Proposed University of Victoria
Bachelor of Software Engineering (BSENG)
Degree Program

Detailed Course Descriptions

V4.5—August 29, 2002

Introduction

This document contains the detailed course descriptions of the proposed University of Victoria *Bachelor of Software Engineering (BSENG) Degree Program*. To get an overview of the courses, please consult the *BSENG Formal Degree Program Proposal*, the *BSENG overview matrix* and the *BSENG prerequisite structure*. To gain insight on how the BSENG curriculum was developed, please consult the *BSENG Curriculum Executive Summary*.

This document consists of two major parts: Part I contains the detailed courses descriptions; Part II contains units of knowledge from different areas of software engineering, computing, and engineering. However, Part II is incomplete and work in progress.

These course descriptions are a combination of University of Victoria calendar descriptions (i.e., SENG, CSC, CENG, and ELEC) IEEE/ACM Computing Curricula 2001 descriptions (i.e., Steelman Draft, August 1, 2001), revised course descriptions, and new course descriptions.

The knowledge units should help in assessing the completeness of the program as well as to calculate the number of accreditation units for mathematics, basic sciences, engineering sciences, engineering design, and complementary studies. This document will eventually have a mapping from knowledge units to accreditation units.

Table of Contents

Required courses

SE 5: Requirements Engineering and Formal Specification
SE 6: Software Evolution
SE 7: Embedded Systems
SE 8: Software Quality Engineering
MECHSYS: Mechanical Systems for Engineers
ELECSYS: Electrical Systems for Engineers
CAS: Computer Architecture and Assembler Programming
ALG 1: Algorithms and Data Structures
DD: Digital Design
AFL: Automata Theory and Formal Languages
CAP: Capstone Project
LA: Linear Algebra (Matrix Algebra for Engineers)
WE: Web Engineering
PS: Introduction to Probability and Statistics
HCI: Human-Computer Interaction
OSDC: Operating Systems and Distributed Computing
DB: Databases
NET: Networks
RT: Real-time Systems
CALC 1: Calculus 1
CALC 3: Calculus 2
SYSDYN: System Dynamcis
SAS: Signal and Systems
CTRL: Control Systems
SEC: Security Engineering
DS 1: Discrete Structures 1
DS 2: Discrete Structures 2
BUS 1: Engineering Economics and Entrepreneurship
BUS 2: Engineering Planning and Management
SOCIAL: Social and Professional Issues

Engineering Electives

ARCH: Software Architecture
CBSE: Component-Based Software Engineering
FTC: Fault Tolerant Computing
CSCW: Computer-Supported Collaborative Work
CG: Computer Graphics
MMS: Multimedia Systems
ALG 2: Algorithms and Data Structures 2
ALG 3: Algorithms and Data Structures 3
CON: Concurrency
CC: Compiler Construction
PL: Programming Languages
AI: Artificial Intelligence
PATREC: Pattern Recognition

Part I: Detailed Course Descriptions

Required Courses

SE 1: Object-Oriented Programming
Introduces the fundamental concepts sequentialand object-oriented programming. Through the study of object design, this course also introduces the basics of human-computer interfaces, and the social implications of computing, along with significant coverage of software engineering.
Related UVic Course: CSc 110
Prerequisites: Students should have sufficient facility with high-school mathematics to solve simple linear equations and to appreciate the use of mathematical notation and formalism. The course progresses rapidly and is intended for engineering or science students.
Syllabus:
• Background: History of computing, overview of programming languages and environments, software engineering
• Introduction to object-oriented programming using the object-oriented language Java; classes and objects; syntax of class definitions; methods; members
• Encapsulation, data and functional abstraction
• Simple data: variables, types, and expressions; assignments
• Message passing: Simple methods; parameter passing mechanisms
• Control structures: Iteration; conditionals
• Software engineering issues: Tools; processes; requirements; design and testing; risks and liabilities of computer-based systems
• Algorithms: Problem-solving strategies; the concept of an algorithm; properties of algorithms; implementation strategies
• Simple data structures: Arrays; strings
• Subclassing and inheritance
• Collection classes and iteration protocols (iterators)
• Assertions and testing strategies

• Introduction to reusable knowledge in software: algorithms, libraries, design patterns, components, architectures
• Introduction to design patterns: exceptions and iterators
• Using APIs: Class libraries; core language packages, packages for graphics and GUI applications
• Object-oriented design: Fundamental design concepts and principles
Units covered:
PF1 Fundamental programming constructs 7 core hours (of 9)
PF2 Algorithms and problem-solving 2 core hours (of 6)
PF3 Fundamental data structures 3 core hours (of 14)
PF4 Recursion 2 core hours (of 5)
AL3 Fundamental computing algorithms 3 core hours (of 12)
PL4 Declarations and types 3 core hours (of 4)
PL5 Abstraction mechanisms 1 core hour (of 3)
PL6 Object-oriented programming 8 core hours (of 10)
GV1 Fundamental techniques in graphics 2 core hours
SP1 History of computing 1 core hour
SP5 Risks and liabilities of computer-based systems 1 core hour (of 2)
SE1 Software design 2 core hours (of 8)
SE2 Using APIs 1 core hour (of 5)
SE3 Software tools and environments 1 core hours (of 3)
Elective topics 2 hours
Notes:
This course introduces the fundamental concepts of sequential and object-oriented programming, starting from the very beginning with the object-oriented paradigm. This course introduces first and foremost object-*based* programming and traditional control structures should not be neglected in the process. Note that the sequential concepts are as important as the object-oriented concepts. This course also introduces exceptions and its control-flow implications. While object-oriented concepts such as polymorphism and inheritance cannot be left out completely, the bulk of this material should be discussed in the follow-up course (SE 2). Most courses that adopt an objects-first approach will do so in an environment that supports a rich collection of application programmer interfaces (APIs). These APIs can be an enormous help to students, because they enable the creation of much more exciting programs at an early level, thereby heightening student motivation—motivating students at this stage is critical. At the same time, the scale of most API packages can be intimidating to many students, since there are so many classes and methods from which to choose. This course deals with text-oriented as well as GUI-oriented input and output.

SE 2: Object-Oriented Design and Methodology
Continues the introduction to object-oriented programming begun in SE 1SE, with an emphasis on algorithms, data structures, software engineering, and the social context of computing.
Related UVic Course: CSc 115/160, SENG 330
Prerequisites: SE 1, LA
Syllabus:

• Object oriented design and object-oriented programming, including tools
• Implementing design specification
• Unit and integartion testing
• Introduction to software quality criteria
• Object-oriented analysis, design, and methods
• Design for reuse
• Design patterns
• Classic techniques for algorithm design and implementation and their place in an object-oriented design including Big Oh notation
• Abstraction and encapsulation through classic data structures: Introduction classic data structures (list, stack, queue, deque, trees, priority queue, heaps, hashtables) and their relation to algorithm design
• Introduction to basic algorithmic analysis including linear, binary and hash search, heapsort, tree traversals, iterators over collections
• Application of algorithm design techniques to a medium-sized project, with an emphasis on formal methods of testing
• Recursion: Recursion as a design technique; implementation of recursion and its relation to iteration; introduction to trees
• Software engineering: Building a medium sized system with algorithmic efficiency in mind
• Separation of concerns, interfaces, serialization, exceptions, genericity, building and using class libraries, APIs
Units covered:
PF1 Fundamental programming constructs 2 core hours (of 9)
PF2 Algorithms and problem-solving 2 core hours (of 6)
PF3 Fundamental data structures 8 core hours (of 14)
PF4 Recursion 3 core hours (of 5)
PF5 Event-driven programming 2 core hours (of 4)
AL1 Basic algorithmic analysis 2 core hours (of 4)
AL2 Algorithmic strategies 2 core hours (of 6)
AL3 Fundamental computing algorithms 3 core hours (of 12)
PL1 Overview of programming languages 2 core hours
PL2 Virtual machines 1 core hour
PL4 Declarations and types 1 core hour (of 3)
PL5 Abstraction mechanisms 2 core hours (of 3)
PL6 Object-oriented programming 4 core hours (of 10)
HC1 Foundations of human-computer interaction 1 core hour (of 6)
SE1 Software design 2 core hours (of 8)
SE2 Using APIs 1 core hour (of 5)
SE5 Software requirements and specifications 1 core hour (of 4)
SE6 Software validation 1 core hour (of 3)
SE3 Software tools and environments 2 core hours (of 3)
Notes: This course builds on the foundation established by SE 1 to complete a full year of introductory programming. Because the first course has included more material on the mechanics of object-oriented programming than is typical in an imperative-first introduction, SE 2 can devote more time to issues of design and software engineering

along with the traditional coverage of data structures and algorithms. This course should balance implementation and use of fundamental data structures.
The goal for the entire year (i.e., SE 1 and SE 2) is solid programming proficiency in one programming language (i.e., Java) and an introduction to software engineering.

SE 3: Software Architecture and Development Methods
Provides an intensive, implementation-oriented introduction to the software-development techniques used to create medium-scale interactive applications, focusing on the use of large object-oriented libraries to create well-designed graphical user interfaces. Topics include software architecture, object-oriented design, event-driven programming, computer graphics, graphical user interfaces, component integration, interfaces, exception handling, serialization, testing, and project management.
Related UVic Course: SENG 330, SENG 365, SENG 480 (components)
Prerequisites: SE 2, WEBENG, DS 2
Syllabus:
• Introduction to software architecture including architecture description languages (ADLs), such as UML; main architectural styles, attribute-based architectural styles (ABASs)
• Layered architectures, levels of indirection, separation of concerns
• Software development techniques: Object-oriented analysis and design; component-level design; software requirements and specifications; prototyping; characteristics of maintainable software; software reuse; team management; project scheduling
• Software design: Fundamental design concepts and principles; design patterns; software architecture; structured design; object-oriented analysis and design; component-level design; design for reuse
• Event-driven programming: Event-handling methods; event propagation; managing concurrency in event handling; exception handling
• Using application programmer interfaces (APIs): API programming; class browsers and related tools; programming by example; debugging in an API environment
• Introduction to component-based software engineering: definitions, models, services, specification, business case, risks
• Introduction to regression and integration testing
• Software engineering principles: separation of concerns, high cohesion, low coupling, small interfaces, design for change and reuse
• Software quality criteria
• Introduction to software process and project management
• Introduction to prototyping techniques, template, stubs, test harness, and tools
• Introduction to configuration management
• Introduction software project management: Team management; project scheduling; software measurement and estimation techniques; risk analysis; software quality assurance; project management tools
• Graphical user interfaces (GUIs): Graphical APIs; choosing interaction styles and interaction techniques; HCI aspects of graphical design (layout, color, fonts, labeling); translation, scaling, rotation; programming environments for creating GUIs
Units covered:

PF5 Event-driven programming 4 core hours
HC1 Foundations of human-computer interaction 6 core hours
HC2 Building a simple graphical user interface 2 core hours
HC3 Human-centered software evaluation 1 hour
HC4 Human-centered software development 1 hour
HC5 Graphical user-interface design 3 hours
HC6 Graphical user-interface programming 3 hours
GV1 Fundamental techniques in graphics 2 core hours
GV2 Graphic systems 1 core hour
SE1 Software design 2 core hours (of 8)
SE2 Using APIs 3 core hours (of 5)
SE3 Software tools and environments 2 core hours (of 3)
SE5 Software requirements and specifications 2 core hours (of 4)
SE6 Software validation 1 core hour (of 3)
SE7 Software evolution 2 core hours (of 3)
SE8 Software project management 2 core hours (of 3)
Elective topics 3 hours

SE 4: Systems Programming and Middleware
An introduction to systems and middleware programming using the C/C++ and Perl programming languages. Interacting with the various components of an operating systems, end-user programmable tools, and the Web. Control, data, and presentation integration mechanisms as well. Web interoperability through Web standards, protocols, and services.
Related UVic Course: SENG 265
Prerequisites: SE 3, CAS (WEBENG and DS 2 through SE 3)
Syllabus:
• File handling; process and thread management; interprocess communication; synchronization; device control; network communications; using Win32 and Unix
• Configuration management systems; configurability and portability of programs
• Windows message system; GUI event handling using MFC
• Sockets and Remote Procedure Calls (RPC)
• Extending the functionality of end-user programmable tools (Internet Explorer, Adobe Acrobat) and development environments using scripting languages
• Building systems using middleware technologies
• Control, data, and presentation integration mechanisms
• Interoperability through Web standards, protocols, and services (XML, DOM, RDF, SOAP, SMIL, SVG)
|*Units covered:* to be written
Notes: The students are assumed to be proficient in object-oriented programming, using APIs, GUI basics, software engineering principles, and software quality criteria.

SE 5: Requirements Engineering and Formal Specification
Combines a range of topics integral to the design, implementation, and testing of a medium-scale software system with the practical experience of implementing such a

project as a member of a programmer team. Introduces formal methods, requirements engineering, specifications, software life cycle models.
Related UVic Courses: SENG 365, SENG 465
Prerequisites: SE 4 (WEBENG, DS 2 though SE 4 and SE 3)
Syllabus:
• Introduction to requirements engineering, prototyping, and participatory design
• Introduction to design specifications
• Introduction to cost estimation
• Introduction to formal methods and verification
• Introduction to software life cycle models
• Introduction to automated testing
• Human-centered development and evaluation; human performance models; accommodating human diversity; principles of good design and good designers; engineering tradeoffs.
• Software processes: Software life-cycle and process models; process assessment models; software process metrics
• Software requirements and specifications: Requirements elicitation; requirements analysis modeling techniques; functional and nonfunctional requirements; prototyping; basic concepts of formal specification techniques
• Software validation: Validation planning; testing fundamentals, including test plan creation and test case generation; black-box and white-box testing techniques; unit, integration, validation, and system testing; object-oriented testing; inspections
• Methods and tools of analysis: Making and evaluating ethical arguments; identifying and evaluating ethical choices; understanding the social context of design; identifying assumptions and values
• Introduction to model checking
• Risks and liabilities of computer-based systems: Historical examples of software risks; implications of software complexity; risk assessment and management
Units covered:
SP3 Methods and tools of analysis 2 core hours
SP4 Professional and ethical responsibilities 3 core hours
SP5 Risks and liabilities of computer-based systems 2 core hours
SE1 Software design 4 core hours (of 8)
SE2 Using APIs 3 core hours (of 5)
SE3 Software tools and environments 1 core hour (of 3)
SE4 Software processes 2 core hours
SE5 Software requirements and specifications 3 core hours (of 4)
SE6 Software validation 2 core hours (of 3)
SE8 Software project management 3 core hours
SE10. Formal methods [core]
SE11. Software reliability [core]
SE12. Specialized systems development [core]
SE13. Testing [core]
Notes: This course introduces the students to formal methods of software development and their industrial applications. This course also discusses project management issues such as skill sets and when and when not to use formal methods.

Professionalism and ethical responsibilities in software development are best presented from a variety of perspectives. This course looks at these issues from a software-centric perspective. The course on social and professional issues investigates these topics more from an engineering perspective.

SE 6: Software Evolution

Introduces problems and solutions of long-term software maintenance/evolution and large-scale, long-lived software systems. Topics include software engineering techniques for programming-in-the-large, programming-in-the-many, legacy software systems, software architecture, software evolution, software maintenance, reverse engineering, program understanding, software visualization, advanced issues in object-oriented programming, design patterns, antipatterns, and client-server computing. This course culminates in a team project.

Related UVic Course: SENG 420, SENG 422, SENG 430

Prerequisites: SE 5

Syllabus:

• Large-system engineering: Separate compilation; design issues; verification and validation; integrating components; documentation

• Advanced issues in object-oriented programming: Modularity; storage management; parallelism; event-centered programming; common design patterns; software reuse

• The Laws of Software Evolution

• Client-server computing: Software support needed for client and server implementation; varieties of server structures; strategies for client-server design; tools for client-server system development; middleware

• Program understanding and software visualization

• Reverse engineering and reverse engineering tools

• Recognizing software architecture and design patterns in existing software systems

• Software transformation, migration, and reengineering

• Recovering software components for reuse

Units covered:

PF3 Fundamental data structures 6 core hours (of 14)
PF5 Event-driven programming 4 core hours
NC4 The web as an example of client-server computing 2 core hours (of 3)
HC1 Foundations of human-computer interaction 3 core hours (of 6)
HC3 Human-centered software evaluation 2 hours
HC4 Human-centered software development 2 hours
HC5 Graphical user-interface design 2 hours
HC6 Graphical user-interface programming 2 hours
PL6 Object-oriented programming 4 core hours (of 10)
SE1 Software design 2 core hours (of 8)
SP5 Risks and liabilities of computer-based systems 1 core hour (of 2)
SE2 Using APIs 3 core hours (of 5)
SE4 Software processes 1 core hour (of 2)
SE5 Software requirements and specifications 2 core hours (of 4)
SE6 Software validation 2 core hours (of 3)

SE7 Software evolution 1 core hour (of 3)
SE8 Software project management 1 core hour (of 3)
Notes:
An important stage in the education of a good software developer consists of making the transition from programming-in-the-small to programming-in-the-large. The purpose of this course is to bridge that gap by enabling students to develop large programs in well defined stages. In the process, this course explores the requirements at each stage of the development along with various issues of quality control. In the practical component of the course, students learn to appreciate the range of facilities that a typical object-oriented language offers and to apply sound approaches to software design in the large-system environment.
Moving from programming-in-the-small to programming-in-the-large, however, is not straightforward. Students need to be provided with a range of reasonable illustrations and examples for them to attempt. In the syllabus presented here, those illustrations are drawn from net-centric computing and user-interface design. Other possibilities, however, exist as well. For example, courses designed to introduce programming-in-the-large might be based on e-commerce, groupware, or other kinds of sophisticated application servers. In every case, it is important to emphasize the importance of complexity management by showing how large tasks can be broken down into smaller tasks that can often be addressed through the selection of appropriate algorithms. In this way, students see the relevance of earlier course work, including the study of algorithms and complexity. Ultimately, however, the students must demonstrate their understanding of the principles by designing an interface of some sophistication. An important aspect of the practical component of HCI lies in exposing students to state-of-the-art software that supports such development, including special-purpose tools and class libraries to support interface development. The laboratory aspect of this course can also benefit from the use of design languages and associated tools.

SE 7: Embedded Systems
Embedded real-time systems are pervasive in today's world, for example, they are found in telecommunications systems, consumer electronic products, automotive systems, and aerospace systems. Design methodologies for embedded systems include hardware-software co-design techniques, should be considered to be essential for future designers of embedded systems. This course introduces the characteristics and design of embedded systems.
Related UVic Course: SENG 422
Prerequisites: SE 6

Syllabus:
• Design of embedded systems: technology trends
• Applicaton areas
• Formal models and specification languages for capturing system behaviour
• Design analysis and optimisation
• Hardware-software co-design
• Mechatronics

• Basic control & feedback theory
• Signals and systems
• Models of computation: finite state machines; dataflow, synchronous languages
• Unified modelling frameworks
• Estimation, simulation and synthesis
• Scheduling: hardware and software tasks
• Architectural models
• Hardware selection: micocontrollers, DSPs, reconfigurable logic, system-on-a-chip
• Real-time operating systems
• Design methodologies and tools
• Tools for validation, verification, simulation, modeling, and information
• Quality and performance metrics for embedded systems
• Intellectual property; design and reuse of IP
Units covered: to be edited further Kin, Jens
Notes: to be written

SE 8: Software Process Improvement and Quality Assurance (Software Quality Engineering)
This course emphasizes software quality engineering as an integral facet of development, from requirements through delivery, maintenance, and process improvement. The students will learn how to choose appropriate quality goals at the outset of a project and select, plan, and execute quality assurance activities throughout development and evolution to predictably meet quality and schedule goals. They will learn how to carry out inspections, manual and automated static analysis techniques, design for testability, and test planning and execution. They will also learn how quality assurance can be incorporated into process improvement feedback loops that amplify the ability of an organization to cost-effectively prevent and detect faults. Students will participate in group projects, which involve industry-relevant software process definition and assessment.
Related UVic Course: SENG 472, SENG 470, SENG 365, SENG 465
Prerequisites: SE 6
Syllabus:
• Software processes and the process cycle
• Software process design, modeling, implementation
• Software process management, assessment and improvement
• Non-process factors that affect software quality
• Software project management
• Software quality assessment
• Students will participate in group projects, which involve industry-relevant software process definition and assessment
• Software maturity and the state of the industry
• Software inspections; testability
• Introduction to software quality and quality assurance
• TQM (Total Quality Management)
• GQM (Goal/Question/Metric) paradigm
• Maximizing customer satisfaction

• Introduction to software measurement
• Minimizing engineering effort and schedule
• Minimizing defects; software metrics
• Object-oriented software metrics
• Use and etiquette of software metrics
• Process improvement and Return on Investment(ROI) analysis
• Software assessment and standards
• Software auditing and the roles of the auditor
• Software process management
• Software technology evolution
Units covered: to be written
Notes: to be written

CAP: Capstone Project
Offers students the opportunity to integrate their knowledge of the undergraduate computer science curriculum by implementing a significant software system as part of a programming team.
Related UVic Course: SENG 499, ELEC 499, CENG 499, CSC 490
Prerequisites: SE 6 (forces the course to be in fourth year for accreditation)
Syllabus:
Units covered:
• Event-driven programming 4 core hours
• Graphical user-interface design 3 hours
• Object-oriented programming 3 core hours
• Graphical user-interface design 2 hours
• Software design 4 hours
• Using APIs 3 core hours
• Software tools and environments 3 core hours
• Software processes 3 core hours
• Software requirements and specifications 2 core hours
• Software validation 3 core hours
• Software evolution 2 core hours
• Software management 4 hours
• Team management 4 hours
• Communications skills 4 hours
• Elective topics 2 hours
Notes: This course is a project course. There are no lectures. The overall idea is that students should have a chance to apply all the skills they have learned in the curriculum toward the completion of a team project. Thus, this course has the effect of reinforcing concepts that have been learned earlier in a more theoretical way. This course can be taken as a one or two-term course.

MECHSYS: Mechanical Systems for Engineers
Introduce engineering students to systems built by mechanical engineers and modes of

thinking used by these engineers through the teaching of selected subjects with lectures by faculty and invited speakers from industry, films, and laboratory demonstrations. As an introductory course students will be exposed to fundamentals of engineering design, engineering ethics, problem solving methods demanding the application of fundamental engineering principles and checking of solutions. Introduce students to the terminology, mathematics, and methods used in mechanical engineering. The lectures also discuss career opportunities in mechanical engineering and related fields, emerging technologies, and the cross-disciplinary nature of engineering. The course features a significant project. Motivating engineering students is a key aspect of this course.
Related UVic Course: MECH 141
Prerequisites: none
Syllabus:
• What is Mechanical Engineering?
• Ethical issues an engineer will face
• Importance of communication skills and team work in engineering
• The design process
• Mathematics and numerical methods in mechanical engineering
• Statics
• Dynamics
• Control systems
• Mechanical systems
• Mechatronics
• Fuel-cell systems
• Micro-electromechanical systems (MEMS)
• Nanotechnology
Units covered: to be written
Notes: This course is intended for all engineering majors particularly software engineering majors. The math is not hard. At the end of this course, any engineering student should be able to communicate with mechanical engineers effectively. Motivating engineering students is a key aspect of this course.

ELECSYS: Electrical Systems for Engineers
Introduce engineering students to systems built by electrical and computer engineers and modes of thinking used by these engineers through the teaching of selected subjects with lectures by faculty and invited speakers from industry, films, and laboratory demonstrations. As an introductory course students will be exposed to fundamentals of engineering design, engineering ethics, problem solving methods demanding the application of fundamental engineering principles and checking of solutions. Introduce students to the terminology, mathematics, and methods used in electrical engineering. The lectures also discuss career opportunities in electrical engineering and related fields, emerging technologies, and the cross-disciplinary nature of engineering. The course features a significant project. Motivating engineering students is a key aspect of this course.

Aaron: Introduction to the language and principles of electrical engineering; Circuit analysis and design techniques; Definitions of circuit elements, fundamental laws. Mathematical models of physical systems. Analysis and design of first and second-order systems. Introduction to computer applications in control systems, signal processing and communications. A practical project is undertaken by teams of students. The project involves mechanical construction, sensing of mechanical quantities by electrical means, as well as interfacing to and programming of a simple microcontroller. Students will be required to demonstrate their designs and write a report documenting their efforts.
Related UVic Course: ELEC 199, ELEC 250
Prerequisites: none
Syllabus:
• What is Electrical and Computer Engineering?
• Ethical issues an engineer will face
• Importance of communication skills and team work in engineering
• The design process
• Mathematics and numerical methods in electrical engineering (complex numbers, sets, functions)
• Building complex systems by connecting simpler subsystems (cascade, parallel, feedback)
• Signal processing
• Analog and digital communication systems
• Digital feedback control systems
• Energy and power systems
• Electromechanical systems
Units covered: to be written
Notes: This course is intended for all engineering majors particularly software engineering majors. The math is not hard, even though the course introduces some Fourier Series. At the end of this course, any engineering student should be able to communicate with electrical engineers effectively.

CAS: Computer Architecture and Assembler Programming
Introduces students to the organization and architecture of computer systems, beginning with the standard von Neumann model and then moving forward to more recent archictural concepts.
Related UVic Course: CSC 230, CENG 355
Prerequisites: SE 2, DS 2
Syllabus:
• Data representation: Bits, bytes, and words; numeric data representation and number bases; fixed- and floating-point systems; signed and twos-complement representations; representation of nonnumeric data (character codes, graphical data); representation of records and arrays
• 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, 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
• Functional organization: Implementation of simple datapaths; control unit: hardwired realization vs. microprogrammed realization; instruction pipelining; pipeline hazards; introduction to instruction-level parallelism (ILP)
• Multiprocessor and alternative architectures: Introduction to SIMD, MIMD, VLIW, EPIC; systolic architecture; interconnection networks; shared memory systems; cache coherence; memory models and memory consistency
• Performance enhancements: RISC architecture; branch prediction; prefetching; scalability; code generation
• Contemporary architectures: Hand-held devices; embedded systems; trends in processor architecture
Project: writing an interpreter for a small assembly language
Units covered:
AR1 Digital bsystems 3 core hours (of 6)
AR2 Machine level representation of data 3 core hours
AR3 Assembly level machine organization 9 core hours
AR4 Memory system organization and architecture 5 core hours
AR5 Interfacing and communication 3 core hours
AR6 Functional organization 7 core hours
AR7 Multiprocessing and alternative architectures 3 core hours
AR8 Performance enhancements 3 hours
Contemporary architectures 2 hours
Elective topics 2 hours
Notes:
This course should be taught from a systems perspective rather than pure digital logic perspective. It is more important that the students learn about tradeoffs than a particular hardware platform or assembler language.
The course should be taught with embedded systems and ubiquitous computing in mind. To emphasize the systems perspective, writing an interpreter for a small assembly language might be more instructive than writing assembler programs.
Differences in the internal structure and organization of a computer 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 addresses the various options involved in designing a computer system, the range of design considerations, and the trade-offs involved in the design process. This course might also address interoperability issues (e.g., Web interface in a cell phone).
A key issue in relation to this course is motivation. 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.

ALG 1: Algorithms and Data Structures
Introduces formal techniques to support the design and analysis of algorithms, focusing on both the underlying mathematical theory and practical considerations of efficiency. Topics include asymptotic complexity bounds, techniques of analysis, algorithmic strategies, and an introduction to automata theory and its application to language translation.
Related UVic Course: CSc 225
Prerequisites: SE 2, DS 2, LA
Syllabus:
• Review of proof techniques
• Basic algorithmic analysis: Asymptotic analysis of upper and average complexity bounds; best, average, and worst case behaviors; big-O, little-o, Ω, and Ø notation; standard complexity classes; empirical measurements of performance; time and space tradeoffs in algorithms; using recurrence relations to analyze recursive algorithms
• Fundamental algorithmic strategies: Brute-force; greedy; divide-and-conquer; backtracking; branch-and-bound; heuristics; pattern matching and string/text algorithms; numerical approximation
• Fundamental data structures: Implementation strategies for graphs and trees; performance issues for data structures
• Graph and tree algorithms: Depth- and breadth-first traversals; shortest-path algorithms (Dijkstra's and Floyd's algorithms); transitive closure (Floyd's algorithm); minimum spanning tree (Prim's and Kruskal's algorithms); topological sort
Units covered:
DS3 Proof techniques 3 core hours (of 12)
DS5 Graphs and trees 4 core hours
PF2 Algorithms and problem-solving 3 core hours (of 6)
PF3 Fundamental data structures 3 core hours (of 14)
PL3 Introduction to language translation 2 core hours
AL1 Basic algorithmic analysis 2 core hours (of 4)
AL2 Algorithmic strategies 6 core hours
AL3 Fundamental computing algorithms 6 core hours (of 12)
AL5 Basic computability 6 core hours
AL6 The complexity classes P and NP 2 hours
Elective topics 1 hour
Notes:
The topic of algorithmic analysis is central to much of computing. The thrust of this course is to explore and examine a range of algorithms that can be used to solve practical problems. Each algorithm possesses strengths and weaknesses. Moreover, the performance or any particular algorithm typically varies according to the size and nature of the input data. Students need a thorough understanding of the tools of analysis in order to select the right algorithm for the job. Students are most receptive to the material

presented in this course if they understand the connections between theory and practice. To this end, instructors should try to find ways to reinforce the theoretical topics through practical activity. It is also important for instructors to provide compelling demonstrations of the enormous differences in running time that can occur when algorithms have different complexity characteristics. The importance of complexity measures must be made real. Algorithmic animation can be a powerful tool toward getting students to understand both the algorithms themselves and the associated complexity measures. Tools for creating graphical animations of classical algorithms are widely available on the Web. These tools provide visible evidence of the complexity measures and thus reinforce the theoretical results. It is also possible to take a more formal approach to this topic that focuses on formal specification of algorithms and proofs of correctness, possibly supported by appropriate specification and verification tools. A more informal approach, however, is likely to appeal to a wider spectrum of students.

Students who complete this course should be able to perform the following tasks:

• Explain the mathematical concepts used in describing the complexity of an algorithm.
• Select and apply algorithms appropriate to a particular situation.
• Employ one from a range of strategies leading to the design of algorithms to serve particular purposes.
• Explain the trade-offs that exist between a range of algorithms that possess the same functionality.

DD: Digital Design

This course develops a structured design methodology for the design of complex digital systems. This is achieved by using a systems level approach to the development of a digital design. An introduction to the use of suitable CAD tools is given in the laboratories whilst covering the more theoretical aspects associated with logic design in the lectures.

Related UVic Course: CSC 355, CENG 290, CENG 355

Prerequisites: CAS (SE 2 and DS 2 through CAS)

Syllabus:

Hardware description language and VHDL

Sequential and concurrent processes, Hardware Description Languages, VHDL entity and architecture

VHDL: architectural views, signals variables and constants, types, operators

VHDL: concurrent assignments, wait statements, processes and procedures, delta time

VHDL: the simulation cycle, assert statements, signal attributes, libraries and packages

Test generation, controllability and observability, scan path testing,VHDL test benches

Sequential system design

Revision of FSM's and bubble graphs. Sequential design using the ASM chart

Example ASM charts, Mealy and Moore machines, link path extraction, looping

Practical example, state assignment, minimum state locus, one-hot state encoding

State tables and forming logic

Map entered variables, linked state machines

Logic implementation and synthesis

Logic architectures and logic synthesis, nand-nor, multiplexers and the ULM

Fusible link devices, ROM based implementation
FPGA and RAM routable gate arrays
Model checking
Microprocessor systems and applications
Units covered:
Notes: This course is an introduction to digital *systems* rather than digital *logic*. Thus, this course is at a higher level than traditional digital design courses (e.g., CENG 290). Students who complete this course should be able to perform the following tasks:
• Understand the importance of hierarchy in the design of complex digital systems
• Know the different levels of abstraction used to represent logic systems and be familiar with the data representation and method of simulation at each level
• Be able to use VHDL at the various levels of abstraction to design a digital system
• Understand the need for controllability and observability in designing systems for test
• Be able to design a FSM from an algorithmic description of a problem
• Understand the requirements of PAL, PLA and FPGA logic architectures for logic synthesis
• Be able to understand and design logic systems to implement basic computer systems
• To be edited further by Micaela

AFL: Automata Theory and Formal Languages
A survey of formal models and results that form the theoretical foundations of computer science; typical topics include regular and context-free languages, finite automata, Chomsky hierarchy, Turing machines, undecidable problems, and computational complexity.
Related UVic Course: CSc 320
Prerequisites: ALG 1
Syllabus:
• Languages, grammars, and automata
• DFAs and their implementation
• NFAs=NDFAs
• Regular expressions
• Regular grammars
• Closure, homomorphism
• Pigeonhole principle, pumping lemma
• Context free grammars (CFGs)
• Parsing and ambiguity
• Pushdown automata
• Turing machines
• Recursively enumerable languages
• The Chomsky hierarchy
• Undecidable problems
• Complexity classes P and NP
• NP-complete problems
Units covered: to be written
Notes: to be written

LA: Linear Algebra (Matrix Algebra for Engineers)
Complex numbers; matrices and basic matrix operations; vectors; linear equations; determinants; eigenvalues and eigenvectors; linear dependence and independence; orthogonality.
Related UVic Course: MATH 133
Prerequisites:
Syllabus:
Units covered:
Notes:

WE: Web Engineering
Introduces students to the world of computing and communications through the World-Wide Web. SE 1 programming background is required, students will learn some programming through scripting languages. Topics include security, privacy, history, multimedia technologies, HCI, network management, electronic commerce.
Related UVic Course: SENG 265
Prerequisites: SE 1
Syllabus:
• Introduction to the Internet: Background and history of networking and the Internet; overview of network architectures
• Introduction to the World-Wide Web: Web technologies; the HTML language; Web authering tools
• Analysis and design of Web applications including small hand-held devices
• Introduction to graphics on the Web using Scaleable Vector Graphics (SVG)
• Scripting and the role of scripting languages; basic system commands; creating scripts, parameter passing; executing a script.
• Multimedia data technologies: Sound and audio, image and graphics, animation and video; input and output devices; tools to support multimedia development
• Interactivity on the web: Scripting languages; the role of applets
• Human-computer interaction: HCI aspects of web-page design; graphical user interface design
• Introduction to XML and other selected W3C standards
• Network management: Overview of the issues of network management; use of passwords and access control mechanisms; domain names and name services; issues for Internet service providers; security issues and firewalls;
• Network security: Fundamentals of cryptography; secret-key algorithms; public-key algorithms; authentication protocols; digital signatures; examples
• Introduction to electronic commerce
• Intellectual property: Foundations of intellectual property; copyrights, patents, and trade secrets; issues regarding the use of intellectual property on the web
• Privacy and civil liberties: Ethical and legal basis for privacy protection; freedom of expression in cyberspace; international and intercultural implications
Units covered:

NC1 Introduction to net-centric computing 2 core hours
NC2 Communication and networking 2 core hours
NC3 Network security 3 core hours
NC4 The web as an example of client-server computing 3 core hours
NC5 Building Web applications 3 hours
NC6 Network management 2 hours
NC7 Compression and decompression 3 hours
NC8 Multimedia data technologies 3 hours
HC5 Graphical user-interface design 2 hours
HC7 HCI aspects of multimedia systems 2 hours
SE3 Software tools and environments 2 core hours (of 3)
SP6 Intellectual property 2 core hours (of 3)
SP7 Privacy and civil liberties 2 core hours
Elective topics 9 hours

PS: Introduction to Probability and Statistics
Descriptive statistics; elementary probability theory; random variables, discrete and continuous probability distributions, expectation, joint, marginal and conditional distributions; linear functions of random variables; random sampling and sampling distributions; point and interval estimation; classical hypothesis testing and significance testing. The mathematical foundations of statistical inference will be introduced and illustrated with examples from a variety of disciplines.
Related UVic Course: STAT 260 or STAT 254
Prerequisites: none
Syllabus:
Units covered:
Notes:

HCI: Human-Computer Interaction
Presents a comprehensive introduction to the principles and techniques of human-computer interaction.
Prerequisites: SE 3, WEBENG
Syllabus:
• Foundations of human-computer interaction: Motivation; contexts for HCI; human-centered development and evaluation; human performance models; human performance models; accommodating human diversity; principles of good design and good designers; engineering tradeoffs; introduction to usability testing
• Human-centered software evaluation: Setting goals for evaluation; evaluation without users; evaluation with users
• Human-centered software development: Approaches, characteristics, and overview of process; functionality and usability; specifying interaction and presentation; prototyping techniques and tools
• Graphical user-interface design: Choosing interaction styles and interaction techniques; HCI aspects of common widgets; HCI aspects of screen design; handling human failure; beyond simple screen design; multi-modal interaction; 3D interaction and virtual reality

• Graphical user-interface programming: Dialogue independence and levels of analysis; widget classes; event management and user interaction; geometry management; GUI builders and UI programming environments; cross-platform design
• HCI aspects of multimedia systems: Categorization and architectures of information; information retrieval and human performance; HCI design of multimedia information systems; speech recognition and natural language processing; information appliances and mobile computing
• HCI aspects of collaboration and communication: Groupware to support specialized tasks; asynchronous group communication; synchronous group communication; online communities; software characters and intelligent agents
Units covered:
PF5 Event-driven programming 2 core hours (of 4)
HC1 Foundations of human-computer interaction 6 core hours
HC2 Building a simple graphical user interface 2 core hours
HC3 Human-centered software evaluation 5 hours
HC4 Human-centered software development 5 hours
HC5 Graphical user-interface design 6 hours
HC6 Graphical user-interface programming 3 hours
HC7 HCI aspects of multimedia systems 5 hours
HC8 HCI aspects of collaboration and communication 3 hours
PL6 Object-oriented programming 2 core hours (of 10)
Elective topics 1 hour

OSDC: Operating Systems and Distributed Computing
Introduces the fundamentals of operating systems together with the basics of distributed computing.
Related UVic Course: CSC 360
Prerequisites: SE 4 (WEBENG and CAS through SE 4)
Syllabus:
• Introduction to event-driven programming
• Overview of operating systems: Role and purpose of the operating system; history of operating system development; functionality of a typical operating system
• Operating system principles: Structuring methods; abstractions, processes, and resources; concepts of application program interfaces; device organization; interrupts; concepts of user/system state and protection
• Introduction to concurrency: Synchronization principles; the "mutual exclusion" problem and some solutions; deadlock avoidance
• Introduction to concurrency: States and state diagrams; structures; dispatching and context switching; the role of interrupts; concurrent execution; the "mutual exclusion" problem and some solutions; deadlock; models and mechanisms; producer-consumer problems and synchronization
• Scheduling and dispatch: Preemptive and nonpreemptive scheduling; schedulers and policies; processes and threads; deadlines and real-time issues
• Memory management: Review of physical memory and memory management hardware; overlays, swapping, and partitions; paging and segmentation; placement and

replacement policies; working sets and thrashing; caching
• Introduction to distributed algorithms: Consensus and election; fault tolerance
• Introduction to net-centric computing: Background and history of networking and the Internet; network architectures; the range of specializations within net-centric computing
• Introduction to networking and communications: Network architectures; issues associated with distributed computing; simple network protocols; APIs for network operations
• Characteristics of Web servers; nature of the client-server relationship; Web protocols
• *Units covered:*
PF5 Event-driven programming 2 core hours
AL4 Distributed algorithms 3 core hours
OS1 Overview of operating systems 2 core hours
OS2 Operating system principles 2 core hours
OS3 Concurrency 6 core hours
OS4 Scheduling and dispatch 3 core hours
OS5 Memory management 5 core hours
NC1 Introduction to net-centric computing 2 core hours
NC2 Communication and networking 7 core hours
NC4 The web as an example of client-server computing 3 core hours
Notes:
In a more traditional implementation of the core, one might require one course in operating systems and another in networks. There is, however, a good deal of interplay between these topics. Today the desktop and the Web readily integrate. It therefore makes sense to design a course that looks at these pieces of system software together, particularly since the Web is extremely appealing to students. Combining the operating system topics with the discussion of networking helps motivate students and stimulates their thinking about both the effect of the web on operating systems and the more general principles involved. The issue of motivation is paramount in the design of the course. The area of operating systems is often regarded as difficult for both students and faculty, but nonetheless contains many ideas of relevance to all computer scientists. Faculty must ask themselves how they can make operating systems relevant to undergraduates. This consideration must drive the choice of approach to learning and teaching. To this end, students must see these issues as related to the systems that they use. As an example, students might be
asked to consider the impact on the operating system of such developments as networking, multimedia, security, and hand-held devices. Similarly, one could also ask about the impact of other developments, such as the following:
• Playing music on a CD at the same time as using the computer
• Downloading TV pictures onto a window
• Docking systems or devices such as digital cameras and hand-held computers
• Client-server architectures
In pursuing any course on operating systems, students need to be made aware of the wider relevance of many of the ideas. It is therefore useful to highlight the following connections:
• The cache idea, while relevant at the hardware level, shows up again in the context of

the web and downloading material from web sites.
• The concepts that arise in the discussion of virtual memory come up again in the development of virtual environments.
• The material on concurrency is relevant in the wider context of concurrent and parallel programming.
• The material on resource allocation and scheduling features as a major component of operations research.
• Much of the course material is relevant to the design and construction of real-time and dependable systems.
Students are likely to take a greater interest in operating systems if they see themselves as working in the context of a real system rather than some highly simplified and more abstract simulation. In this regard, the open-source movement has made an important contribution to pedagogy in the operating systems area, because the source code for several well-known operating systems is now available free of charge. These publicdomain
resources make it easier to illustrate aspects of operating systems and can often provide useful examples of how different systems implement particular features. It is worth observing that many of the students are likely to be fired up with the idea of installing Linux (for example) on their own machines.
Students who complete this course should be able to perform the following tasks:
• Summarize the principles underlying the design and construction of a typical operating system, giving particular recognition to the wider applicability of the ideas and the influences from such developments as high-level languages, networking, multimedia, and security concerns.
• Use the facilities of the operating system to achieve a range of simple tasks, including enhancing the functionality by integrating new software components.
• Identify the security issues associated with distributed web applications and be able to suggest mechanisms leading to a resolution of these problems.

DB: Databases
Introduces the concepts and techniques of database systems.
Related UVic Course: CSC 370
Prerequisites: SE 4 (WEBENG through SE 4)
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; multi-valued 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
Units covered:
HC1 Foundations of human-computer interaction 2 core hours (of 6)
IM1 Information models and systems 3 core hours
IM2 Database systems 3 core hours
IM3 Data modeling 4 core hours
IM4 Relational databases 5 hours
IM5 Database query languages 4 hours
IM6 Relational database design 4 hours
IM7 Transaction processing 3 hours
IM8 Distributed databases 3 hours
IM9 Physical database design 3 hours
SP6 Intellectual property 3 core hours
SP7 Privacy and civil liberties 2 core hours
Elective topics 1 hour

NET: Networks
Introduces the structure, implementation, and theoretical underpinnings of computer networking and the applications that have been enabled by that technology.
Related UVic Course: SENG 450, CSC450, CENG 460
Prerequisites: OSNET (SE 4 and WEBENG through OSNET)
Syllabus:
• Communication and networking: Network standards and standardization bodies; the ISO 7-layer reference model in general and its instantiation in TCP/IP; circuit switching and packet switching; streams and datagrams; physical layer networking concepts; data link layer concepts; Internetworking and routing; transport layer services
• The web as an example of client-server computing: Web technologies; characteristics of web servers; role of client computers; nature of the client-server relationship; web protocols; support tools for web-site creation and web management; developing Internet information servers; publishing information and applications
• Building web applications: Protocols at the application layer; principles of web engineering; database-driven web sites; remote procedure calls; lightweight distributed objects; the role of middleware; support tools; security issues in distributed object systems; enterprise-wide web-based applications

• Network management: Review of the issues of network management; issues for Internet service providers; security issues and firewalls; quality of service issues
• Compression and decompression: Review of basic data compression; audio compression and decompression; image compression and decompression; video compression and decompression; performance issues
• Multimedia data technologies: Review of multimedia technologies; multimedia standards; capacity planning and performance issues; input and output devices; MIDI keyboards, synthesizers; storage standards; multimedia servers and file systems; tools to support multimedia development
• Wireless and mobile computing: Overview of the history, evolution, and compatibility of wireless standards; the special problems of wireless and mobile computing; wireless local area networks and satellite-based networks; wireless local loops ; mobile Internet protocol; mobile-aware adaption; extending the client-server model to accommodate mobility; mobile data access; the software packages to support mobile and wireless computing; the role of middleware and support tools; performance issues; emerging technologies
Units covered:
PF5 Event-driven programming 2 core hours (of 4)
NC1 Introduction to net-centric computing 2 core hours
NC2 Communication and networking 7 core hours
NC3 Network security 3 core hours
NC5 Building Web applications 8 hours
NC6 Network management 2 hours
NC7 Compression and decompression 3 hours
NC8 Multimedia data technologies 3 hours
NC9 Wireless and mobile computing 4 hours
Elective topics 1 hour

RT: Real-time Systems
Fundamental issues in design of real-time operating systems and application software. Typical topics include: hard real-time scheduling, interrupt driven systems, process communication and synchronization, language requirements for real-time systems, decomposition of real-time requirements into process models, and case studies. A project involving design, implementation and testing of a real-time executive and real-time application software will also be included
Related UVic Course: CSC 460, CENG 455
Prerequisites: SE 7
Syllabus: to be written by Mantis
Units covered:
Notes:

CALC1: Calculus 1
Review of analytic geometry; functions and graphs; limits; derivatives; techniques and applications of differentiation; antiderivatives; the definite integral and area; logarithmic

and exponential functions; trigonometric functions; Newton's, Simpson's and trapezoidal methods.
Related UVic Course: MATH 100
Prerequisites: none
Syllabus: to be written
Units covered: to be written
Notes: none

CALC2: Calculus 2
Volumes; arc length and surface area; techniques of integration with applications; polar coordinates and area; l'Hospital's rule; Taylor's formula; improper integrals; series and tests for convergence; power series and Taylor series; complex numbers.
Related UVic Course: MATH 101
Prerequisites: CALC 1
Syllabus: to be written
Units covered: to be written
Notes: none

SYSDYN: System Dynamics
Modelling of dynamic systems using differential and difference equations with applications. Simulation of engineering systems and applications; transfer functions and block diagrams; convolution; Complex numbers; Fourier series and the Fourier transform. Frequency response. Functions of a complex variable. The Laplace transform in the representation of signals. Interrelation between the Fourier and Laplace transforms. Partial differentiation.
Related UVic Course: MATH 202
Prerequisites: CALC 2
Syllabus: to be written
Units covered: to be written
Notes: none

SAS: Signal and Systems
Continuous time signals and waveform calculations. The Fourier series in the analysis of periodic signals. The impulse and other elementary functions. Resolution of signals into impulse and unit step functions. The Fourier transform in spectral analysis. Functions of a complex variable. Analytic functions. Partial fractions. The Laplace transform in the representation of signals. Interrelation between the Fourier and Laplace transforms. Design project using Matlab.
Related UVic Course: ELEC 260
Prerequisites: CALC 3, DS 2
Syllabus:
• Signals as functions; signal types: sound, images, position in space, angles of a robot arm, binary sequences, word sequences, and event sequences

• Systems as functions that map functions (signals) into functions (signals); DTMF signaling, modems, digital voice, and audio storage and retrieval
• State: relating declarative and imperative descriptions of signals and systems; state machines; Matlab components
• Determinism: nondeterminism and equivalence in state machines; simulation relations and bisimulation are defined for both deterministic and nondeterministic machines
• Composition of state machines; semantics to block diagrams and feedback
• Linearity: linear systems as state machines
• Responses: time-invariant state updates; impulses and impulse response
• Frequency domain concepts and the Fourier series; periodic signals are defined; Fourier series coefficients are calculated by inspection for certain signals; frequency domain decomposition
• Frequency response: linear, time-invariant (LTI) systems
• Filtering: lowpass, bandpass, and highpass; applications to audio and images; composition of LTI systems
• Convolution: signals as sums of weighted impulses; linearity and time invariance to derive convolution; FIR systems
• Transforms: relation of frequency response and convolution; building the bridge between time and frequency domain views of systems; DTFT and the continuous-time Fourier transform
• Sampling: sampling and aliasing as a major application; Nyquist-Shannon sampling theorem
• Filter design is considered with the objective of illustrating how frequency response applies to real problems
• Extensive use of Matlab
Units covered: to be written
Notes: The course is designed to be as relevant to computer scientists as to electrical engineers. Thus, it does not have a circuits prerequisite, and does not use circuits as an illustration of systems. Fundamental limits have also changed. Although we still face thermal noise and the speed of light, we are likely to encounter other limits before we get to these, such as
complexity, computability, and chaos. The mathematical basis for the discipline has also shifted. Although we still need calculus and differential equations, we more frequently need discrete math, set theory, and mathematical logic.

CTRL: Control Systems
To introduce the basics of design and analysis of control systems. Principles of control: block diagrams, transfer functions, open and closed loop systems, linear-time invariant systems, Bode Plot and Nichols' Chart. Performance specification and estimation: stability criterion, Routh-Hurwitz and Nyquist stability criteria; root locus methods; steady state errors, transient performance. Simple design methods. Discrete systems: Z-transform, stability criterion, discrete continuous equivalence, sampling interval considerations. Automation: the design process, design specification, technological alternatives, economics, sensor systems, actuation systems, interfacing, signal conditioning, DC servos, Proportional-Integral-Derivative (PID) controllers; lead and lag

compensators; robust design. Microprocessor based control systems. Product design. Aspects of robot system design.
Related UVic Course: MECH 435, ELEC 360
Prerequisites: CALC 3, DS 2
Syllabus:
• Properties of control and linear-time invariant systems
• Feedback system characteristics: Sensitivity reduction, transient response control, noise attenuation, steady-state error improvement.
• Two-dominant-pole model: Damping ratio, natural frequency, relationships of pole locations to transient spec's; model order reduction by partial fraction expansion and justification of the two-dominant-pole assumption.
• Steady-state error, final-value theorem, performance indices.
• Stability: Relation to pole location, Routh-Hurwitz stability criterion. Root locus: The concept of root-locus, relation to open-loop pole-zero plot; phase angle and magnitude conditions; asymptotic behavior for large and small gains; behavior on real axis; Root locus behavior at break-away points; sketching examples; use for parameter variation analysis; root sensitivity.
• Frequency response model: Bode plot, polar plot, and log magnitude vs. phase angle plot; Relation to open-loop pole-zero plot.
• Procedures for sketching Bode plot given the pole-zero plot; determination of transfer function from Bode plot.
• Minimum and non-minimum phase systems; two-dominant-pole system; resonant peak and resonant frequency, relation to damping ratio and natural frequency; transient properties from a closed-loop frequency response; determination of steady-state error from open-loop frequency response.
• Nyquist stability criterion
• Compensation using Nichols' chart.
• Proportional-Integral-Derivative (PID) control
• State-space methods: State-space models of dynamic systems, transfer function, and stability. Controllability and observability. Pole placement and state estimation.
• Discrete-time systems and digital control: discrete-time linear time-invariant systems; sampled-data systems and conversions between continuous-time and discrete-time; digital control of continuous time systems; design and realization of digital control systems; stability; transient response of sampled data systems, the relationship between the s-plane and the z-plane; simulation of Discrete-Time Systems;
• Introduction to intelligent systems for modeling and control.
• Introductions to Matlab control toolbox and Simulink.
Units covered: to be written
Notes:

SEC: Security Engineering
This course presents the fundamentals of contemporary computer security and cryptology. Topics included an overview of computer security, protection, disaster planning, and recovery. Risk analysis and security plans. Basics of cryptography. Public key cryptography and protocols. Security models, kernel design and systems testing.

Database, network and Web security. The course discusses applications which need various combinations of confidentiality, availability, integrity and covertness properties; mechanisms to incorporate these properties in systems. The course also deals with policy and legal issues.
Related UVic Course: none
Prerequisites: OSNET (SE 4 through OSNET)
Syllabus:
• Security requirements and applications. Bookkeeping systems; transaction processing systems; multilevel secure systems; electronic warfare.
• Multilevel security policy models. The Bell-LaPadula model; System Z; the lattice model; the Biba model; composability, noninterference and nondeducibility; polyinstantiation; the effect of viruses; covert channels.
• Further security policy topics. The Clark-Wilson model; the Chinese Wall; privacy and compartmented-mode models; inference control.
• Access control basics: access matrices, access control lists, capabilities, roles and groups. Password cracking, malicious code and intrusion detection.
• Unix and Internet security including sendmail problems, the Internet worm, password sniffing attacks and firewalls.
• Malicious code, intrusion detection, application level controls, denial of service.
• Copyright marking; steganography; unobtrusive communications; tamper resistance.
• Stream ciphers; monoalphabetics, Vigenere ciphers. Linear feedback shift registers; the nonlinear filter generator; the multiplexer generator; divide and conquer attacks; fast correlation attacks.
• Basic block ciphers. Feistel ciphers including DES and TEA. Modes of operation: electronic code book, cipher feedback, output feedback, cipher block chaining; MACs and hash functions.
• Elementary cryptographic protocols, including Needham-Schroder, Otway-Rees, Kerberos and Kryptoknight. Key management in banking systems. Protocol failures and the BAN logic.
• Public key cryptography: Diffie-Hellman, ElGamal, DSA, RSA, digital cash, threshold signatures.
• Public key protocols and failures: Denning-Sacco, Needham Schroder, Tatebayashi-Matsuzaki-Newmann. Chosen protocol attacks. The BAN logic for public key protocols.
• Security engineering: what actually goes wrong with real systems. Threat trees and risk models. Evaluation and accreditation. Policy and legal issues: civil and criminal evidence rules; data protection acts computer misuse acts. Organisational issues; due diligence and the role of insurance.
Units covered: to be written
Notes: none

DS 1: Discrete Structures 1
Introduces the foundations of discrete mathematics as they apply to computer science, focusing on providing a solid theoretical foundation for further work. Topics include functions, relations, sets, simple proof techniques, Boolean algebra, propositional logic, digital logic, elementary number theory, and the fundamentals of counting.

Related UVic Course: Math 222, CENG 290
Prerequisites: Mathematical preparation sufficient to take calculus at the college level.
Syllabus:
• Introduction to logic and proofs: Direct proofs; proof by contradiction; mathematical induction
• Fundamental structures: Functions (surjections, injections, inverses, composition); relations (reflexivity, symmetry, transitivity, equivalence relations); sets (Venn diagrams, complements, Cartesian products, power sets); pigeonhole principle; cardinality and countability
• Boolean algebra: Boolean values; standard operations on Boolean values; de Morgan's laws
• Propositional logic: Logical connectives; truth tables; normal forms (conjunctive and disjunctive); validity
• Digital logic: Logic gates, flip-flops, counters; circuit minimization
• Elementary number theory: Factorability; properties of primes; greatest common divisors and least common multiples; Euclid's algorithm; modular arithmetic; the Chinese Remainder Theorem
• Basics of counting: Counting arguments; pigeonhole principle; permutations and combinations; binomial coefficients
Units covered:
DS1 Functions, relations, and sets 9 hours (6 core + 3)
DS2 Basic logic 5 core hours (of 10)
DS3 Proof techniques 4 core hours (of 12)
DS4 Basics of counting 9 hours (5 core + 4)
AR1 Digital logic and digital systems 3 core hours (of 6)
Elementary number theory 5 hours
Elective topics 5 hours
Notes:
This implementation of the Discrete Structures area (DS) divides the material into two courses. DS 1 covers the first half of the material and is followed by DS 2, which completes the core topic coverage. Although the principal focus is discrete mathematics, the course is likely to be more successful if it highlights applications whose solutions require proof, logic, and counting. For example, the number theory section could be developed in the context of public-key cryptography and security, so that students who tend to focus on the applications side of engineering and computing will have an incentive to learn the underlying theoretical material. For a software engineer and particularly specifications and formal methods (SE 5), logic is particularly relevant.

DS 2: Discrete Structures 2
Continues the discussion of discrete mathematics introduced in DS 1. Topics in the second course include predicate logic, recurrence relations, graphs, trees, matrices, computational complexity, elementary computability, and discrete probability.
Related UVic Course: Math 222, CENG 290
Prerequisites: DS 1
Syllabus:

• Review of previous course
• Predicate logic: Universal and existential quantification; modus ponens and modus tollens; limitations of predicate logic
• Recurrence relations: Basic formulae; elementary solution techniques
• Graphs and trees: Fundamental definitions; simple algorithms ; traversal strategies; proof techniques; spanning trees; applications
• Matrices: Basic properties; applications
• Computational complexity: Order analysis; standard complexity classes
• Elementary computability: Countability and uncountability; diagonalization proof to show uncountability of the reals; definition of the P and NP classes; simple demonstration of the halting problem
• Discrete probability: Finite probability spaces; conditional probability, independence, Bayes' rule; random events; random integer variables; mathematical expectation
Units covered:
DS2 Basic logic 7 core hours (of 10)
DS3 Proof techniques 8 core hours (of 12)
DS5 Graphs and trees 4 core hours
DS6 Discrete probability 6 core hours
AL1 Basic algorithmic analysis 2 core hours (of 4)
AL5 Basic computability 3 core hours (of 6)
AL6 The complexity classes P and NP 2 hours
Matrices 3 hours
Elective topics 5 hours
Notes:
This course introduces mathematical topics in the context of applications that require those concepts as tools. For this course, likely applications include formal methods (i.e., SE 5) or transportation network and resource allocation (i.e., ALGO 1, 2, 3).

BUS1: Engineering Economics and Entrepreneurship
Macroeconomic principles: money, interest rates, growth. Microeconomic principles: demand and supply, production, consumer utility and elasticity. Net present value, equivalence, rate of return. Public vs private sector cost-benefit analysis, externalities, risk and uncertainty. Industry and innovation life cycles. Entrepreneurship: starting and running a business, identifying market need, researching financial viability, and resource requirements (financial, human, technical).
Related UVic Course: ENGR 280
Prerequisites: none
Syllabus:
Units covered: to be written
Notes: Ideally taught by the business school so that students get exposed to business perspectives.

BUS2: Engineering Planning and Management
An introduction to and overview of finance and accounting for engineering management.

Topics include basic accounting concepts and terminology; preparation and interpretation of financial statements; and uses of accounting information for planning, budgeting, decision-making, control, and quality improvement. Price and output decisions. Choosing among alternative inputs and production processes. Evaluating alternative investments, equipment service life, product development, business plan development and marketing.
Related UVic Course: COM 240
Prerequisites: none
Syllabus:
Units covered: to be written
Notes: Ideally taught by the business school so that students get exposed to business perspectives.

SOCIAL: Social and Professional Issues
Introduces students to the legal, social, and professional issues that arise in the context of software engineering practice.
Related UVic Courses: CSc 212, SENG 400, ethics courses, computers and society, law
Prerequisites: introduction to computer science (SE II)
Syllabus:
•*His*tory of computing: Prehistory-the world b*efore 1946*; history of computer hardware, software, networking; pioneers of computing
• Social context of computing: Introduction to the social implications of computing; social implications of networked communication; growth of, control of, and access to the Internet; gender-related issues; international issues
• Methods and tools of analysis: Making and evaluating ethical arguments; identifying and evaluating ethical choices; understanding the social context of design; identifying assumptions and values
• Professional and ethical responsibilities: Community values and the laws by which we live; the nature of professionalism; various forms of professional credentialing and the advantages and disadvantages; the role of the professional in public policy; maintaining awareness of consequences; ethical dissent and whistle-blowing; codes of ethics, conduct, and practice; dealing with harassment and discrimination; "Acceptable use" policies for computing in the workplace
• Risks and liabilities of computer-based systems: Historical examples of software risks; implications of software complexity; risk assessment and management
• Intellectual property: Foundations of intellectual property; copyrights, patents, and trade secrets; software piracy; software patents; transnational issues concerning intellectual property
• Privacy and civil liberties: Ethical and legal basis for privacy protection; privacy implications of massive database systems; technological strategies for privacy protection; freedom of expression in cyberspace; international and intercultural implications
• Computer crime: History and examples of computer crime; "Cracking" and its effects; viruses, worms, and Trojan horses; c
SIG 2: Signal Analysis 2
Related UVic Course: ELEC 310

rime prevention strategies
• Economic issues in computing: Monopolies and their economic implications; effect of skilled labor supply and demand on the quality of computing products; pricing strategies in the computing domain; differences in access to computing resources and the possible effects thereof
• Philosophical frameworks: Philosophical frameworks, particularly utilitarianism and deontological theories; problems of ethical relativism; scientific ethics in historical perspective; differences in scientific and philosophical approaches

Units covered:
SP1 History of computing 1 core hour
SP2 Social context of computing 3 core hours
SP3 Methods and tools of analysis 2 core hours
SP4 Professional and ethical responsibilities 3 core hours
SP5 Risks and liabilities of computer-based systems 2 core hours
SP6 Intellectual property 3 core hours
SP7 Privacy and civil liberties 2 core hours
SP8 Computer crime 3 hours
SP9 Economic issues in computing 2 hours
SP10 Philosophical frameworks 2 hours
Elective topics 17 hours
Notes:
A computer science program can incorporate social and professional issues into the curriculum in many different ways. In many ways, the ideal approach is to include discussion of this material in a wide variety of courses so that students have the chance to consider these issues in the context of each technical area. Unfortunately, this strategy sometimes fails to have the desired effect. Unless faculty members commit to give this material serious consideration, social and professional issues are often given low priority in the context of other courses, to the sometimes wind up being left out altogether in the press to cover more traditional material.
To ensure that students have a real opportunity to study this material, many departments choose to devote an entire course to social and professional issues. Programs that adopt this strategy must make sure that they make the material relevant to students by discussing these issues in the context of concrete examples that arise in computer science.

Engineering Electives

ARCH: Software Architecture
Architectural design of complex software systems. Commonly-used software system structures, techniques for designing and implementing these structures, models and formal notations for characterizing and reasoning about architectures, tools for generating specific instances of an architecture, and case studies of actual system architectures. The role of standards, reuse, and quality. Skills needed to evaluate the architectures of existing systems and to design new systems in principled ways using well-founded

architectural paradigms.
Related UVic Course: SENG 422
Prerequisites: SE 6
Syllabus:
• IEEE recommended practice for software architecture description (3h)
• Introduction to an (standard) Architecture Description Language (ADL) (6h)
• Software architecture construction process (6h): architecture definition process; tradeoff analysis
• Common Architectural Styles (Patterns) and Design Mechanisms (Patterns) (3h)
• Software Architecture as a Collection of Components and Connectors (6h)
• Components, Connectors, and Protocols Design
• Standard Component Model: Corba Component Model (CCM); Microsoft Component Object Model (COM); Enterprise Java Bean Component Model (EJB)
• Distributed Object Architectures (DOA) (6h)
• Selection of a DOA and integration in a software architecture
• CORBA and COM architectures and mechanisms
• Specialized Software Architecture (6h): focus on design issues and strategies specific to these kinds of architectures, and introduction of corresponding architecture models
• Architectures for real-time, embedded, fault-tolerant systems; security software architectures; architectures of legacy systems and product lines
Units covered:
Notes:
The course includes three major themes:
• Software architecture as a blueprint for software design and development: the notion of viewpoints and views; viewpoints definitions; architecture description according to multiple viewpoints.
• Tradeoff Analysis: identification of the factors and quality attributes that characterize a system; identification of underlying design issues and suitable design strategies that will lead the construction of an instance of software architecture. Introduction to software risk analysis and mitigation, with a particular emphasis on architecture related risks.
• Software architecture as a combination of components and connectors: design and/or selection of components and connectors based on tradeoff analysis; standard components models and technologies.

CBSE: Component-Based Software Engineering
Building large-scale and complex software systems from available parts by consistently increasing return on investment and time to market, while assuring high quality and reliability. The course covers advanced topics on software components and component-based software engineering from research and practice.
Related UVic Course: Jens' SENG 480 topics course
Prerequisites: SE 6
Syllabus: to be written by Jens
Units covered:
Notes:

FTC: Fault Tolerant Computing
An introduction to selected issues in fault tolerant computing. Topics include: definitions of reliability, availability, safety, maintainability, testability and dependability; system protection through both hardware and information redundancy; quantitative methods for the evaluation of reliability; the design and test of integrated circuits; software fault tolerance and software testing. The course includes a number of case studies of practical fault tolerant systems.
Related UVic Course: CSC454
Prerequisites: DD and OSDC
Syllabus: to be written by Jon Muzio
Units covered:
Notes:

CSCW: Computer-Supported Collaborative Work
Most of the work that people do requires some degree of coordination and communication with others--some kind of teamwork. But the development of technology to support teamwork has proven to be a considerable challenge in practice. Successful designs require (1) Social psychological insight into group processes, (2) Computer science insight into mechanisms to organize information, coordinate, share, and communicate, and (3) HCI design insight to achieve successful designs for computer-mediated tools. The course focuses primarily on the first two of these factors, examining them in the context of a number of examples of teamwork systems. This course examines problems in group coordination and systems attempting to overcome these problems including group decision support, organizational memory, video conferencing, virtual spaces, awareness and information organization, and collaborative design.
Related UVic Course: none
Prerequisites: SE 6
Syllabus: to be written by Daniela
Units covered:
Notes:

CG: Computer Graphics
Offers an introduction to computer graphics, which has become an increasingly important area within computer science. Computer graphics, particularly in association with the multimedia aspects of the World-Wide Web, have opened up exciting new possibilities for the design of human-computer interfaces. The purpose of this course is to investigate the principles, techniques, and tools that have enabled these advances.
Prerequisites: HCI (LA and SE 3 through HCI)
Syllabus:
• Graphic systems: Raster and vector graphics systems; video display devices; physical and logical input devices; issues facing the developer of graphical systems; scalable vector graphics (SVG)
• Fundamental techniques in graphics: Hierarchy of graphics software; using a graphics

API; simple color models; homogeneous coordinates; affine transformations; viewing transformation; clipping
• Graphical algorithms: Line and circle generation algorithms; incremental algorithms; structure and use of fonts; parametric polynomial curves and surfaces; polygonal representation of 3D objects; introduction to ray tracing; image synthesis, sampling techniques, and anti-aliasing; image enhancement
• Graphical user-interface design: Choosing interaction styles and interaction techniques; HCI aspects of interface design; dynamics of color; structuring a view for effective understanding
• Graphical user-interface programming: Graphical widgets; event management and user interaction; GUI builders and programming environments
• Computer animation: Key-frame animation; camera animation; scripting system; animation of articulated structures; motion capture; procedural animation; deformation
• Virtual reality
• Fractals
• Chaos

Units covered:

AL10 Geometric algorithms 2 hours
HC2 Building a simple graphical user interface 2 core hours
HC4 Human-centered software development 2 hours
HC6 Graphical user-interface programming 5 hours
GV1 Fundamental techniques in graphics 2 core hours
GV2 Graphic systems 1 core hour
GV3 Graphic communication 2 hours
GV4 Geometric modeling 3 hours
GV5 Basic rendering 3 hours
GV8 Computer animation 2 hours
GV10 Virtual reality 2 hours
IM13 Multimedia information and systems 4 hours
SE2 Using APIs 2 core hours (of 5)
Elective topics 1 hour

Notes:

Computer graphics is extremely exciting to students and can serve as an excellent motivator for students, particularly to the extent that the course structure offers students the opportunity to create graphical systems. Although implementation must be a central component of this course, it is equally important to emphasize the mathematical underpinnings of the area, thereby reinforcing the relationship between theory and practice. Software tools play a particularly critical role in this course. While it is useful for students to learn basic principles at an abstract level, it is also essential for them to have exposure to sophisticated graphical libraries, which will vastly extend their ability to construct interesting applications. In addition to programmer-oriented graphical APIs, it may make sense to include other packages-multimedia tools, modeling languages, virtual reality-in this course as well.

Students who complete this course should be able to perform the following tasks:
• Apply the principles that underpin the design of graphics and multimedia systems.
• Describe the range of tools that can be used to support the development of graphical and

multimedia systems.
• Use existing graphics and multimedia packages to develop appropriate graphical applications.

MMS: Multimedia Systems
The influence of technology, especially digital technology, on how we express ourselves, how we communicate with each other, and how we preceive, think about, and interact with our world. The invention and creative use of enabling technologies for understanding and expression by people and machines. Topics include: digital video representations; three-dimensional images; physical interfaces; computational tools and media that help people learn new things in new ways (tele-learning); knowledge representation; machine interpretation of sensory data.
Related UVic Course: SENG 410, CSC 461, ELEC 483
Prerequisites: HCI (LA and SE 3 through HCI)
Syllabus:
Units covered:
Notes:

ALG2: Algorithms and Data Structures 2
This course covers algorithm design and analysis in software engineering. Specific topics include advanced data structures (such as Binomial heaps and Fibonacci Heaps), graph algorithms (such as minimum spanning trees, maximum flow, all-pairs shortest paths, and single-source shortest paths), and advanced design and analysis techniques (such as dynamic programming, greedy algorithms, linear programming, and amortized analysis).
Related UVic Course: CSc 326
Prerequisites: ALG 1
Syllabus:
Units covered:
Notes:

ALG3: Analysis of Algorithms
General techniques for designing and analyzing algorithms; an in depth examination of several problems and algorithms with respect to their time and space requirements; advanced data structures; sorting and searching; graph algorithms; backtracking; NP-complete problems; approximation algorithms.
Related UVic Course: CSC 425
Prerequisites: ALG 2
Syllabus:
Units covered:
Notes:

CON: Concurrency
Systematic treatment of concepts and issues in concurrency and discussion of a wide variety of common concurrency problems. Students learn how to specify, model, and

verify concurrent systems/programs. Formalisms include the process algebra FSP and related algebras such as Milner's CCS and Hoare's CSP. To master the principles of concurrent programming and become proficient in *applying* these principles through programming exercises in multithreading, networking, and simulation.
Related UVic Course: existing topics course in computer science
Prerequisites: OSDC
Syllabus: The first part of the course introduces concurrency theories. The second part deals with system design and formalisms for system design.
• Foundations of Calculus of Communicating Systems (CCS)
• Communicating Sequential Processes (CSP), Petri Nets, Linear Time
• Branching Time Temporal Logic
• Model Checking
• Formalisms for specifying concurrency
• Communicating FSM
• Statecharts
• Petri nets
• Modeling and design issues
• Applications include embedded and safety critical systems
• Understand the limits of our reasoning and tools
• Understanding the difficulties in testing concurrent systems/programs
Units covered:
Notes:

CC: Compiler Construction
Compilation, including: lexical analysis, syntax analysis, semantic routines, code optimization, block structured languages and interpreters. Students will implement a compiler-interpreter for a simple language
Related UVic Course: CSC 435
Prerequisites: AFL
Syllabus:
Units covered:
Notes:

PL: Programming Languages
The fundamental concepts of imperative, object-oriented, applicative, and logic programming languages
Related UVic Course: CSC 330
Prerequisites: AFL
Syllabus:
Units covered:
Notes:

AI: Artificial Intelligence
Philosophy of artificial intelligence. AI programs and languages, representations and descriptions, exploiting constraints. Rule based and heuristic systems. Applications to engineering.
Related UVic Course: CENG 420
Prerequisites: ALG 1
Syllabus:
Units covered:
Notes:

PATREC: Pattern Recognition
Parallel and sequential recognition methods. Bayesian decision procedures, perceptrons, statistical and syntactic approaches, recognition grammars. Feature extraction and selection, scene analysis, and optical character recognition.
Related UVic Course: ELEC 485
Prerequisites: AFL
Syllabus:
Units covered:
Notes:

ROBOT: Robotics
Structure and specification of robot manipulators. Homogenous transformations. Link description. Manipulator kinematics. Inverse manipulator kinematics. Velocity and static forces in manipulators. An introduction to manipulator dynamics. Linear control of robot motion. Model-based nonlinear control of robot manipulators.
Related UVic Course: ELEC 426, MECH 430
Prerequisites: MECHSYS
Syllabus:
Units covered:
Notes:

IKM: Information and Knowledge Management
Uses the idea of information as a unifying theme to investigate a range of issues in computer science, including database systems, artificial intelligence, human-computer interaction, multimedia system, and data communication.
Related UVic Course: CENG420
Prerequisites: DB (SE 4, DS 2, WEBENG through DB)
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; use of

a database query language
• 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
• 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
• Fundamental issues in intelligent systems: History of artificial intelligence; the role of heuristics
• Cryptographic algorithms: Historical overview of cryptography; private-key cryptography and the key-exchange problem; public-key cryptography; digital signatures; security protocols
• Introduction to compression and decompression: Encoding and decoding algorithms; lossless and lossy compression
• Multimedia information and systems
• Intellectual property: Foundations of intellectual property; copyrights, patents, and trade secrets; software piracy; software patents; transnational issues concerning intellectual property
• Privacy and civil liberties: Ethical and legal basis for privacy protection; privacy implications of massive database systems; technological strategies for privacy protection; freedom of expression in cyberspace; international and intercultural implications

Units covered:

AL9 Cryptographic algorithms 3 hours
NC7 Compression and decompression 2 hours
IS1 Fundamental issues in intelligent systems 1 core hour
IS2 Search and constraint satisfaction 5 core hours
IS3 Knowledge representation and reasoning 4 core hours
IM1 Information models and systems 3 core hours
IM2 Database systems 3 core hours
IM3 Data modeling 4 core hours
IM4 Relational databases 4 hours
IM13 Multimedia information and systems 2 hours
SP6 Intellectual property 3 core hours
SP7 Privacy and civil liberties 2 core hours

Notes:

Given that it addresses a mix of topics from such areas as databases, artificial intelligence, and human-computer interaction, it is unlikely that this course appears in existing curricula. We believe, however, that courses of this sort, which take a unifying theme and use that to provide structure to an otherwise diverse set of topics, provide a useful way to develop a “crosscutting core” that focuses on broad themes rather than specific artifacts. In this case, the broad theme is that of the management, representation, and manipulation of information. It addresses, for example, the entire area of storing, retrieving, encoding, and managing information, whether for database use, intelligent

systems use, telecommunications, or graphics. It also addresses the social and ethical issues related to information management, such as the ownership of intellectual property and individual privacy rights.

NC: Network-Centric Computing (Architectures for Networking and Communication)

Presents those aspects of computer architecture that are central to communications and networking.

Related UVic Course: SENG 450 Network-centric computing, CSC450, CENG 460

Prerequisites: WEBENG and OS

Syllabus:

• Distributed algorithms: Consensus and election; termination detection; fault tolerance; stabilization

• Interfacing and communication: I/O fundamentals; interrupt structures; external storage, physical organization, and drives; buses; introduction to networks; multimedia support; RAID architectures

• Multiprocessing and alternative architectures: Introduction to SIMD, MIMD, VLIW, EPIC; systolic architecture; interconnection networks; shared memory systems; cache coherence; memory models and memory consistency

• Architecture for networks and distributed systems: Introduction to LANs and WANs; layered protocol design, ISO/OSI, IEEE 802; impact of architectural issues on distributed algorithms; network computing; distributed multimedia

• Concurrency: States and state diagrams; structures; dispatching and context switching; the role of interrupts; concurrent execution; the "mutual exclusion" problem and some solutions; deadlock; models and mechanisms; producer-consumer problems and synchronization; multiprocessor issues

• Scheduling and dispatch: Review of processes and scheduling; deadlines and real-time issues

• Real-time and embedded systems: Process and task scheduling; memory/disk management requirements in a real-time environment; failures, risks, and recovery; special concerns in real-time systems

• Fault tolerance: Fundamental concepts; spatial and temporal redundancy; methods used to implement fault tolerance; examples of reliable systems

• System performance evaluation: Why system performance needs to be evaluated; what is to be evaluated; policies for caching, paging, scheduling, memory management, security, and so forth; evaluation models; how to collect evaluation data

• Scripting: Scripting and the role of scripting languages; basic system commands; creating scripts, parameter passing; executing a script; influences of scripting on programming

Units covered:

AL4 Distributed algorithms 3 core hours

AR5 Interfacing and communication 3 core hours

AR7 Multiprocessing and alternative architectures 3 core hours

AR9 Architecture for networks and distributed systems 5 hours

OS3 Concurrency 4 core hours

OS4 Scheduling and dispatch 2 core hours
OS9 Real-time and embedded systems 5 hours
OS10 Fault tolerance 5 hours
OS11 System performance evaluation 4 hours
Elective topics 3 hours

DC: Distributed Computing
Introduces concurrency in the context of distributed systems. The course covers both the abstract principles of concurrent programming and their concrete realization in distributed, network-based systems. Topics include the basic theory of concurrency, hardware and software features to support concurrency, concurrent and distributed algorithms, and middleware.
Related UVic Course: SENG 462, CSC 462
Prerequisites: OSNET (SE 4 through OSNET)
Syllabus:
• Concurrent model of execution: Interleaving; atomic operations; critical sections and mutual exclusion ; deadlock; starvation; invariants
• Concurrent and distributed algorithms: producer-consumer; reader -writer problems; dining philosophers
• Architectural features to support concurrent and distributed systems
• Language features for concurrent and distributed systems
• Performance evaluation
• Middleware
Units covered:
• Distributed algorithms 3 core hours
• Parallel algorithms 2 hours
• Architecture for networks and distributed systems 3 hours
• Concurrency 4 core hours
• Network security 2 core hours
• Wireless and mobile computing 4 hours
Notes: Most operating systems support some form of concurrent and distributed development. It is important therefore to address the issues that force the programmer to be concerned with the intricate problems associated with the development of these systems. In terms of applications, the web offers a rich vein of possibilities. Students who complete this course should be able to perform the following tasks:
• Outline the potential benefits of concurrent and distributed systems.
• Apply standard design principles in the construction of these systems.
• Select appropriate approaches for building a range of distributed systems, including some that employ middleware.
• Summarize the major security issues associated with distributed systems along with the range of techniques available for increasing system security.

MGMT ECON: Management and Engineering Economics
Basic microeconomic theory and optimization techniques and their application to

managerial decision making. Topics include demand, production, and cost analysis; market structure and pricing practices; and regulation. Course also examines estimation, forecasting, international implications, and case studies.
Related UVic Course: ECON 205
Prerequisites: none
Syllabus:
Units covered: to be written
Notes: none

MGMT FIN: Management Finance
This course serves as an introduction to corporate financial management. The primary objective is to provide a framework, concepts, and tools for analyzing financial decisions. Main topics include discounted cash flow techniques, financial statement analysis, capital budgeting, valuation of stocks and bonds, tax environments, risk and return tradeoffs, diversification, capital market efficiency, and an introduction to international finance issues.
Related UVic Course: COM 240
Prerequisites: none
Syllabus:
Units covered: to be written
Notes: none

MGMT MKT: Fundamentals of Marketing
Product design and management, distribution channels, and marketing communications are examined as key elements of the marketing mix. Consumer buyer behaviour, sales force management, and marketing research are other topics to be reviewed.
Related UVic Course: COM 250
Prerequisites: none
Syllabus:
Units covered: to be written
Notes: none

ENG: Introduction to Engineering
Introduction to engineering disciplines and careers, role of the engineer in society, engineering approach to problem-solving, engineering design process, concurrent engineering, and engineering ethics.
Related UVic Course: SOCIAL
Prerequisites: none
Syllabus:
• Academic success skills and time management
• Communication skills and requirements on the job
• Group dynamics, team work, and diversity
• Electrical engineering

• Mechanical engineering
• Industrial engineering
• Computer engineering
• Software engineering
• Concurrent engineering philosophy and methods
• Technical competition
• Safety, concurrent engineering
• Ethics and ecological considerations
Units covered:
Notes: Could be used instead of ELECSYS and MECHSYS; some aspects covered by SOCIAL

DSP: Digital Signal Processing
Generation of discrete-time signals through the sampling process and their spectral representation. Mathematical representation and properties of digital signal processing (DSP) systems. Typical DSP systems: digital filters and applications. The z-transform and its relation to the Laurent series. Evaluation of the inverse z-transform using complex series and contour integrals. Application of the z-transform for the representation and analysis of DSP systems. The processing of continuous-time signals using DSP systems. The discrete-Fourier transform and the use of fast Fourier transforms for its evaluation. Introduction to the design of DSP systems. Design project using Matlab.
Related UVic Course: ELEC 310
Prerequisites: CALC 3, DS 2, SAS
Syllabus:
• Discrete time and sampled data signals
• Z-transforms
• Math needed for inverse Z-transforms: complex integrals
• Sequences and series
• Power series, Laurent series
• Residue theorem
• Applications of Z-transforms: sampling theorem, digital filter design
• Discrete Fourier Transform
Units covered: to be written
Notes: none

COM: Digital Communications
The course is concerned with the transmission, communication and processing of signals (information) and the necessary technical equipment for these purposes. Introduction to protocol engineering; PDU encoders and decoders; buffer management in communications programs; real-time constrains; timer management. Land and satellite-based mobile radio services, systems and networks. Mobile cellular telephone, paging, telepoint and wireless LAN systems. Switching and other protocols in support of mobility. Frequency reuse and channel allocation. Multiple access methods. Architectures of mobile distributed computing systems. Future developments in mobile

telecommunications and associated new design problems. Communication via optical fibres.
Related UVic Course: ELEC 350
Prerequisites: DSP
Syllabus:
Units covered: to be written
Notes: none

WMC: Wireless and Mobile Computing
This course focuses on the design and implementation of wireless and mobile computing *solutions*. The students study emerging technologies such as Jini, WAP, IEEE802.11, and Bluetooth. Targetted applications include handheld and mobile devices such as the Palm Connected Organizer, Handspring Visor, and PocketPC/WindowsCE devices such as the Compaq iPaq and HP Jornada.
Related UVic Course: CENG 440, ELEC 456
Prerequisites: SIG 2
Syllabus:
Units covered: to be written
Notes: none

NA: Numerical Methods
The study of computational methods for solving problems in linear algebra, nonlinear equations, approximation, and ordinary differential equations. The student will write programs in a suitable high level language to solve problems in some of the areas listed above but the course will also teach the student how to use mathematical subroutine packages currently available in computer libraries.
Related UVic Course: CSC 340
Prerequisites: SE 2, LA
Syllabus:
Units covered: to be written
Notes: none

ORLP: Operations Research: Linear Programming
An introduction to linear programming and its applications. Topics include: the simplex method, the revised simplex method, computer implementations, duality. Optional topics include: parametric and sensitivity analysis, primal-dual algorithm, network simplex method, the network flow problem, and game theory. Typical applications include: fitting curves to data, the transportation problem, inventory problems and blending problems.
Related UVic Course: CSC 445
Prerequisites: CSC 340
Syllabus:
Units covered: to be written
Notes: none

ORSIM: Operations Research: Simulation
An introduction to discrete event simulation. Topics include: elementary queueing theory, basic techniques of discrete event simulation, generating random numbers, sampling from non-uniform distributions, simulation programming using general purpose languages and also special purpose simulation languages.
Related UVic Course: CSC 446
Prerequisites: CSC 340
Syllabus:
Units covered: to be written
Notes: none

MIN: Data Mining
An introduction to data mining in the context of customer relationship management. Data preparation, model building, and data mining techniques such as clustering, decision trees and neural networks will be discussed and applied to case studies. Data-mining software tools will be reviewed and compared.
Related UVic Course: Topics course taught by Vlad, ELEC 485 (pattern recognition)
Prerequisites: DB
Syllabus:
• Describe the data mining process and explain why automated data mining tools alone cannot provide usable and reliable business intelligence.
• Describe the three core areas of data mining necessary for a successful data mining project.
• Identify and discuss the strengths and weaknesses of data mining techniques and recognize which technique is most appropriate for a given business problem.
• Describe how to analyze, clean and prepare data for a data mining project.
• Discuss predictive models and their pitfalls.
• Compare and contrast commercial data mining software tools.
• Use data mining techniques in case studies.
• Discuss data mining and privacy.
Units covered: to be written
Notes: none

Part II: Software engineering knowledge units

Programming Fundamentals (PF)

PF1. Fundamental programming constructs [core]
PF2. Algorithms and problem-solving [core]
PF3. Fundamental data structures [core]
PF4. Recursion [core]

PF5. Event-driven programming [core]

Fluency in a programming language is prerequisite to the study of most of software engineering. In the CC1991 report, knowledge of a programming language-while identified as essential-was given little emphasis in the curriculum. The "Introduction to a Programming Language" area in CC1991 represents only 12 hours of class time and is identified as optional, under the optimistic assumption that "increasing numbers of students . . . gain such experience in secondary school." We believe that undergraduate software engineering, computer science, computer engineering, and software engineering programs must teach students how to use at least one programming language well; furthermore, we recommend that software computing programs should teach students to become competent in languages that make use of at least two programming paradigms. We recommend that the University of Victoria SEDP students become proficient in Java and C/C++. Accomplishing this goal requires considerably more than 12 hours.

This knowledge area consists of those skills and concepts that are essential to programming practice independent of the underlying paradigm. As a result, this area includes units on fundamental programming concepts, basic data structures, and algorithmic processes. These units, however, by no means cover the full range of programming knowledge that a computer science undergraduate must know. Many of the other areas-most notably Programming Languages (PL) and Software Engineering (SE)-also contain programming-related units that are part of the undergraduate core. In most cases, these units could equally well have been assigned to either Programming Fundamentals or the more advanced area.

PF1. Fundamental programming constructs [core]
Minimum core coverage time: 9 hours
Topics:
Basic syntax and semantics of a higher-level language
Variables, types, expressions, and assignment
Simple I/O
Conditional and iterative control structures
Functions and parameter passing
Structured decomposition
Assertions
Learning objectives:
1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.
2. Modify and expand short programs that use standard conditional and iterative control structures and functions.
3. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.
4. Choose appropriate conditional and iteration constructs for a given programming task.
5. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.

6. Describe the mechanics of parameter passing.

PF2. Algorithms and problem-solving [core]
Minimum core coverage time: 6 hours
Topics:
Problem-solving strategies
The role of algorithms in the problem-solving process
Implementation strategies for algorithms
Debugging strategies
The concept and properties of algorithms
Learning objectives:
1. Discuss the importance of algorithms in the problem-solving process.
2. Identify the necessary properties of good algorithms.
3. Create algorithms for solving simple problems.
4. Use pseudo-code or a programming language to implement, test, and debug algorithms for solving simple problems.
5. Describe strategies that are useful in debugging.

PF3. Fundamental data structures [core]
Minimum core coverage time: 14 hours
Topics:
Primitive types
Arrays
Records
Strings and string processing
Data representation in memory
Static, stack, and heap allocation
Runtime storage management
Pointers and references
Linked structures
Implementation strategies for stacks, queues, and hash tables
Implementation strategies for graphs and trees
Strategies for choosing the right data structure
Learning objectives:
1. Discuss the representation and use of primitive data types and built-in data structures.
2. Describe how the data structures in the topic list are allocated and used in memory.
3. Describe common applications for each data structure in the topic list.
4. Implement the user-defined data structures in a high-level language.
5. Compare alternative implementations of data structures with respect to performance.
6. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables.
7. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
8. Choose the appropriate data structure for modeling a given problem.

PF4. Recursion [core]
Minimum core coverage time: 5 hours

Topics:
The concept of recursion
Recursive mathematical functions
Simple recursive procedures
Divide-and-conquer strategies
Recursive backtracking
Implementation of recursion
Learning objectives:
1. Describe the concept of recursion and give examples of its use.
2. Identify the base case and the general case of a recursively defined problem.
3. Compare iterative and recursive solutions for elementary problems such as factorial.
4. Describe the divide-and-conquer approach.
5. Implement, test, and debug simple recursive functions and procedures.
6. Describe how recursion can be implemented using a stack.
7. Discuss problems for which backtracking is an appropriate solution.
8. Determine when a recursive solution is appropriate for a problem.

PF5. Event-driven programming [core]
Minimum core coverage time: 4 hours
Topics:
Event-handling methods
Event propagation
Exception handling
Learning objectives:
1. Explain the difference between event-driven programming and command-line programming.
2. Design, code, test, and debug simple event-driven programs that respond to user events.
3. Develop code that responds to exception conditions raised during execution.

Software Engineering (SE)

SE1. Software design [core]
SE2. Using APIs [core]
SE3. Software tools and environments [core]
SE4. Software processes [core]
SE5. Software requirements and specifications [core]
SE6. Software validation [core]
SE7. Software evolution [core]
SE8. Software project management [core]
SE9. Component-based computing [core]
SE10. Formal methods [core]
SE11. Software reliability [core]
SE12. Specialized systems development [core]
SE13. Testing [core]
SE14. Software maintenance [core]

SE15. Software configuration management [core]
SE16. Program understanding [core]
SE17. Reverse engineering [core]
SE18. Software quality [core]
SE19. Software metrics [core]

Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building software systems that satisfy the requirements of users and customers. Software engineering is applicable to small, medium, and large-scale systems. It encompasses all phases of the life cycle of a software system. The life cycle includes requirement analysis and specification, design, construction, testing, and operation and maintenance.

Software engineering employs engineering methods, processes, techniques, and measurement. It benefits from the use of tools for managing software development; analyzing and modeling software artifacts; assessing and controlling quality; and for ensuring a disciplined, controlled approach to software evolution and reuse. Software development, which can involve an individual developer or a team of developers, requires choosing the tools, methods, and approaches that are most applicable for a given development environment.

The elements of software engineering are applicable to the development of software in any computing application domain where professionalism, quality, schedule, and cost are important in producing a software system.

SE1. Software design [core]
Minimum core coverage time: 8 hours
Topics:
Fundamental design concepts and principles
Design patterns
Software architecture
Structured design
Object-oriented analysis and design
Component-level design
Design for reuse
Learning objectives:
1. Discuss the properties of good software design.
2. Compare and contrast object-oriented analysis and design with structured analysis and design.
3. Evaluate the quality of multiple software designs based on key design principles and concepts.
4. Select and apply appropriate design patterns in the construction of a software application.
5. Create and specify the software design for a medium-size software product using a software requirement specification, an accepted program design methodology (e.g., structured or object-oriented), and appropriate design notation.

6. Conduct a software design review using appropriate guidelines.
7. Evaluate a software design at the component level.
8. Evaluate a software design from the perspective of reuse.

SE2. Using APIs [core]
Minimum core coverage time: 5 hours
Topics:
API programming
Class browsers and related tools
Programming by example
Debugging in the API environment
Introduction to component-based computing
Learning objectives:
1. Explain the value of application programming interfaces (APIs) in software development.
2. Use class browsers and related tools during the development of applications using APIs.
3. Design, implement, test, and debug programs that use large-scale API packages.

SE3. Software tools and environments [core]
Minimum core coverage time: 3 hours
Topics:
Programming environments
Requirements analysis and design modeling tools
Testing tools
Configuration management tools
Tool integration mechanisms
Learning objectives:
1. Select, with justification, an appropriate set of tools to support the development of a range of software products.
2. Analyze and evaluate a set of tools in a given area of software development (e.g., management, modeling, or testing).
3. Demonstrate the capability to use a range of software tools in support of the development of a software product of medium size.

SE4. Software processes [core]
Minimum core coverage time: 2 hours
Topics:
Software life-cycle and process models
Process assessment models
Software process metrics
Learning objectives:
1. Explain the software life cycle and its phases including the deliverables that are produced.
2. Select, with justification the software development models most appropriate for the development and maintenance of a diverse range of software products.
3. Explain the role of process maturity models.

4. Compare the traditional waterfall model to the incremental model, the object-oriented model, and other appropriate models.
5. For each of various software project scenarios, describe the project's place in the software life cycle, identify the particular tasks that should be performed next, and identify metrics appropriate to those tasks.

SE5. Software requirements and specifications [core]
Minimum core coverage time: 4 hours
Topics:
Requirements elicitation
Requirements analysis modeling techniques
Functional and nonfunctional requirements
Prototyping
Basic concepts of formal specification techniques
Learning objectives:
1. Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system.
2. Discuss the challenges of maintaining legacy software.
3. Use a common, non-formal method to model and specify (in the form of a requirements specification document) the requirements for a medium-size software system.
4. Conduct a review of a software requirements document using best practices to determine the quality of the document.
5. Translate into natural language a software requirements specification written in a commonly used formal specification language.

SE6. Software validation [core]
Minimum core coverage time: 3 hours
Topics:
Validation planning
Testing fundamentals, including test plan creation and test case generation
Black-box and white-box testing techniques
Unit, integration, validation, and system testing
Object-oriented testing
Inspections
Learning objectives:
1. Distinguish between program validation and verification.
2. Describe the role that tools can play in the validation of software.
3. Distinguish between the different types and levels of testing (unit, integration, systems, and acceptance) for medium-size software products.
4. Create, evaluate, and implement a test plan for a medium-size code segment.
5. Undertake, as part of a team activity, an inspection of a medium-size code segment.
6. Discuss the issues involving the testing of object-oriented software.

SE7. Software evolution [core]
Minimum core coverage time: 3 hours
Topics:

Software maintenance
Characteristics of maintainable software
Reengineering
Legacy systems
Software reuse
Learning objectives:
1. Identify the principal issues associated with software evolution and explain their impact on the software life cycle.
2. Discuss the challenges of maintaining legacy systems and the need for reverse engineering.
3. Outline the process of regression testing and its role in release management.
4. Estimate the impact of a change request to an existing product of medium size.
5. Develop a plan for re-engineering a medium-sized product in response to a change request.
6. Discuss the advantages and disadvantages of software reuse.
7. Exploit opportunities for software reuse in a given context.

SE8. Software project management [core]
Minimum core coverage time: 3 hours
Topics:
Team management
- Team processes
- Team organization and decision-making
- Roles and responsibilities in a software team
- Role identification and assignment
- Project tracking
- Team problem resolution
Project scheduling
Software measurement and estimation techniques
Risk analysis
Software quality assurance
Software configuration management
Project management tools
Learning objectives:
1. Demonstrate through involvement in a team project the central elements of team building and team management.
2. Prepare a project plan for a software project that includes estimates of size and effort, a schedule, resource allocation, configuration control, change management, and project risk identification and management.
3. Compare and contrast the different methods and techniques used to assure the quality of a software product.

SE9. Component-based computing [elective]
Topics:
Fundamentals
- The definition and nature of components
- Components and interfaces

- Interfaces as contracts
- The benefits of components
Basic techniques
- Component design and assembly
- Relationship with the client-server model and with patterns
- Use of objects and object lifecycle services
- Use of object brokers
- Marshalling
Applications (including the use of mobile components)
Architecture of component-based systems
Component-oriented design
Event handling: detection, notification, and response
Middleware
- The object-oriented paradigm within middleware
- Object request brokers
- Transaction processing monitors
- Workflow systems
- State-of-the-art tools
Learning objectives:
1. Explain and apply recognized principles to the building of high-quality software components.
2. Discuss and select an architecture for a component-based system suitable for a given scenario.
3. Identify the kind of event handling implemented in one or more given APIs.
4. Explain the role of objects in middleware systems and the relationship with components.
5. Apply component-oriented approaches to the design of a range of software including those required for concurrency and transactions, reliable communication services, database interaction including services for remote query and database management, secure communication and access.

SE10. Formal methods [elective]
Topics:
Formal methods concepts
Formal specification languages
Executable and non-executable specifications
Pre and post assertions
Formal verification
Learning objectives:
1. Apply formal verification techniques to software segments with low complexity.
2. Discuss the role of formal verification techniques in the context of software validation and testing.
3. Explain the potential benefits and drawbacks of using formal specification languages.
4. Create and evaluate pre- and post-assertions for a variety of situations ranging from simple through complex.
5. Using a common formal specification language, formulate the specification of a simple

software system and demonstrate the benefits from a quality perspective.

SE11. Software reliability [elective]
Topics:
Software reliability models
Redundancy and fault tolerance
Defect classification
Probabilistic methods of analysis
Learning objectives:
1. Demonstrate the ability to apply multiple methods to develop reliability estimates for a software system.
2. Identify and apply redundancy and fault tolerance for a medium-sized application.
3. Explain the problems that exist in achieving very high levels of reliability.
4. Identify methods that will lead to the realization of a software architecture that achieves a specified reliability level.

SE12. Specialized systems development [elective]
Topics:
Real-time systems
Client-server systems
Distributed systems
Parallel systems
Web-based systems
High-integrity systems
Learning objectives:
1. Identify and discuss different specialized systems.
2. Discuss life cycle and software process issues in the context of software systems designed for a specialized context.
3. Select, with appropriate justification, approaches that will result in the efficient and effective development and maintenance of specialized software systems.
4. Given a specific context and a set of related professional issues, discuss how a software engineer involved in the development of specialized systems should respond to those issues.
5. Outline the central technical issues associated with the implementation of specialized systems development.

Net-Centric Computing (NC)

NC1. Introduction to net-centric computing [core]
NC2. Communication and networking [core]
NC3. Network security [core]
NC4. The web as an example of client-server computing [core]
NC5. Building web applications [elective]
NC6. Network management [elective]
NC7. Compression and decompression [elective]
NC8. Multimedia data technologies [elective]
NC9. Wireless and mobile computing [elective]

Recent advances in computer and telecommunications networking, particularly those based on TCP/IP, have increased the importance of networking technologies in the computing discipline. Net-centric computing covers a range of sub-specialties including: computer communication network concepts and protocols, multimedia systems, Web standards and technologies, network security, wireless and mobile computing, and distributed systems.

Mastery of this subject area involves both theory and practice. Learning experiences that involve hands-on experimentation and analysis are strongly recommended as they reinforce student understanding of concepts and their application to real-world problems. Laboratory experiments should involve data collection and synthesis, empirical modeling, protocol analysis at the source code level, network packet monitoring, software construction, and evaluation of alternative design models. All of these are important concepts that can best understood by laboratory experimentation.

NC1. Introduction to net-centric computing [core]
Minimum core coverage time: 2 hours
Topics:
Background and history of networking and the Internet
Network architectures
The range of specializations within net-centric computing
- Networks and protocols
- Networked multimedia systems
- Distributed computing
- Mobile and wireless computing

Learning objectives:
1. Discuss the evolution of early networks and the Internet.
2. Demonstrate the ability to use effectively a range of common networked applications including e-mail, telnet, FTP, newsgroups, and web browsers, online web courses, and instant messaging.
3. Explain the hierarchical, layered structure of a typical network architecture.
4. Describe emerging technologies in the net-centric computing area and assess their current capabilities, limitations, and near-term potential.

NC2. Communication and networking [core]
Minimum core coverage time: 7 hours
Topics:
Network standards and standardization bodies
The ISO 7-layer reference model in general and its instantiation in TCP/IP
Circuit switching and packet switching
Streams and datagrams
Physical layer networking concepts (theoretical basis, transmission media, standards)
Data link layer concepts (framing, error control, flow control, protocols)
Internetworking and routing (routing algorithms, internetworking, congestion control)

Transport layer services (connection establishment, performance issues)
Learning objectives:
1. Discuss important network standards in their historical context.
2. Describe the responsibilities of the first four layers of the ISO reference model.
3. Discuss the differences between circuit switching and packet switching along with the advantages and disadvantages of each.
4. Explain how a network can detect and correct transmission errors.
5. Illustrate how a packet is routed over the Internet.
6. Install a simple network with two clients and a single server using standard host-configuration software tools such as DHCP.

NC3. Network security [core]
Minimum core coverage time: 3 hours
Topics:
Fundamentals of cryptography
Secret-key algorithms
Public-key algorithms
Authentication protocols
Digital signatures
Examples
Learning objectives:
1. Discuss the fundamental ideas of public-key cryptography.
2. Describe how public-key cryptography works.
3. Distinguish between the use of private- and public-key algorithms.
4. Summarize common authentication protocols.
5. Generate and distribute a PGP key pair and use the PGP package to send an encrypted e-mail message.
6. Summarize the capabilities and limitations of the means of cryptography that are conveniently available to the general public.

NC4. The web as an example of client-server computing [core]
Minimum core coverage time: 3 hours
Topics:
Web technologies
- Server-side programs
- Common gateway interface (CGI) programs
- Client-side scripts
- The applet concept

Characteristics of web servers
- Handling permissions
- File management
- Capabilities of common server architectures

Role of client computers
Nature of the client-server relationship
Web protocols
Support tools for web site creation and web management
Developing Internet information servers

Publishing information and applications
Learning objectives:
1. Explain the different roles and responsibilities of clients and servers for a range of possible applications.
2. Select a range of tools that will ensure an efficient approach to implementing various client-server possibilities.
3. Design and build a simple interactive web-based application (e.g., a simple web form that collects information from the client and stores it in a file on the server).

NC5. Building web applications [elective]
Topics:
Protocols at the application layer
Principles of web engineering
Database-driven web sites
Remote procedure calls (RPC)
Lightweight distributed objects
The role of middleware
Support tools
Security issues in distributed object systems
Enterprise-wide web-based applications
Learning objectives:
1. Illustrate how interactive client-server web applications of medium size can be built using different types of Web technologies.
2. Demonstrate how to implement a database-driven web site, explaining the relevant technologies involved in each tier of the architecture and the accompanying performance tradeoffs.
3. Implement a distributed system using any two distributed object frameworks and compare them with regard to performance and security issues.
4. Discuss security issues and strategies in an enterprise-wide web-based application.

NC6. Network management [elective]
Topics:
Overview of the issues of network management
Use of passwords and access control mechanisms
Domain names and name services
Issues for Internet service providers (ISPs)
Security issues and firewalls
Quality of service issues: performance, failure recovery
Learning objectives:
1. Explain the issues for network management arising from a range of security threats, including viruses, worms, Trojan horses, and denial-of-service attacks
2. Summarize the strengths and weaknesses associated with different approaches to security.
3. Develop a strategy for ensuring appropriate levels of security in a system designed for a particular purpose.
4. Implement a network firewall.

NC7. Compression and decompression [elective]
Topics:
Analog and digital representations
Encoding and decoding algorithms
Lossless and lossy compression
Data compression: Huffman coding and the Ziv-Lempel algorithm
Audio compression and decompression
Image compression and decompression
Video compression and decompression
Performance issues: timing, compression factor, suitability for real-time use
Learning objectives:
1. Summarize the basic characteristics of sampling and quantization for digital representation.
2. Select, giving reasons that are sensitive to the specific application and particular circumstances, the most appropriate compression techniques for text, audio, image, and video information.
3. Explain the asymmetric property of compression and decompression algorithms.
4. Illustrate the concept of run-length encoding.
5. Illustrate how a program like the UNIX **`compress`** utility, which uses Huffman coding and the Ziv-Lempel algorithm, would compress a typical text file.

NC8. Multimedia data technologies [elective]
Topics:
Sound and audio, image and graphics, animation and video
Multimedia standards (audio, music, graphics, image, telephony, video, TV)
Capacity planning and performance issues
Input and output devices (scanners, digital camera, touch-screens, voice-activated)
MIDI keyboards, synthesizers
Storage standards (Magneto Optical disk, CD-ROM, DVD)
Multimedia servers and file systems
Tools to support multimedia development

Learning objectives:
1. For each of several media or multimedia standards, describe in non-technical language what the standard calls for, and explain how aspects of human perception might be sensitive to the limitations of that standard.
2. Evaluate the potential of a computer system to host one of a range of possible multimedia applications, including an assessment of the requirements of multimedia systems on the underlying networking technology.
3. Describe the characteristics of a computer system (including identification of support tools and appropriate standards) that has to host the implementation of one of a range of possible multimedia applications.
4. Implement a multimedia application of modest size.

NC9. Wireless and mobile computing [elective]
Topics:

Overview of the history, evolution, and compatibility of wireless standards
The special problems of wireless and mobile computing
Wireless local area networks and satellite-based networks
Wireless local loops
Mobile Internet protocol
Mobile aware adaption
Extending the client-server model to accommodate mobility
Mobile data access: server data dissemination and client cache management
Software package support for mobile and wireless computing
The role of middleware and support tools
Performance issues
Emerging technologies
Learning objectives:
1. Describe the main characteristics of mobile IP and explain how differs from IP with regard to mobility management and location management as well as performance.
2. Illustrate (with home agents and foreign agents) how e-mail and other traffic is routed using mobile IP.
3. Implement a simple application that relies on mobile and wireless data communications.
4. Describe areas of current and emerging interest in wireless and mobile computing, and assess the current capabilities, limitations, and near-term potential of each.

Social and Professional Issues (SP)

SP1. History of computing [core]
SP2. Social context of computing [core]
SP3. Methods and tools of analysis [core]
SP4. Professional and ethical responsibilities [core]
SP5. Risks and liabilities of computer-based systems [core]
SP6. Intellectual property [core]
SP7. Privacy and civil liberties [core]
SP8. Computer crime [elective]
SP9. Economic issues in computing [elective]
SP10. Philosophical frameworks [elective]

Although technical issues are obviously central to any computing curriculum, they do not by themselves constitute a complete educational program in the field. Students must also develop an understanding of the social and professional context in which computing is done.

This need to incorporate the study of social issues into the curriculum was recognized in the following excerpt from Computing Curricula 1991 [Tucker91]:

> Undergraduates also need to understand the basic cultural, social, legal, and ethical issues inherent in the discipline of computing. They should understand where the discipline has been, where it is, and where it is heading. They should also understand their individual roles in this process, as well as appreciate the philosophical questions, technical problems, and aesthetic values that play an important part in the development of the discipline. Students also need to develop the ability to

> ask serious questions about the social impact of computing and to evaluate proposed answers to those questions. Future practitioners must be able to anticipate the impact of introducing a given product into a given environment. Will that product enhance or degrade the quality of life? What will the impact be upon individuals, groups, and institutions? Finally, students need to be aware of the basic legal rights of software and hardware vendors and users, and they also need to appreciate the ethical values that are the basis for those rights. Future practitioners must understand the responsibility that they will bear, and the possible consequences of failure. They must understand their own limitations as well as the limitations of their tools. All practitioners must make a long-term commitment to remaining current in their chosen specialties and in the discipline of computing as a whole.

The material in this knowledge area is best covered through a combination of one required course along with short modules in other courses. On the one hand, some units listed as core-in particular, SP2, SP3, SP4, and SP6-do not readily lend themselves to being covered in other traditional courses. Without a standalone course, it is difficult to cover these topics appropriately. On the other hand, if ethical considerations are covered only in the standalone course and not "in context," it will reinforce the false notion that technical processes are void of ethical issues. Thus it is important that several traditional courses include modules that analyze ethical considerations in the context of the technical subject matter of the course. Courses in areas such as software engineering, databases, computer networks, and introduction to computing provide obvious context for analysis of ethical issues. However, an ethics-related module could be developed for almost any course in the curriculum. It would be explicitly against the spirit of the recommendations to have only a standalone course. Running through all of the issues in this area is the need to speak to the computer practitioner's responsibility to proactively address these issues by both moral and technical actions.

The ethical issues discussed in any class should be directly related to and arise naturally from the subject matter of that class. Examples include a discussion in the database course of data aggregation or data mining, or a discussion in the software engineering course of the potential conflicts between obligations to the customer and obligations to the user and others affected by their work. Programming assignments built around applications such as controlling the movement of a laser during eye surgery can help to address the professional, ethical and social impacts of computing.

SP1. History of computing [core]
Minimum core coverage time: 1 hour
Topics:
Prehistory-the world before 1946
History of computer hardware, software, networking
Pioneers of computing
Learning objectives:
1. List the contributions of several pioneers in the computing field.
2. Compare daily life before and after the advent of personal computers and the Internet.
3. Identify significant continuing trends in the history of the computing field.

SP2. Social context of computing [core]
Minimum core coverage time: 3 hours

Topics:
Introduction to the social implications of computing
Social implications of networked communication
Growth of, control of, and access to the Internet
Gender-related issues
International issues
Learning objectives:
1. Interpret the social context of a particular implementation.
2. Identify assumptions and values embedded in a particular design.
3. Evaluate a particular implementation through the use of empirical data.
4. Describe positive and negative ways in which computing alters the modes of interaction between people.
5. Explain why computing/network access is restricted in some countries.

SP3. Methods and tools of analysis [core]
Minimum core coverage time: 2 hours
Topics:
Making and evaluating ethical arguments
Identifying and evaluating ethical choices
Understanding the social context of design
Identifying assumptions and values
Learning objectives:
1. Analyze an argument to identify premises and conclusion.
2. Illustrate the use of example, analogy, and counter-analogy in ethical argument.
3. Detect use of basic logical fallacies in an argument.
4. Identify stakeholders in an issue and our obligations to them.
5. Articulate the ethical tradeoffs in a technical decision.

SP4. Professional and ethical responsibilities [core]
Minimum core coverage time: 3 hours
Topics:
Community values and the laws by which we live
The nature of professionalism
Various forms of professional credentialing and the advantages and disadvantages
The role of the professional in public policy
Maintaining awareness of consequences
Ethical dissent and whistle-blowing
Codes of ethics, conduct, and practice (IEEE, ACM, SE, AITP, and so forth)
Dealing with harassment and discrimination
"Acceptable use" policies for computing in the workplace
Learning objectives:
1. Identify progressive stages in a whistle-blowing incident.
2. Specify the strengths and weaknesses of relevant professional codes as expressions of professionalism and guides to decision-making.
3. Identify ethical issues that arise in software development and determine how to address them technically and ethically.

4. Develop a computer use policy with enforcement measures.
5. Analyze a global computing issue, observing the role of professionals and government officials in managing the problem.
6. Evaluate the professional codes of ethics from the ACM, the IEEE Computer Society, and other organizations.

SP5. Risks and liabilities of computer-based systems [core]
Minimum core coverage time: 2 hours
Topics:
Historical examples of software risks (such as the Therac-25 case)
Implications of software complexity
Risk assessment and management
Learning objectives:
1. Explain the limitations of testing as a means to ensure correctness.
2. Describe the differences between correctness, reliability, and safety.
3. Discuss the potential for hidden problems in reuse of existing components.
4. Describe current approaches to managing risk, and characterize the strengths and shortcomings of each.

SP6. Intellectual property [core]
Minimum core coverage time: 3 hours
Topics:
Foundations of intellectual property
Copyrights, patents, and trade secrets
Software piracy
Software patents
Transnational issues concerning intellectual property
Learning objectives:
1. Distinguish among patent, copyright, and trade secret protection.
2. Discuss the legal background of copyright in national and international law.
3. Explain how patent and copyright laws may vary internationally.
4. Outline the historical development of software patents.
5. Discuss the consequences of software piracy on software developers and the role of relevant enforcement organizations.

SP7. Privacy and civil liberties [core]
Minimum core coverage time: 2 hours
Topics:
Ethical and legal basis for privacy protection
Privacy implications of massive database systems
Technological strategies for privacy protection
Freedom of expression in cyberspace
International and intercultural implications
Learning objectives:
1. Summarize the legal bases for the right to privacy and freedom of expression in one's own nation and how those concepts vary from country to country.
2. Describe current computer-based threats to privacy.

3. Explain how the Internet may change the historical balance in protecting freedom of expression.
4. Explain both the disadvantages and advantages of free expression in cyberspace.
5. Describe trends in privacy protection as exemplified in technology.

SP8. Computer crime [elective]
Topics:
History and examples of computer crime
"Cracking" ("hacking") and its effects
Viruses, worms, and Trojan horses
Crime prevention strategies
Learning objectives:
1. Outline the technical basis of viruses and denial-of-service attacks.
2. Enumerate techniques to combat "cracker" attacks.
3. Discuss several different "cracker" approaches and motivations.
4. Identify the professional's role in security and the tradeoffs involved.

SP9. Economic issues in computing [elective]
Topics:
Monopolies and their economic implications
Effect of skilled labor supply and demand on the quality of computing products
Pricing strategies in the computing domain
Differences in access to computing resources and the possible effects thereof
Learning objectives:
1. Summarize the rationale for antimonopoly efforts.
2. Describe several ways in which the information technology industry is affected by shortages in the labor supply.
3. Suggest and defend ways to address limitations on access to computing.
4. Outline the evolution of pricing strategies for computing goods and services.

SP10. Philosophical frameworks [elective]
Topics:
Philosophical frameworks, particularly utilitarianism and deontological theories
Problems of ethical relativism
Scientific ethics in historical perspective
Differences in scientific and philosophical approaches
Learning objectives:
1. Summarize the basic concepts of relativism, utilitarianism, and deontological theories.
2. Recognize the distinction between ethical theory and professional ethics.
3. Identify the weaknesses of the "hired agent" approach, strict legalism, naïve egoism, and naïve relativism as ethical frameworks.

Discrete Structures (DS)

DS1. Functions, relations, and sets [core]
DS2. Basic logic [core]
DS3. Proof techniques [core]

DS4. Basics of counting [core]
DS5. Graphs and trees [core]
DS6. Discrete probability [core]

Discrete structures are foundational materials for software engineering. By *foundational* we mean that relatively few computer scientists will be working primarily on discrete structures, but that many other areas of computer science require the ability to work with concepts from discrete structures. Discrete structures include important materials from such areas as set theory, logic, graph theory, and combinatorics.

The material in discrete structures is pervasive in the areas of data structures and algorithms but appears elsewhere in computer science as well. For example, an ability to create and understand a formal proof is essential in formal specification, in verification, and in cryptography. Graph theory concepts are used in networks, operating systems, and compilers. Set theory concepts are used in software engineering and in databases.

As the field of computer science matures, more and more sophisticated analysis techniques are being brought to bear on practical problems. To understand the computational techniques of the future, today's students will need a strong background in discrete structures.

Finally, we note that while areas often have somewhat fuzzy boundaries, this is especially true for discrete structures. We have gathered together here a body of material of a mathematical nature that computer science education must include, and that computer science educators know well enough to specify in great detail. However, the decision about where to draw the line between this area and the Algorithms and Complexity area (AL) on the one hand, and topics left only as supporting mathematics on the other hand, was inevitably somewhat arbitrary. We remind readers that there are vital topics from those two areas that some schools will include in courses with titles like discrete structures.

DS1. Functions, relations, and sets [core]
Minimum core coverage time: 6 hours
Topics:
Functions (surjections, injections, inverses, composition)
Relations (reflexivity, symmetry, transitivity, equivalence relations)
Sets (Venn diagrams, complements, Cartesian products, power sets)
Pigeonhole principle
Cardinality and countability
Learning objectives:
1. Explain with examples the basic terminology of functions, relations, and sets.
2. Perform the operations associated with sets, functions, and relations.
3. Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
4. Demonstrate basic counting principles, including uses of diagonalization and the pigeonhole principle.

DS2. Basic logic [core]
Minimum core coverage time: 10 hours
Topics:
Propositional logic
Logical connectives
Truth tables
Normal forms (conjunctive and disjunctive)
Validity
Predicate logic
Universal and existential quantification
Modus ponens and modus tollens
Limitations of predicate logic
Learning objectives:
1. Apply formal methods of symbolic propositional and predicate logic.
2. Describe how formal tools of symbolic logic are used to model algorithms and reallife situations.
3. Use formal logic proofs and logical reasoning to solve problems such as puzzles.
4. Describe the importance and limitations of predicate logic.

DS3. Proof techniques [core]
Minimum core coverage time: 12 hours
Topics:
Notions of implication, converse, inverse, contrapositive, negation, and contradiction
The structure of formal proofs
Direct proofs
Proof by counterexample
Proof by contraposition
Proof by contradiction
Mathematical induction
Strong induction
Recursive mathematical definitions
Well orderings
Learning objectives:
1. Outline the basic structure of and give examples of each proof technique described in this unit.
2. Discuss which type of proof is best for a given problem.
3. Relate the ideas of mathematical induction to recursion and recursively defined structures.
4. Identify the difference between mathematical and strong induction and give examples of the appropriate use of each.

DS4. Basics of counting [core]
Minimum core coverage time: 5 hours
Topics:
Counting arguments
- Sum and product rule

- Inclusion-exclusion principle
- Arithmetic and geometric progressions
- Fibonacci numbers

The pigeonhole principle

Permutations and combinations
- Basic definitions
- Pascal's identity
- The binomial theorem

Solving recurrence relations
- Common examples
- The Master theorem

Learning objectives:

1. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
2. State the definition of the Master theorem.
3. Solve a variety of basic recurrence equations.
4. Analyze a problem to create relevant recurrence equations or to identify important counting questions.

DS5. Graphs and trees [core]

Minimum core coverage time: 4 hours

Topics:

Trees

Undirected graphs

Directed graphs

Spanning trees

Traversal strategies

Learning objectives:

1. Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each.
2. Demonstrate different traversal methods for trees and graphs.
3. Model problems in computer science using graphs and trees.
4. Relate graphs and trees to data structures, algorithms, and counting.

DS6. Discrete probability [core]

Minimum core coverage time: 6 hours

Topics:

Finite probability space, probability measure, events

Conditional probability, independence, Bayes' theorem

Integer random variables, expectation

Learning objectives:

1. Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance.
2. Differentiate between dependent and independent events.
3. Apply the binomial theorem to independent events and Bayes theorem to dependent events.

4. Apply the tools of probability to solve problems such as the Monte Carlo method, the average case analysis of algorithms, and hashing.

Algorithms and Complexity (AL)

AL1. Basic algorithmic analysis [core]
AL2. Algorithmic strategies [core]
AL3. Fundamental computing algorithms [core]
AL4. Distributed algorithms [core]
AL5. Basic computability [core]
AL6. The complexity classes P and NP [elective]
AL7. Automata theory [elective]
AL8. Advanced algorithmic analysis [elective]
AL9. Cryptographic algorithms [elective]
AL10. Geometric algorithms [elective]
AL11. Parallel algorithms [elective]

Algorithms are fundamental to computer science and software engineering. The real-world performance of any software system depends on only two things: (1) the algorithms chosen and (2) the suitability and efficiency of the various layers of implementation. Good algorithm design is therefore crucial for the performance of all software systems. Moreover, the study of algorithms provides insight into the intrinsic nature of the problem as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or any other implementation aspect.

An important part of computing is the ability to select algorithms appropriate to particular purposes and to apply them, recognizing the possibility that no suitable algorithm may exist. This facility relies on understanding the range of algorithms that address an important set of well-defined problems, recognizing their strengths and weaknesses, and their suitability in particular contexts. Efficiency is a pervasive theme throughout this area.

AL1. Basic algorithmic analysis [core]
Minimum core coverage time: 4 hours
Topics:
Asymptotic analysis of upper and average complexity bounds
Identifying differences among best, average, and worst case behaviors
Big O, little o, omega, and theta notation
Standard complexity classes
Empirical measurements of performance
Time and space tradeoffs in algorithms
Using recurrence relations to analyze recursive algorithms
Learning objectives:
1. Explain the use of big O, omega, and theta notation to describe the amount of work done by an algorithm.
2. Use big O, omega, and theta notation to give asymptotic upper, lower, and tight bounds

on time and space complexity of algorithms.
3. Determine the time and space complexity of simple algorithms.
4. Deduce recurrence relations that describe the time complexity of recursively defined algorithms.
5. Solve elementary recurrence relations.

AL2. Algorithmic strategies [core]
Minimum core coverage time: 6 hours
Topics:
Brute-force algorithms
Greedy algorithms
Divide-and-conquer
Backtracking
Branch-and-bound
Heuristics
Pattern matching and string/text algorithms
Numerical approximation algorithms
Learning objectives:
1. Describe the shortcoming of brute-force algorithms.
2. For each of several kinds of algorithm (brute force, greedy, divide-and-conquer, backtracking, branch-and-bound, and heuristic), identify an example of everyday human behavior that exemplifies the basic concept.
3. Implement a greedy algorithm to solve an appropriate problem.
4. Implement a divide-and-conquer algorithm to solve an appropriate problem.
5. Use backtracking to solve a problem such as navigating a maze.
6. Describe various heuristic problem-solving methods.
7. Use pattern matching to analyze substrings.
8. Use numerical approximation to solve mathematical problems, such as finding the roots of a polynomial.

AL3. Fundamental computing algorithms [core]
Minimum core coverage time: 12 hours
Topics:
Simple numerical algorithms
Sequential and binary search algorithms
Quadratic sorting algorithms (selection, insertion)
O(N log N) sorting algorithms (Quicksort, heapsort, mergesort)
Hash tables, including collision-avoidance strategies
Binary search trees
Representations of graphs (adjacency list, adjacency matrix)
Depth- and breadth-first traversals
Shortest-path algorithms (Dijkstra's and Floyd's algorithms)
Transitive closure (Floyd's algorithm)
Minimum spanning tree (Prim's and Kruskal's algorithms)
Topological sort
Learning objectives:
1. Implement the most common quadratic and *O*(N log N) sorting algorithms.

2. Design and implement an appropriate hashing function for an application.
3. Design and implement a collision-resolution algorithm for a hash table.
4. Discuss the computational efficiency of the principal algorithms for sorting, searching, and hashing.
5. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data.
6. Solve problems using the fundamental graph algorithms, including depth-first and breadth-first search, single-source and all-pairs shortest paths, transitive closure, topological sort, and at least one minimum spanning tree algorithm.
7. Demonstrate the following capabilities: to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in programming context.

AL4. Distributed algorithms [core]
Minimum core coverage time: 3 hours
Topics:
Consensus and election
Termination detection
Fault tolerance
Stabilization
Learning objectives:
1. Explain the distributed paradigm.
2. Explain one simple distributed algorithm.
3. Determine when to use consensus or election algorithms.
4. Distinguish between logical and physical clocks.
5. Describe the relative ordering of events in a distributed algorithm.

AL5. Basic computability [core]
Minimum core coverage time: 6 hours
Topics:
Finite-state machines
Context-free grammars
Tractable and intractable problems
Uncomputable functions
The halting problem
Implications of uncomputability
Learning objectives:
1. Discuss the concept of finite state machines.
2. Explain context-free grammars.
3. Design a deterministic finite-state machine to accept a specified language.
4. Explain how some problems have no algorithmic solution.
5. Provide examples that illustrate the concept of uncomputability.

AL6. The complexity classes P and NP [elective]
Topics:
Definition of the classes P and NP

NP-completeness (Cook's theorem)
Standard NP-complete problems
Reduction techniques
Learning objectives:
1. Define the classes P and NP.
2. Explain the significance of NP-completeness.
3. Prove that a problem is NP-complete by reducing a classic known NP-complete problem to it.

AL7. Automata theory [elective]
Topics:
Deterministic finite automata (DFAs)
Nondeterministic finite automata (NFAs)
Equivalence of DFAs and NFAs
Regular expressions
The pumping lemma for regular expressions
Push-down automata (PDAs)
Relationship of PDAs and context-free grammars
Properties of context-free grammars
Turing machines
Nondeterministic Turing machines
Sets and languages
Chomsky hierarchy
The Church-Turing thesis
Learning objectives:
1. Determine a language's location in the Chomsky hierarchy (regular sets, context-free, context-sensitive, and recursively enumerable languages).
2. Prove that a language is in a specified class and that it is not in the next lower class.
3. Convert among equivalently powerful notations for a language, including among DFAs, NFAs, and regular expressions, and between PDAs and CFGs.
4. Explain at least one algorithm for both top-down and bottom-up parsing.
5. Explain the Church-Turing thesis and its significance.

AL8. Advanced algorithmic analysis [elective]
Topics:
Amortized analysis
Online and offline algorithms
Randomized algorithms
Dynamic programming
Combinatorial optimization

Learning objectives:
1. Use the potential method to provide an amortized analysis of previously unseen data structure, given the potential function.
2. Explain why competitive analysis is an appropriate measure for online algorithms.
3. Explain the use of randomization in the design of an algorithm for a problem where a

deterministic algorithm is unknown or much more difficult.
4. Design and implement a dynamic programming solution to a problem.

AL9. Cryptographic algorithms [elective]
Topics:
Historical overview of cryptography
Private-key cryptography and the key-exchange problem
Public-key cryptography
Digital signatures
Security protocols
Applications (zero-knowledge proofs, authentication, and so on)
Learning objectives:
1. Describe efficient basic number-theoretic algorithms, including greatest common divisor, multiplicative inverse mod n, and raising to powers mod n.
2. Describe at least one public-key cryptosystem, including a necessary complexity-theoretic assumption for its security.
3. Create simple extensions of cryptographic protocols, using known protocols and cryptographic primitives.

AL10. Geometric algorithms [elective]
Topics:
Line segments: properties, intersections
Convex hull finding algorithms
Learning objectives:
1. Describe and give time analysis of at least two algorithms for finding a convex hull.
2. Justify the Omega(N log N) lower bound on finding the convex hull.
3. Describe at least one additional efficient computational geometry algorithm, such as finding the closest pair of points, convex layers, or maximal layers.

AL11. Parallel algorithms [elective]
Topics:
PRAM model
Exclusive versus concurrent reads and writes
Pointer jumping
Brent's theorem and work efficiency
Learning objectives:
1. Describe implementation of linked lists on a PRAM.
2. Use parallel-prefix operation to perform simple computations efficiently in parallel.
3. Explain Brent's theorem and its relevance.

Architecture and Organization (AR)

AR1. Digital logic and digital systems [core]
AR2. Machine level representation of data [core]
AR3. Assembly level machine organization [core]
AR4. Memory system organization and architecture [core]
AR5. Interfacing and communication [core]

AR6. Functional organization [core]
AR7. Multiprocessing and alternative architectures [core]
AR8. Performance enhancements [elective]
AR9. Architecture for networks and distributed systems [elective]
The computer lies at the heart of computing. Without it most of the computing disciplines today would be a branch of theoretical mathematics. To be a professional in any field of computing today, one should not regard the computer as just a black box that executes programs by magic. All students of computing should acquire some understanding and appreciation of a computer system's functional components, their characteristics, their performance, and their interactions. There are practical implications as well. Students need to understand computer architecture in order to structure a program so that it runs more efficiently on a real machine. In selecting a system to use, they should to able to understand the tradeoff among various components, such as CPU clock speed vs. memory size.

The learning outcomes specified for these topics correspond primarily to the core and are intended to support programs that elect to require only the minimum 36 hours of computer architecture of their students. For programs that want to teach more than the minimum, the same topics (AR1-AR7) can be treated at a more advanced level by implementing a two-course sequence. For programs that want to cover the elective topics, those topics can be introduced within a two-course sequence and/or be treated in a more comprehensive way in a third course.

AR1. Digital logic and digital systems [core]
Minimum core coverage time: 6 hours
Topics:
Overview and history of computer architecture
Fundamental building blocks (logic gates, flip-flops, counters, registers, PLA)
Logic expressions, minimization, sum of product forms
Register transfer notation
Physical considerations (gate delays, fan-in, fan-out)
Learning objectives:
1. Describe the progression of computer architecture from vacuum tubes to VLSI.
2. Demonstrate an understanding of the basic building blocks and their role in the historical development of computer architecture.
3. Use mathematical expressions to describe the functions of simple combinational and sequential circuits.
4. Design a simple circuit using the fundamental building blocks.

AR2. Machine level representation of data [core]
Minimum core coverage time: 3 hours
Topics:
Bits, bytes, and words
Numeric data representation and number bases
Fixed- and floating-point systems
Signed and twos-complement representations
Representation of nonnumeric data (character codes, graphical data)

Representation of records and arrays
Learning objectives:
1. Explain the reasons for using different formats to represent numerical data.
2. Explain how negative integers are stored in sign-magnitude and twos-complement representation.
3. Convert numerical data from one format to another.
4. Discuss how fixed-length number representations affect accuracy and precision.
5. Describe the internal representation of nonnumeric data.
6. Describe the internal representation of characters, strings, records, and arrays.

AR3. Assembly level machine organization [core]
Minimum core coverage time: 9 hours
Topics:
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
Learning objectives:
1. Explain the organization of the classical von Neumann machine and its major functional units.
2. Explain how an instruction is executed in a classical von Neumann machine.
3. Summarize how instructions are represented at both the machine level and in the context of a symbolic assembler.
4. Explain different instruction formats, such as addresses per instruction and variable length vs. fixed length formats.
5. Write simple assembly language program segments.
6. Demonstrate how fundamental high-level programming constructs are implemented at the machine-language level.
7. Explain how subroutine calls are handled at the assembly level.
8. Explain the basic concepts of interrupts and I/O operations.

AR4. Memory system organization and architecture [core]
Minimum core coverage time: 5 hours
Topics:
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
Learning objectives:

1. Identify the main types of memory technology.
2. Explain the effect of memory latency on running time.
3. Explain the use of memory hierarchy to reduce the effective memory latency.
4. Describe the principles of memory management.
5. Describe the role of cache and virtual memory.
6. Explain the workings of a system with virtual memory management.

AR5. Interfacing and communication [core]
Minimum core coverage time: 3 hours
Topics:
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
Learning objectives:
1. Explain how interrupts are used to implement I/O control and data transfers.
2. Identify various types of buses in a computer system.
3. Describe data access from a magnetic disk drive.
4. Compare the common network configurations.
5. Identify interfaces needed for multimedia support.
6. Describe the advantages and limitations of RAID architectures.

AR6. Functional organization [core]
Minimum core coverage time: 7 hours
Topics:
Implementation of simple data paths
Control unit: hardwired realization vs. micro-programmed realization
Instruction pipelining
Introduction to instruction-level parallelism (ILP)
Learning objectives:
1. Compare alternative implementation of data paths.
2. Discuss the concept of control points and the generation of control signals using hardwired or micro-programmed implementations.
3. Explain basic instruction level parallelism using pipelining and the major hazards that may occur.

AR7. Multiprocessing and alternative architectures [core]
Minimum core coverage time: 3 hours
Topics:
Introduction to SIMD, MIMD, VLIW, EPIC
Systolic architecture
Interconnection networks (hypercube, shuffle-exchange, mesh, crossbar)
Shared memory systems
Cache coherence

Memory models and memory consistency
Learning objectives:
1. Discuss the concept of parallel processing beyond the classical von Neumann model.
2. Describe alternative architectures such as SIMD, MIMD, and VLIW.
3. Explain the concept of interconnection networks and characterize different approaches.
4. Discuss the special concerns that multiprocessing systems present with respect to memory management and describe how these are addressed.

AR8. Performance enhancements [elective]
Topics:
Superscalar architecture
Branch prediction
Prefetching
Speculative execution
Multithreading
Scalability
Learning objectives:
1. Describe superscalar architectures and their advantages.
2. Explain the concept of branch prediction and its utility.
3. Characterize the costs and benefits of prefetching.
4. Explain speculative execution and identify the conditions that justify it.
5. Discuss the performance advantages that multithreading can offer in an architecture along with the factors that make it difficult to derive maximum benefits from this approach.
6. Describe the relevance of scalability to performance.

AR9. Architecture for networks and distributed systems [elective]
Topics:
Introduction to LANs and WANs
Layered protocol design, ISO/OSI, IEEE 802
Impact of architectural issues on distributed algorithms
Network computing
Distributed multimedia
Learning objectives:
1. Explain the basic components of network systems and distinguish between LANs and WANs.
2. Discuss the architectural issues involved in the design of a layered network protocol.
3. Explain how architectures differ in network and distributed systems.
4. Discuss architectural issues related to network computing and distributed multimedia.