use Maple, MATLAB, or other appropriate software to solve problems and complete assignments.

- **Delivery Mode and Teaching Method (s):** This one semester course will be delivered through 3 hours of classroom lectures and a1 hour computer laboratory weekly.

- **Student Evaluation**: The students will complete four assignments each worth 5%, four written tests each worth 15%, and a 20% final exam. Students will be evaluated using a combination of tests, assignments, and a final examination.

- **Resources to be purchased by students:** to be identified by professor(s) hired to teach the course

- **Textbook requirements:** to be identified by professor(s) hired to teach the course

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: Work with floating-point number systems in base 10 and base 2.
Outcome 2: Investigate elementary error analysis, pitfalls of computer arithmetic, and \ quantify error using “O” notation.
Outcome 3: Approximate fixed points and roots of equations using a variety of iterative methods, find the order of convergence, and quantify error.
Outcome 4: Use the Lagrange Interpolating Polynomial and divided differences to generate interpolating polynomials.
Outcome 5: Derive and apply the formulas for piecewise polynomials including clamped and natural cubic spline interpolation.
Outcome 6: Derive and apply numerical integration and composite integration formulas, and quantify error.
Outcome 7: Derive and apply Euler’s Method and low-order Runge-Kutta Methods to solve initial value problems and quantify error.
Outcome 8: Approximate solutions to systems of linear algebraic equations, using pivoting strategies to minimize error.

- **Rationale:** Lectures are designed to ensure that students understand the concepts and techniques of numerical methods, with emphasis on theory, geometric interpretation, and practical examples. In the laboratory sessions students will write and run programs in Maple, MatLab or other software to apply the techniques under study.

- **Information about Course Designer/ Developer:** Dr. Wm. Smith, Dean of Science

- **List faculty eligible to teach the course:** Dr. G. Lewis, Dr. M. Staley, Dr. W. Smith (School of Science)

- **Are there any plans to teach all or portions of this course on-line?** No

- **Faculty Qualifications to teach/supervise the course:** Ph.D. (preferred) or postgraduate degree in Mathematics.

COURSE TITLE: SIMULATION AND MODELING (CSCI3010U)
Pre-requisite(s): Fundamentals of Programming (CSCI1600U), Computational Science I
Year and Semester: Year 3, semester 1

- **Course Description and Content Outline:** This course provides a basic introduction to simulation and modeling. The goal is provide the student with an appreciation of the role of simulation in various scientific, engineering, and business fields, and to provide some experience in writing simulation programs.

- **Delivery Mode and Teaching Method (s):** This one semester course will be delivered as 3 hours of classroom lectures and 2 hours tutorial weekly.

- **Student Evaluation**: Students will be evaluated using a combination of tests, assignments, and a final examination.

- **Resources to be purchased by students:** to be identified by professor(s) hired to teach the course

- **Textbook requirements:** to be identified by professor(s) hired to teach the course

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: Explain the role of simulation in different areas of science and business.
Outcome 2: Understand the role of object-oriented programming in simulation.
Outcome 3: Understand how to measure the sensitivity of simulation results to changes in modeling assumptions.
Outcome 4: Use the results of simulation to aid in discovering laws of nature.
Outcome 5: Understand the use of simulation as a tool for risk management.
Outcome 6: Implement a project based on simulation of a problem from an application area

- **Rationale:** Lectures are designed to ensure that students understand the concepts and techniques of simulation.

- **Information about Course Designer/ Developer:** Dr. Wm. Smith, Dean of Science

- **List faculty eligible to teach the course** Faculty to be hired

- **Are there any plans to teach all or portions of this course on-line?** No

- **Faculty Qualifications to teach/supervise the course:** Ph.D. (preferred) or postgraduate degree in Mathematics or Computing Science

COURSE TITLE: SCIENTIFIC VISUALIZATION AND COMPUTER GRAPHICS (CSCI 3020U)
Pre-requisite(s): Computational Science I (MATH 2070U), Principles of Computer Science (CSCI 2000U)
Year and Semester: Year 4, semester 1 of Computer Science program (may be taken in Year 3)

- **Course Description and Content Outline:** This course provides a basic introduction to computer graphics and scientific visualization. Basic properties of display devices, graphics objects, and common graphics operations will be identified. The use of colour, texture, lighting and surface/contour plots will be surveyed. Examples from modeling of PDEs will be presented.

- **Delivery Mode and Teaching Method (s):** This one semester course will be delivered as 3 hours of classroom lectures and 2 hours tutorial weekly.

- **Student Evaluation**: Students will be evaluated using a combination of tests, assignments, and a final examination.

- **Resources to be purchased by students:** to be identified by professor(s) hired to teach the course

- **Textbook requirements:** to be identified by professor(s) hired to teach the course

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: ability to make effective use of visualization tools available in modern problem solving environments (PSEs).
Outcome 2: an understanding of the trade-offs in accurately rendering a graphics object on a high resolution display device.
Outcome 3: an understanding of the cost of adding colour, texture and lighting to a visualization.
Outcome 4: familiarity with standard graphics systems and packages.
Outcome 5: familiarity with techniques for displaying a small subset of the results from a large scale simulation
Outcome 6: an understanding of the need for scalability and parallelism when simulations generate massive data sets.

- **Rationale:** Lectures are designed to ensure that students understand the concepts and techniques of computer graphics and scientific visualization.

- **Information about Course Designer/ Developer:** Dr. Wm. Smith, Dean of Science

- **List faculty eligible to teach the course** Faculty to be hired

- **Are there any plans to teach all or portions of this course on-line?** No

- **Faculty Qualifications to teach/supervise the course:** Ph.D. (preferred) or postgraduate degree in Mathematics or Computing Science

COURSE TITLE: COMPUTATIONAL SCIENCE II (MATH4020U)
Pre-requisite(s): Computational Science I (MATH2070U), Scientific Visualization and Computer Graphics (CSCI3020U)
Year and Semester: Year 4 elective

- **Course Description and Content Outline:** This course provides a variety of results and algorithms from a theoretical and practical point of view. Students study approximation of functions; quadrature ; numerical solution of ordinary differential equations; the algebraic eigenvalue problem. Maple, MatLab or other software will be used in assignments.

- **Delivery Mode and Teaching Method (s):** This one semester course will be delivered through 3 hours of classroom lectures per week.

- **Student Evaluation**:Five assignments will be given. They will consist of knowledge testing questions, analysis skill testing problems, and applications. Tests will evaluate the students' progress in grasping and applying the material. The final examination will test both retention of important facts and the analytical skills of the students.

- **Resources to be purchased by students:** to be identified by professor(s) hired to teach the course

- **Textbook requirements:** to be identified by professor(s) hired to teach the course

- **Learning Outcomes.** Students who successfully complete the course have reliably demonstrated the ability to:

Outcome 1: Be able to choose the interpolatory points which minimize the error in Lagrange interpolation and use these to find the Lagrange polynomial.
Outcome 2: Demonstrate familiarity with Newton's Interpolatory Divided-Difference Formula; and Hermite Interpolation.
Outcome 3: Be familiar with data fitting and linear least squares including the use of Normal equations and the QR algorithm.
Outcome 4: Be able to apply numerical integration methods and use the method of Gaussian Quadrature.
Outcome 5: Demonstrate familiarity with different numerical methods available for initial value problems for ordinary differential equations, including Runge-Kutta methods .
Outcome 6: Use the finite element method for certain partial differential equations.
Outcome 7: Use the power method and the iterative QR method to approximate the eigenvalues of a matrix. Be familiar with the SVD decomposition of a matrix and its applications.
Outcome 8: Using the course material, identify and solve a variety of problems in diverse areas of mathematics.

- **Rationale:** Lectures are designed to ensure that students understand the concepts and techniques and applications of numerical analysis.

- **Information about Course Designer/ Developer:** Dr. Wm. Smith, Dean of Science

- **List faculty eligible to teach the course:** Dr. G. Lewis, Dr. W. Smith (School of Science)

- **Are there any plans to teach all or portions of this course on-line?** No

- **Faculty Qualifications to teach/supervise the course:** Ph.D. (preferred) or postgraduate degree in Mathematics.

Appendix C:

A Layman's Guide to Computing Programs and Terms

(Sept. 15,2003; by W.R. Smith, UOIT, with assistance from Deb Stacey, Un. Of Guelph; Wayne Enright, Un of Toronto; Mike Bauer, Un. Of Western Ontario; Mark Staley, UOIT)

Computing/Computer Science (CS):

This discipline is concerned with the study of the basis for and use of computational devices. The discipline examines issues pertaining to the organization of computer hardware, the design, implementation and maintenance of software, the interface between the hardware and the software and with the human user, and the mathematical and formal principles that underlie computation.

Canadian university programs in CS are usually associated with Faculties of Science.

Computer Engineering (CE):

This discipline is concerned with the design of computing devices and systems. It has traditionally been viewed as "hardware-oriented", as opposed to the more "software-oriented" CS discipline. This distinction has become more blurred in recent years, as Engineering programs have introduced more software design into their programs (with the cooperation of Computer Science departments).

The principles of circuit design, board design and system integration are studied, as well as the firmware necessary to connect the hardware to the software. CE students also study the uses of computing systems (including software), but not at the same level as those studying CS or Software Engineering (SE).

Canadian university programs in CE are often associated with Electrical Engineering departments and programs.

(the following is a paraphrase of material on the web site http://feynman.ee.ualberta.ca/ce/ce.vs.cs.html at the Un. of Alberta):

The difference between Computer Science and Computer Engineering is that the former focuses on algorithms and data structures, whereas the latter encompasses hardware, software, project management, etc.: taking a concept and turning it into a product, just like other engineering areas. So, for instance, a computer science person would investigate artificial intelligence algorithms in mathematical detail, but a computer engineer would design the whole system that includes this artificial intelligence component. The engineer probably wouldn't be as interested in the A.I. algorithm details. It's similar to the distinction between electrical engineers and device physicists.

Information Technology:

(from the web site http://whatis.techtarget.com)

IT (information technology) is a term that encompasses all forms of technology used to create, store, exchange, and use information in its various forms (business data, voice conversations, still images, motion pictures, multimedia presentations, and other forms.

Information Technology practitioners typically emphasize the application of computer software related to these forms of information in a business or social science setting. Studies involve the industrial, business and social deployment of computing/information technology.

Students in this area are rarely extremely "technical". They are more concerned with the uses of the technology in a particular setting, and their level of design expertise generally concentrates on the *use of* software packages and the assembly of software components to accomplish a particular task.

University programs in Information Technology are often associated with Business schools and departments.

Communications Engineering:

This discipline is not very close to CS; its focus is typically on the design of networks and communications systems, while most CS people are concerned with protocols for the use of communications equipment but not in their physical design. Some CS work is done in the simulation of communications networks for the purposes of identifying optimal layout and protocol design and evaluation.

University programs are often associated with Electrical Engineering departments and programs.

The Canadian Software Engineering Debate:

"Software engineering" is a politically charged term with a recent and interesting history (see below). Carleton University's web site, http://www.scs.carleton.ca/se/ states:

Software Engineering has been defined in many ways. It is the branch of Computer Science that develops and uses techniques to design and build correct, reliable, and secure software that is delivered on time and within budget.

The "Great Canadian Debate" concerning Software Engineering began when Memorial University's Department of Computer Science introduced a Software Engineering Program outside the Faculty of Engineering, in 1998. In early 1999, The Association of Professional Engineers and Geoscientists of Newfoundland (APEGN) subsequently engaged in a court battle to revoke Memorial University's accreditation for all its engineering programs; this attempt failed. This led to extensive consultations involving the Canadian Council of Professional Engineers (CCPE) and the Canadian Information Processing Society (CIPS) - a "low-power" Canadian version of the ACM – Association for Computing Machinery, concerning the accreditation of Canadian university programs in software engineering.

The concern with the use of the term "engineering", both in general and within the term "software engineering", appears to be a particularly Canadian phenomenon. The CCPE maintains that it is simply ensuring that the public is protected, by insisting that only licensed (by them) engineers use the term "engineer". The term "software engineering" itself arose in the US; a single engineering accreditation body with comparable influence in Canada to the CCPE does not apparently exist, and there is little or no similar controversy in the US. The US-originated term "Microsoft Certified Software Engineer" (MCSE) has similarly caused great consternation to the CCPE when used in Canada. (Interestingly, the commonly-used term "genetic engineering" has not caused similar concern.)

There are currently two accreditation committees that have processes in place for approving software engineering programs: the CSAC (Computer Science Accreditation Council) for Computer Science, which is affiliated with CIPS, and the CEAB (Canadian Engineering Accreditation Board), which is the engineering one. There is still a fair bit of controversy between the accreditation processes of the two competing bodies. In September 2000, the two accrediting bodies jointly drafted an accreditation plan for a proposed new joint accreditation board, a recommendation that arose from a Committee chaired by former University of Waterloo President James Downey following the Memorial case. However, in February 2002, the CEAB recommended and approved a final report containing a number of significant changes to the previously jointly approved accreditation process without consulting with the CSAC. The CSAC and the AUCC (Association of Universities and Colleges of Canada), which had sided with Memorial, rejected the CEAB process.

According to a Canada Newswire Release of June 11, 2002, CIPS is urging the CCPE to work amicably with the IT community in resolving the dispute over the usage and practice of "software engineering". CCPE, on the other hand, has threatened to take legal action against

those they declare are practicing “software engineering” without a license because it claims that “software engineering” is a branch of engineering and not computer science.

- For an interesting view of Software Engineering (SE) accreditation check out the following Canadian Association of Computer Science (CACS) webpages:
  http://www.cs.usask.ca/spec_int/cacs/page14.htm
  http://www.cs.usask.ca/spec_int/cacs/page13.htm

- University Affairs also has an interesting background article on the debate:
  http://www.universityaffairs.ca/pdf/past_articles/2001/oct/pg42.pdf

- Dave Calvert and Deb Stacey of the University of Guelph’s Department of Computing and Information Science debated an engineer about the place of software engineering in UoG's @Guelph:
  http://www.uoguelph.ca/atguelph/99-08-11/insight.html

The CIPS 2002-3 Annual Report (http://www.cipsar.ca/university.htm) states that the following programs were successfully reviewed against CSAC's software engineering accreditation criteria:
* University of Waterloo: Honours Computer Science/Software Engineering Option
* University of Western Ontario: Honours B. Sc. in Computer Science with Software Engineering Specialization and B.Sc. in Computer Science with Software Engineering Specialization

Software Engineering programs in Engineering faculties accredited by CEAB include:
* Calgary: 2002-
* Carleton: 2003-
* Concordia: 2002-
* Lakehead: 2002-
* McMaster: 2001-
* Ottawa: 2001-
* Western Ontario: 2001-

The situation has not been resolved to date, although some cooperative trends appear to be emerging. In the face of the stalemate, individual universities have gone ahead in a local fashion to develop software engineering programs and software engineering streams based both in Computer Science and in Engineering.

The most successful SE programs appear to be those run with some cooperation/collaboration, but lead to two different degrees with a “software engineering” designation - one in Engineering and the other through a Computer Science Department located in either Science or Math. Examples include Calgary, Carleton, Western Ontario, and Waterloo.

Appendix D:

Computing Programs at Canadian Universities

D. Stacey, Un. Of Guelph; W. Smith, UOIT;

Although the location of Software Engineering Programs is the subject of some debate in Canada, for Computing (or Computer) Science itself, the situation is much clearer. Very few CS programs are solely in Engineering; the notable exceptions are Victoria and Concordia. Some of the major CS schools are listed below, with extracts from their web sites.

Some Major Canadian CS Schools (web site extracts in italics):

University of Alberta:

The Department of Computing Science was formed on April 1, 1964. From the beginning the Department had a strong orientation towards the mathematical disciplines rather than engineering or business. The choice of "Computing Science" instead of the more common "Computer Science" as the name of the Department was deliberate and was intended to indicate computing rather than computers as the foundation of the discipline.

University of British Columbia:

Computing Science is located in the Faculty of Science. There is also a Software Engineering stream or option.

Software engineering has gained significant momentum in the last five years. The demand for graduates with thorough knowledge of the software development methods and practices has increased tremendously. In response to such demand, many Universities in North America and Europe have designed new Software Engineering specializations for their bachelor degrees or have created software engineering options for their programs. That is why we are implementing Software Engineering options for our Major and Honours programs. The intent of each option is two-fold. First, the options offer a coherent set of courses pertaining to software engineering that will provide our undergraduate students with the necessary knowledge for a successful career in the software industry. Second, the options would enable a software engineering designation to be added to the student's degree. This designation may aid the student in the pursuit of their career choices.

University of Calgary:

Computer Science at the University of Calgary is a department of the Faculty of Science. Prior to 1975, we were part of the Department of Mathematics, Statistics and Computer Science (now the Department of Mathematics and Statistics).

Carleton University:

Carleton offers both a Bachelor of Computer Science degree in the School of Computer Science, located administratively within the Faculty of Science, and a Software Engineering Program within the Faculty of Engineering and Design. Within the Computer Science program, it offers a Software Engineering stream.

Of some relevance to UOIT situation, beginning in Fall, 2003, Carleton offers an Information Systems Security stream within its Computer Science program. This is based on a Computer Science program with 2 additional courses: COMP4108 (Computer Systems Security) and COMP4109 (Applied Cryptography)

Concordia University:

Computer Science is located in the Faculty of Engineering and Computer Science. Programs are available in Computer Science and in Software Engineering.

Lakehead University:

Lakehead offers a BSc in Computer Science, with options in Business, Hardware, and Science. Computer Science is located in the Faculty of Science and Environmental Studies. Lakehead also offers a Software Engineering degree in the Faculty of Engineering, which it states on its web site is *developed in collaboration with the Department of Computer Science.*

Memorial University (Newfoundland):

Computer Science is in the Faculty of Science.

McGill University:

The university offers a new B.Sc. Major program in Software Engineering. This B.Sc. program is offered through the Faculty of Science. The program is closely related to the one in computer science, but puts more emphasis on software development related activities, such as programming techniques, object-orientation, software processes (requirements engineering and analysis, design), project planning, quality assurance, and fault tolerance. Students choosing the software engineering option will also do more practical work, for example participate in larger group projects.

In addition to the B.Sc. Major program in Software Engineering offered by the School of Computer Science, a Bachelor of Software Engineering (B.S.E.) program is offered through the Faculty of Engineering.

Queen's University

The School of Computing is located in the Faculty of Arts and Science. A Computer Engineering program is offered in the Dept of Computer and Electrical Engineering.

Simon Fraser University:

Computing Science is in the Faculty of Applied Sciences, which brings together Computing Science, Kinesiology, Communications, and Natural Resource Management (now Resource and Environmental Management) from the Faculty of Interdisciplinary Studies and Engineering Science (previously in a separate Faculty). Computing Science is within a School of Computing Science and has a Director.

University of Toronto:

The Department of Computer Science at the University of Toronto has been offering courses in "software engineering" since the early 70's and a BSc in Computer Science with a Specialty in software engineering since the early 90's. Currently, one can specialize in the SE stream of Computer Science or take a BEng in SE in the Department of Electrical and Computer Engineering. The required courses in these programs are jointly designed and the lectures are shared between members of the two departments.

University of Victoria:

Computer Science and Software Engineering are in the School of Engineering.

University of Waterloo:

Computer Science is located in the Faculty of Mathematics. A degree is offered in B. Computer Science/Software Engineering Option.

Software engineering is a systematic and disciplined approach to developing software. It applies both computer science and engineering principles and practices to the creation, operation, and maintenance of software systems. At the University of Waterloo, Software Engineering is an independent, interdisciplinary program supported by both the Faculty of Mathematics and the Faculty of Engineering. Graduates of this program will earn a Bachelor of Software Engineering (BSE) degree.

University of Western Ontario

Computer Science is located in the Faculty of Science, and Computer Engineering is offered by Electrical and Computer Engineering Department in the School of Engineering.

Overall Observations:

It appears to be clear that

- *Computing Science is generally seen as being in a Faculty of Science*
- *Computer Engineering is in a Faculty of Engineering (often associated with an Electrical Engineering Department)*
- *Software Engineering is best done as a collaboration between Computing Science and Engineering.*