Appendix A

The Computer Engineering Body of Knowledge

This appendix to the Computing Curricula - Computer Engineering report defines the knowledge domain that is likely to be taught in an undergraduate curriculum in computer engineering. The underlying rationale for this categorization scheme and additional details about its history, structure, and application are included in the full task force report. Because we expect the appendices to have wider circulation than the full report, the task force feels it necessary to include in each appendix a summary of the fundamental concepts that are necessary to understand the recommendations. The most important concepts are outlined in the sections that follow.

Introduction

In developing a curriculum for undergraduate study in computer engineering, one of the first steps is to identify and organize the material that would be appropriate for that level. As noted in Chapter 1, the IEEE Computer Society/ACM Computing Curriculum - Computer Engineering Task Force sought to accomplish this goal by assigning to each task force member the responsibility of defining the body of knowledge associated with one or more of the following areas:

Most Likely Will Contain Core Topics

- SPR - Social and Professional Issues
- CSE - Computer Systems Engineering
- CAO - Computer Architecture and Organization
- SWE - Software Engineering
- ESY - Embedded Systems
- OPS - Operating Systems
- CSY - Circuits and Systems
- NWK - Networks
- ELE - Electronics
- DIG - Digital Logic
- PRF - Programming Fundamentals
- ALG - Algorithms and Complexity
- DSC - Discrete Structures

Likely To Be More Elective Based

- DSP - Digital Signal Processing
- VLS - VLSI/ASIC Design
DGA - Design Automation
ACP - Alternative Computing Paradigms
TFT - Testing and Fault Tolerance

For each subdiscipline, a second task force member was assigned to critique and revise the initial draft. After each revision, the draft was sent to the entire task force for comment. At the completion of this process, the entire task force met as a group to review the draft body of knowledge, with follow-up modifications made as appropriate. Finally, the body of knowledge was made available on the task force web site for review by the entire computer engineering community.

Structure of the body of knowledge

The CPE body of knowledge is organized hierarchically into three levels, as was done in the Computer Science volume. The highest level of the hierarchy is the area, which represents a particular disciplinary subfield. Each area is identified by a three-letter abbreviation, such as DLG for Digital Logic or CAO for Computer Architecture and Organization. The areas are broken down into smaller divisions called units, which represent individual thematic modules within an area. Each unit is identified by adding a numeric suffix to the area name; as an example, CAO3 is a unit on memory system organization and architecture. Each unit is further subdivided into a set of topics, which are the lowest level of the hierarchy.

Core and elective units

As discussed in Chapter 4, one of our goals in proposing curricular recommendations is to keep the required component of the body of knowledge as small as possible. To implement this principle, the IEEE-CS/ACM Computing Curriculum - Computer Engineering Task Force has defined a minimal core comprising those units for which there is broad consensus that the corresponding material is essential to anyone obtaining an undergraduate degree in computer engineering. Units that are taught as part of an undergraduate program, but which fall outside the core, are considered to be elective.

In discussing the Task Force recommendations during their development, we have found that it helps to emphasized the following points:

- **The core is not a complete curriculum.** Because the core is defined as minimal, it does not, by itself, constitute a complete undergraduate curriculum. Every undergraduate program must include additional elective units from the body of knowledge, although the CC2001 report does not define what those units will be.

- **Core units are not necessarily limited to a set of introductory courses taken early in the undergraduate curriculum.** Although many of the unit defined as core are indeed introductory, there are also some core units that clearly must be covered only after
students have developed significant background in the field. For example, the task force believes that all students must develop a significant application at some point during their undergraduate program. The material that is essential to successful management of projects at this scale is therefore part of the core, since it is required of all students. At the same time, the project course experience is very likely to come toward the end of a student's undergraduate program. Similarly, introductory courses may include elective units alongside the coverage of core material. The designation core simply means required and says nothing about the level of the course in which it appears.

Assessing the time required to cover a unit

To give readers a sense of the time required to cover a particular unit, the CC2001 report must define a metric that establishes a standard of measurement. Choosing such a metric has proven difficult, because no standard measure is recognized throughout the world. For consistency with the Computer Science Volume and earlier curriculum reports, the task force has chosen to express time in hours, corresponding to the in-class time required to present that material in a traditional lecture-oriented format. To dispel any potential confusion, however, it is important to underscore the following observations about the use of lecture hours as a measure.

• *The task force does not seek to endorse the lecture format.* Even though we have used a metric with its roots in a classical lecture-oriented form, the task force believes there are other styles - particularly given recent improvements in educational technology - that can be at least as effective. For some of these styles, the notion of hours may be difficult to apply. Even so, the time specifications should at least serve as a comparative measure, in the sense that a 5-hour unit will presumably take roughly five times as much time to cover as a 1-hour unit, independent of the teaching style.

• *The hours specified do not include time spent outside of a class.* The time assigned to a unit does not include the instructor's reparation time or the time students spend outside of class. As a general guideline, the amount of out-of-class work is approximately three times the in-class time. Thus, a unit that is listed as requiring 3 hours will typically entail a total of 12 hours (3 in class and 9 outside).

• *The hours listed for a unit represent a minimum level of coverage.* The time measurements we have assigned for each unit should be interpreted as the minimum amount of time necessary to enable a student to perform the learning objectives for that unit. It is always appropriate to spend more time on a unit than the mandated minimum.

Summary of the CPE body of knowledge

A summary of the body of knowledge - showing the areas, units, which units are core, and the minimum time required for each - appears as Figure A-1. The details of each section of the body of knowledge follow as separate sections.
Figure A-1. The Body of Knowledge

<table>
<thead>
<tr>
<th>SPR. Social and Professional Issues (16 core hours)</th>
<th>CSE. Computer Systems Engineering (12 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPR0. History and overview of social and professional issues (1)</td>
<td>CSE0. History and overview of computer systems design (1)</td>
</tr>
<tr>
<td>SPR1. Social context of computing (3)</td>
<td>CSE1. Overview of systems engineering (2)</td>
</tr>
<tr>
<td>SPR2. Methods and tools of analysis (2)</td>
<td>CSE2. Theoretical considerations (elective)</td>
</tr>
<tr>
<td>SPR3. Professional and ethical responsibilities (3)</td>
<td>CSE3. Life cycle (2)</td>
</tr>
<tr>
<td>SPR4. Risks and liabilities of computer-based systems (2)</td>
<td>CSE4. Requirements analysis and elicitation (2)</td>
</tr>
<tr>
<td>SPR5. Intellectual property (3)</td>
<td>CSE5. Specification (2)</td>
</tr>
<tr>
<td>SPR6. Privacy and civil liberties (2)</td>
<td>CSE6. Architectural design (3)</td>
</tr>
<tr>
<td>SPR7. Computer crime (elective)</td>
<td>CSE7. Implementation (elective)</td>
</tr>
<tr>
<td>SPR8. Economic issues in computing (elective)</td>
<td>CSE8. Testing (elective)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CAO. Computer Architecture and Organization (85 core hours)</th>
<th>SWE. Software Engineering (11 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAO0. History and overview of computer architecture (1)</td>
<td>SWE0. History and overview of software engineering (1)</td>
</tr>
<tr>
<td>CAO1. Fundamentals of computer architecture (18)</td>
<td>SWE1. Software processes (2)</td>
</tr>
<tr>
<td>CAO2. Computer arithmetic (3)</td>
<td>SWE2. Software requirements and specifications (2)</td>
</tr>
<tr>
<td>CAO3. Memory system organization and architecture (9)</td>
<td>SWE3. Software design (2)</td>
</tr>
<tr>
<td>CAO4. Interfacing and communication (12)</td>
<td>SWE4. Software testing and validation (2)</td>
</tr>
<tr>
<td>CAO5. Interface subsystems (12)</td>
<td>SWE5. Software evolution (elective)</td>
</tr>
<tr>
<td>CAO6. Processor systems design (12)</td>
<td>SWE6. Software tools and environments (2)</td>
</tr>
<tr>
<td>CAO7. Organization of the CPU (12)</td>
<td>SWE7. Software project management (elective)</td>
</tr>
<tr>
<td>CAO8. Performance (3)</td>
<td></td>
</tr>
<tr>
<td>CAO9. Performance enhancements (elective)</td>
<td></td>
</tr>
<tr>
<td>CAO10. Multiprocessing (3)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ESY. Embedded Systems (17 core hours)</th>
<th>OPS. Operating Systems (26 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESY0. History and overview of embedded systems (1)</td>
<td>OPS0. History and overview of operating systems (1)</td>
</tr>
<tr>
<td>ESY1. Fundamentals of embedded systems (4)</td>
<td>OPS1. Operating system function and design (2)</td>
</tr>
<tr>
<td>ESY2. Language issues (3)</td>
<td>OPS2. Operating system principles (3)</td>
</tr>
<tr>
<td>ESY3. Hardware considerations (elective)</td>
<td>OPS3. Concurrency (6)</td>
</tr>
<tr>
<td>ESY4. Mapping between languages and hardware (3)</td>
<td>OPS4. Scheduling and dispatch (3)</td>
</tr>
<tr>
<td>ESY5. Real-time Operating Systems (elective)</td>
<td>OPS5. Memory management (3)</td>
</tr>
<tr>
<td>ESY6. Classification of embedded systems (6)</td>
<td>OPS6. Device management (4)</td>
</tr>
<tr>
<td>ESY7. Software engineering considerations (elective)</td>
<td>OPS7. Security and protection (elective)</td>
</tr>
<tr>
<td>ESY8. Particular techniques and applications (elective)</td>
<td>OPS8. File systems (elective)</td>
</tr>
<tr>
<td>ESY10. High integrity software systems (elective)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CSY. Circuits and Systems (73 core hours)</th>
<th>NWK. Networks (24 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSY0. History and overview of systems and circuits (1)</td>
<td>NWK0. History and overview of networks (1)</td>
</tr>
<tr>
<td>CSY1. Fundamental Electrical Quantities (3)</td>
<td>NWK1. Communications Network Architecture (4)</td>
</tr>
<tr>
<td>CSY2. Resistive Circuits and Networks(20)</td>
<td>NWK2. Communications Network Protocols (6)</td>
</tr>
<tr>
<td>CSY3. Reactive Circuits and Networks (14)</td>
<td>NWK3. Local and Wide Area Networks (10)</td>
</tr>
<tr>
<td>CSY4. Frequency Response (10)</td>
<td>NWK4. The web as an example of client-server computing (3)</td>
</tr>
<tr>
<td>CSY5. Sinusoidal Analysis (7)</td>
<td>NWK5. Data Security and Integrity (elective)</td>
</tr>
<tr>
<td>CSY6. Convolution (4)</td>
<td></td>
</tr>
</tbody>
</table>

| OPS0. History and overview of operating systems (1) | OPS1. Operating system function and design (2) |
| OPS2. Operating system principles (3) | OPS3. Concurrency (6) |
| OPS4. Scheduling and dispatch (3) | OPS5. Memory management (3) |
| OPS6. Device management (4) | OPS7. Security and protection (elective) |
### CSY. Discrete Time Signals (8)
- CSY7. Discrete Time Signals (8)
- CSY8. Fourier Analysis (6)
- CSY9. Filters (elective)
- CSY10. Laplace Transforms (elective)
- CSY11. z – Transforms (elective)
- CSY12. Digital Filters (elective)

### NWK. Performance Evaluation (elective)
- NWK6. Performance Evaluation (elective)
- NWK7. Data Communications (elective)
- NWK8. Wireless and mobile computing (elective)

### ELE. Electronics (45 core hours)
- ELE0. History and overview of electronics (1)
- ELE1. Electronic properties of materials (4)
- ELE2. Diodes and diode circuits (5)
- ELE3. MOS transistors and biasing (3)
- ELE4. MOS logic families (9)
- ELE5. Bipolar transistors and logic families (5)
- ELE6. Design parameters and issues (5)
- ELE7. Storage elements (3)
- ELE8. Interfacing logic families and standard buses (3)
- ELE9. Operational amplifiers (6)
- ELE10. Data conversion circuits (elective)
- ELE11. SPICE circuit simulation (3)
- ELE12. Electronic voltage and current sources (elective)
- ELE13. Linear amplification and biasing (elective)
- ELE14. Single-transistor amplifiers (elective)
- ELE15. Multistage transistor amplifiers (elective)
- ELE16. Power circuits (elective)
- ELE17. Feedback in electronics (elective)
- ELE18. Active filters (elective)
- ELE19. Integrated circuit building blocks (elective)
- ELE20. Circuits for wireless applications (elective)

### DIG. Digital Logic (50 core hours)
- DIG0. History and overview of digital logic (1)
- DIG1. Switching theory (6)
- DIG2. Combinational logic circuits (4)
- DIG3. Modular design of combinational circuits (6)
- DIG4. Memory elements (3)
- DIG5. Sequential logic circuits (12)
- DIG6. Register Transfer Logic (6)
- DIG7. Digital Systems Design (12)

### PRF. Programming Fundamentals (44 core hours)
- PRF0. History and overview of programming fundamentals (1)
- PRF1. Fundamental programming constructs (9)
- PRF2. Algorithms and problem-solving (6)
- PRF3. Fundamental data structures (14)
- PRF4. Programming Paradigms (8)
- PRF5. Recursion (6)
- PRF6. Object-oriented programming (elective)
- PRF7. Event-driven and concurrent programming (elective)
- PRF8. Using APIs (elective)

### VLS. VLSI and ASIC Design (0 core hours)
- VLS1. MOS Transistor Fundamentals
- VLS2. Processing and Layout
- VLS3. Function of the Basic Inverter Structure
- VLS4. Circuit Characterization and Performance
- VLS5. Combinational Logic Circuits
- VLS6. Sequential Logic Circuits
- VLS7. Alternative Circuit Structures/Low Power Design
- VLS8. Semiconductor Memories and Array Structures
- VLS9. Chip Input/Output Circuits
- VLS10. Semi custom Design Technologies
- VLS11. ASIC Design Methodology

### ALG. Algorithms and Complexity (34 core hours)
- ALG0. History and overview of algorithms and complexity (1)
- ALG1. Basic algorithmic analysis (4)
- ALG2. Algorithmic strategies (8)
- ALG3. Fundamental computing algorithms (12)
- ALG4. Distributed algorithms (3)
- ALG5. Basic computability theory (6)
- ALG6. The complexity classes P and NP (elective)

### DSC. Discrete Structures (36 core hours)
- DSC0. History and overview of discrete structures (1)
- DSC1. Functions, relations, and sets (6)
- DSC2. Basic logic (10)
- DSC3. Proof techniques (6)
- DSC4. Basics of counting (3)
- DSC5. Graphs and trees (4)
- DSC6. Discrete probability (6)
- DSC7. Recursion (elective)

### DSC. Discrete Structures (36 core hours)
- DSC0. History and overview of discrete structures (1)
- DSC1. Functions, relations, and sets (6)
- DSC2. Basic logic (10)
- DSC3. Proof techniques (6)
- DSC4. Basics of counting (3)
- DSC5. Graphs and trees (4)
- DSC6. Discrete probability (6)
- DSC7. Recursion (elective)
<table>
<thead>
<tr>
<th>ACP. Alternative Computing Paradigms (0 core hours)</th>
<th>TFT. Testing and Fault Tolerance (0 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACP1. Overview/History</td>
<td>TFT1 – Faults and Fault Models in Digital Circuits</td>
</tr>
<tr>
<td>ACP2. Paradigms</td>
<td>TFT2 - Test generation methods</td>
</tr>
<tr>
<td>ACP3. Architectures</td>
<td>TFT3 - Design for testability</td>
</tr>
<tr>
<td>ACP4. Operating systems issues</td>
<td>TFT4 - Testing non-stuck-at faults</td>
</tr>
<tr>
<td>ACP5. Software issues</td>
<td>TFT5 - System-level test and diagnosis</td>
</tr>
<tr>
<td>ACP6. Algorithms</td>
<td>TFT6 - Reliability and fault tolerance definitions</td>
</tr>
<tr>
<td>ACP7. Applications</td>
<td>TFT7 - Error detecting and correcting codes</td>
</tr>
<tr>
<td></td>
<td>TFT8 – Fault Tolerant System Design</td>
</tr>
<tr>
<td></td>
<td>TFT9 – Software approaches and software fault tolerance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSP. Digital Signal Processing (0 core hours)</th>
<th>DSV. Digital System Verification (0 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP1. Overview of Digital Audio and its application</td>
<td>DSV0. History and Overview Including Pentium Bugs and other Horror Stories, Verification vs. Validation. Relationship of Good Design Practice to Verification</td>
</tr>
<tr>
<td>DSP3. Multimedia programming, data streaming</td>
<td>DSV3. Formal Verification: Model Checking</td>
</tr>
<tr>
<td>DSP4. Wave Table Generation</td>
<td>DSV4: Formal Verification: Proofs</td>
</tr>
<tr>
<td>DSP5. Convolution (SC8)</td>
<td>DSV5: Formal Verification: Equivalence Checking</td>
</tr>
<tr>
<td>DSP6. Fourier Analysis (SC10, modified)</td>
<td>DSV6: Verification by Simulation and Testbenches</td>
</tr>
<tr>
<td>DSP7. Audio Processing</td>
<td>DSV7: Verification by Assertions and Verification Languages</td>
</tr>
<tr>
<td>DSP8. Generalized Modulations and Demodulations</td>
<td>DSV8: Verification by Testing</td>
</tr>
<tr>
<td>DSP9. LaPlace Transforms (SC12 with modifications)</td>
<td>DSV9: Other Verification: Signal Integrity, Specification, Reliability, Safety, Power, Cooling, ASIC Physical Design</td>
</tr>
<tr>
<td>DSP11. Digital Filters (SC14, with modifications)</td>
<td>DSV11: Configuration Control, Bug Tracking, Regression Testing</td>
</tr>
<tr>
<td>DSP13. Simple Graphics</td>
<td></td>
</tr>
<tr>
<td>DSP14. Displaying Images</td>
<td></td>
</tr>
<tr>
<td>DSP15. Reading and Writing Image Files</td>
<td></td>
</tr>
<tr>
<td>DSP16. Edge Detection</td>
<td></td>
</tr>
<tr>
<td>DSP17. Boundary Processing</td>
<td></td>
</tr>
<tr>
<td>DSP18. Image Enhancement Techniques</td>
<td></td>
</tr>
<tr>
<td>DSP19. Achromatic and Colored Light</td>
<td></td>
</tr>
<tr>
<td>DSP20. Thresholding techniques</td>
<td></td>
</tr>
<tr>
<td>DSP21. Morphological filtering</td>
<td></td>
</tr>
<tr>
<td>DSP22. Warping</td>
<td></td>
</tr>
<tr>
<td>DSP23. The Cosine Transform</td>
<td></td>
</tr>
<tr>
<td>DSP24. The InLine MPEG CODEC</td>
<td></td>
</tr>
<tr>
<td>DSP25. The Wavelet Transform</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSI. Digital Systems Engineering: Signal Integrity (0 core hours)</th>
<th>ISA. Intelligent Systems and Automation (0 core hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSI0: History and Overview, Motivation, Importance, Horror Stories</td>
<td>ISA (to be developed)</td>
</tr>
<tr>
<td>DSI1: Signals</td>
<td></td>
</tr>
<tr>
<td>DSI2: Lossless Transmission Lines</td>
<td></td>
</tr>
<tr>
<td>DSI3: Coupled Lines</td>
<td></td>
</tr>
<tr>
<td>DSI4: Measurement</td>
<td></td>
</tr>
<tr>
<td>DSI5: Simulation</td>
<td></td>
</tr>
<tr>
<td>DSI6: Signaling</td>
<td></td>
</tr>
<tr>
<td>DSI7: Power distribution</td>
<td></td>
</tr>
<tr>
<td>DSI8: EMI/EMC(?)</td>
<td></td>
</tr>
</tbody>
</table>
Social and Professional Issues (SPR)

- SPR0. History and overview of social and professional issues [core]
- SPR1. Social context of computing [core]
- SPR2. Methods and tools of analysis [core]
- SPR3. Professional and ethical responsibilities [core]
- SPR4. Risks and liabilities of computer-based systems [core]
- SPR5. Intellectual property [core]
- SPR6. Privacy and civil liberties [core]
- SPR7. Computer crime [elective]
- SPR8. Economic issues in computing [elective]
- SPR9. Philosophical frameworks [elective]

Although technical issues are obviously central to any curriculum in computer engineering, they do not in themselves constitute a complete educational program in the field. Students must also develop an understanding of the social and professional context in which they apply their computer engineering education.

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, SPR1, SPR2, SPR3, and SPR5—do not readily conform to topics covered in other technical 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” of technical topics, it will reinforce the false notion that technological 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.

Ethics-related modules could be developed for almost any course in the curriculum. Courses in areas such as software engineering, databases, computer networks, data mining, and human computer interfaces provide obvious context for analysis of ethical issues and should arise naturally from those subjects. For example, a programming assignment 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.

Running through all of the issues in this area is the need to speak to the computer engineer’s responsibility to proactively address societal issues by both moral and technical actions. Computer engineers must be cognizant of their responsibility to the public. They must also be aware of the potential conflicts between the obligations to their employer and the obligations to the customer, user, and others affected by their work. Chapter 2 of this report provides a more in-depth discussion of professionalism, professional practices, and the societal responsibilities of computer engineers.
SPR0. History and overview of social and professional issues [core]

Suggested time: 1 hour

Topics:
- Knowledge themes include social context of computing, professional and ethical responsibilities, risks and trade-offs, intellectual property, privacy, and codes of ethics and professional conduct
- Contributors to the subject include philosophies of ethics, laws of different nations, case studies, and codes of professional practice and conduct.
- Purpose and role of professionalism in computer engineering

Learning objectives:
- Associate the themes involved with social and professional issues of computing.
- List the contributions of several pioneers in the computing field.
- Identify significant continuing trends in the history of the computing field.
- Contrast how different political societies might view ethics and professional practice.
- Describe how computer engineering could make use of social and professional issues of computing.

SPR1. Social context of computing [core]

Suggested 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:
- Interpret the social context of a particular implementation.
- Identify assumptions and values embedded in a particular design.
- Evaluate a particular implementation through the use of empirical data.
- Describe positive and negative ways in which computing alters the modes of interaction between people.
- Explain why computing/network access is restricted in some countries.

SPR2. Methods and tools of analysis [core]

Suggested 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:
- Analyze an argument to identify premises and conclusion.
- Illustrate the use of example, analogy, and counter-analogy in ethical argument.
- Detect use of basic logical fallacies in an argument.
- Identify stakeholders in an issue and our obligations to them.
- Articulate the ethical tradeoffs in a technical decision.

SPR3. Professional and ethical responsibilities [core]
Suggested 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
- The role of licensure and practice in engineering
- Contrasts of licensure in engineering but not other disciplines
- Maintaining awareness of consequences
- Ethical dissent and whistle blowing
- Codes of ethics, conduct, and practice (NSPE, IEEE, ACM, SE, AITP, and so forth)
- Dealing with harassment and discrimination
- “Acceptable use” policies for computing in the workplace

Learning objectives:
- Identify progressive stages in a whistle-blowing incident.
- Specify the strengths and weaknesses of relevant professional codes as expressions of professionalism and guides to decision-making.
- Provide arguments for and against licensure in non-engineering professions.
- Identify ethical issues that arise in software development and determine how to address them technically and ethically.
- Develop a computer use policy with enforcement measures.

SPR4. Risks and liabilities of computer-based systems [core]
Suggested time: 2 hours

Topics:
- Historical examples of software risks (such as the Therac-25 case)
- Product safety and public consumption
- Implications of software complexity
- Risk assessment and management
Learning objectives:
- Explain the limitations of testing as a means to ensure correctness.
- Recognize the importance of product safety when designing computer systems.
- Describe the differences between correctness, reliability, and safety.
- Recognize unwarranted assumptions of statistical independence of errors.
- Discuss the potential for hidden problems in reuse of existing components.

SPR5. Intellectual property [core]
Suggested time: 3 hours

Topics:
- Foundations of intellectual property
- Copyrights, patents, and trade secrets
- Software piracy
- Software patents
- Transnational issues concerning intellectual property

Learning objectives:
- Distinguish among patent, copyright, and trade secret protection.
- Discuss the legal background of copyright in national and international law.
- Explain how patent and copyright laws may vary internationally.
- Outline the historical development of software patents.

SPR6. Privacy and civil liberties [core]
Suggested 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:
- 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.
- Describe current computer-based threats to privacy.
- Explain how the Internet may change the historical balance in protecting freedom of expression.

SPR7. Computer crime [elective]
Suggested time: 3 hours
Topics:
- History and examples of computer crime
- “Cracking” ("hacking") and its effects
- Viruses, worms, and Trojan horses
- Crime prevention strategies

Learning objectives:
- Outline the technical basis of viruses and denial-of-service attacks.
- Enumerate techniques to combat “cracker” attacks.
- Discuss several different “cracker” approaches and motivations.
- Identify the professional’s role in security and the tradeoffs involved.

SPR8. Economic issues in computing [elective]
Suggested time: 6 hours

Topics:
- Costing out jobs with considerations on manufacturing, hardware, software, and engineering implications.
- Cost estimates versus actual costs in relation to total costs
- Use of engineering economics in dealing with finances
- Entrepreneurship: prospects and pitfalls
- 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:
- Describe the assessment of total job costs.
- Evaluate the risks of entering one’s own business.
- Apply engineering economic principles when considering fiscal arrangements.
- Summarize the rationale for antimonopoly efforts.
- Describe several ways in which the information technology industry is affected by shortages in the labor supply.
- Suggest and defend ways to address limitations on access to computing.

SPR9. Philosophical frameworks [elective]
Suggested time: 2 hours

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:

- Summarize the basic concepts of relativism, utilitarianism, and deontological theories.
- Recognize the distinction between ethical theory and professional ethics.
- Identify the weaknesses of the “hired agent” approach, strict legalism, naïve egoism, and naïve relativism as ethical frameworks.
Computer Systems Engineering (CSE)

CSE0. History and overview of computer systems design [core]
CSE1. Overview of systems engineering [core]
CSE 2. Theoretical considerations [elective]
CSE 3. Life cycle [core]
CSE 4. Requirements analysis and elicitation [core]
CSE 5. Specification [core]
CSE 6. Architectural design [core]
CSE 7. Implementation
CSE 8. Testing
CSE 9. Maintenance
CSE 10. Project management
CSE 11. Specialist systems
CSE 12. Hardware and software co-design

A frequent occurrence within computer engineering is the challenge to build a system containing hardware and software components possibly as part of a larger system. The development of computer systems themselves falls within this. But included also is the development of new devices such as digital camera, hand-held computers, location aware systems and so on; for many of these the idea of a human computer interface is an essential component influencing usage and effectiveness. Embedded systems developments are also clearly included; it is important to recognise the current potential of this expanding area of application here. For instance, web sites on spacecraft are all included.

In all of these cases decisions have to be made about how to design to have maximum impact and effect. Decisions have to be made about alternative approaches, trade-offs need to be addressed and decisions on all these matters need to be justified on the grounds of technical insight and judgement. Often the computer engineer will be part of a multi-disciplinary team and will have to react accordingly. This creates additional challenge.

Systems engineering is the name given to the discipline within which such matters are addressed in a carefully considered fashion. The development of new system is a highly creative and rewarding activity. Generally it is about insight, about the difficult topic of design with its many facets, including decisions about trade-offs. Furthermore it is vitally important for the economy of many countries and is a central concern of Computer Engineering.

CSE0. History and overview of computer systems design [core]

*Suggested time:* 1 hour

*Topics:*

- Knowledge themes include system-level design, hardware-software interface, direct and indirect interaction, and the human-computer interface
- Contributors to the subject include many individuals from industry and government
- Importance of computer system design
- Purpose and role of computer system design in computer engineering
Learning objectives:
- Associate the themes involved with computer system design.
- Identify contributors to the subject area.
- Articulate the consequence of a good system design.
- Describe how computer engineering could make use of computer systems design.

CSE1. Overview of systems engineering

Minimum core coverage: 2 hours

Topics:
Nature of systems engineering; balancing costs, performance, market considerations.
Design decisions at the systems level, trade-offs. Need for flexibility and agility; reflects on the approach used. Competency of individuals. Hardware, software trade-offs.
Illustrations, applications.
The concept of a system, subsystem. Role of people. Recursive nature of this concept and relevance of this observation. The different disciplines involved, and the need for interdisciplinary approach.
The human computer interface; its importance in systems development and its influence.
Building reliable systems from unreliable components.
Different technologies, their strengths and weaknesses, the trends.
Standards and guidelines, legislation, regulations, professional issues.
Possibility of continually evolving systems.

Learning objectives:
1. To recognise and explain the disciplines and the possible interdisciplinary nature associated with the development of the range of computer-based systems.
2. To recognise and explain in general terms the mechanisms that are typically used in the development of computer based systems

CSE2. Theoretical considerations [elective]

Topics:
Reliability theory; underpinning probability rules; limitations of this in the systems context. Reliability of series systems and parallel systems; voting mechanisms, impact of maintenance regimes on reliability calculations.
Queueing theory
Simulation and modelling: discrete and continuous. Applications of these.
Technological considerations: performance issue related to different technologies; matching technologies.

Learning objectives:
1. Recognise the role that theoretical considerations have in relation to system design and the limitations of this theory
2. Apply the theoretical principles in designing systems which will exhibit certain performance levels.

**CSE3. Life cycle [core]**

*Minimum core coverage : 2 hours*

**Topics:**
- Influence of system size on choice of life cycle model and nature of system – agility issues.
- Different models of the life cycle – strengths and weaknesses of each.
- The concept of process. Process improvement. Basis for this is information.
- Gathering information.
- Maturity models. Standards and guidelines.

*Learning objectives:*
1. Recognise the need for a disciplined approach to system development and explain the elements of this in particular contexts.
2. Explain how data should be gathered to inform process improvement.

**CSE4. Requirements analysis and elicitation [core]**

*Minimum core coverage : 2 hours*

**Topics:**
- System analysis: identification of need, feasibility considerations, economic considerations.
- Nature of requirements: functional and non-functional requirements
- Approaches to the determination of requirements: analysis task, elements of this including communications and information gathering.
- Prototyping, simulation and modelling
- Human factors
- Building expertise over time
- Role of experts and experience
- Non-functional requirements; the range of possibilities, the quantification issue
- Human factors issues: standards, user interface design
- Specific applications; building computer systems such as desktops, laptops, hand-held devices, digital cameras, mobile phones, video phones

*Learning objectives:*
1. To describe the strengths and weaknesses of the major approaches to requirements elicitation and capture.
2. Apply one of a range of techniques to elicit and then describe the requirements for a particular system.
CSE5. Specification [core]

Minimum core coverage: 2 hours

Topics:
- Functional and non-functional specifications: different approaches and different possibilities
- Quality in relation to specification: completeness, consistency, simplicity, verifiability, basis for design. Specification in the event of failure.
- Test plans based on the specification: role of independence in relation to test.
- Safety cases.
- Limitations of such tests.
- Degraded mode of operation: possibilities, testing in this context

Learning objectives:
1. To recognise the characteristics of a high quality specification and to be able to assess the quality of a given specification.
2. To create a high quality specification of a given system.

CSE6. Architectural design [core]

Minimum core coverage: 3 hours

Topics:
- Basis for subdivision into systems and subsystems – basis for making these decisions
- Elements of high quality design
- Systems-level design strategies used in computer systems engineering – their strengths and weaknesses. Inclusion of diagnostics in the event of failure. The special problems of the hardware/software interface.
- Design issues associated with achieving reliability – role of redundancy.
- Different approaches to architectural design, their strengths and weaknesses
- Design to achieve performance measures, e.g. reliability, safety
- Concept of common cause failure
- Failure modes, approaches to fault tolerant design. Dealing with failure

Learning objectives:
1. Describe the strengths and weaknesses of the range of design decisions and methods.
2. Select and implement an appropriate approach to design for a range of possible applications.

CSE7. Implementation

Topics:
- Choosing technologies appropriate for particular purposes
- Rapid applications development
Role of standards and documentation in relation to this
Ensuring levels of performance, nature of tests, regression testing
Technology specific issues

**Learning objectives:**
1. Selecting technologies appropriate for achieving a high quality product over a range of applications.
2. Demonstrate the ability to implement at least one from a range of computer based systems

**CSE8. Testing**

**Topics:**
- Nature of testing – do throughout life cycle – efficient and effective processes
- Test plans – purpose, nature
- White board, black board, regression testing, stress testing, interface testing
- Tool support to accommodate efficient and effective development including regression testing.

**Learning objectives:**
1. Recognise the range of tests appropriate for each stage of the systems life cycle.
2. Select an appropriate combination of tests for ensuring the quality of a system.

**CSE9. Maintenance**

**Topics:**
- Inevitability of maintenance in certain systems – hardware upgrade, tool development, for instance
- Patterns of behaviour in relation to maintenance – hardware, software, communications, trends
- Measurement to inform maintenance, bottlenecks, etc.
- Nature of maintenance: defect removal, upgrade, enhancement
- Impact analysis; decision making in relation to maintenance, configuration control board role
- Configuration management and version control in engineering systems – the need for this, the issues associated with it, the nature of the information to be held; legal requirements; planning for possible disasters
- Tool support – the nature of this
- Building expertise for later re-use; the issues, balances, options

**Learning objectives:**
1. Understand the nature of maintenance in computer systems engineering and to recognise this
2. Recognise and apply mechanisms that ensure design support maintenance
3. How to apply the principles in situations of modest complexity

**CSE10. Project management**

**Topics:**
The nature of project management in systems engineering: the basic principles
Composition of teams, difficulty of software project management
Resource allocation
Allocation of decision making to teams: the issues, the options
Gantt charts, Project planning, costing, teamwork
Ensuring project management information; ensuring timely compliance with
specification; timely delivery
Standards, legal requirements, consultants subcontractors; their use and the
management of these
Role of metrics in support of management

Learning objectives:
1. To recognise and to know how to address the major problems of project
management in computer engineering including multi disciplinary issues.
2. To describe the tools used to provide evidence to support all phases of the systems
lifecycle.

CSE11. Specialist systems

Topics:
Risk and hazard analysis; strategies for risk reduction, risk control – implications
for implementation. Role of preliminary hazard analysis.
Concept of integrity level: quantifying this and its impact on the life cycle issues
Safety critical; concept of safety plan
Security critical systems and other high integrity systems; high integrity functions:
ensuring their performance
Design based on these key functions required for achieving the required integrity
level
Range of strategies for achieving a variety of possible high performance levels; to
include safety, reliability, security
Choosing approaches throughout the life cycle appropriate to required integrity
level
International standards, legal requirements

Learning objectives:
1. Recognise the special requirements of a range of specialist systems.
2. Demonstrate an ability to select approaches to the development of a range of
specialist systems that are commensurate with the intended integrity level

CSE12. Hardware and software co-design

Topics:
Applications areas with particular performance constraints that make the co-
ordinated development of both hardware and software important, e.g. speech
coders, radio modems
Demands of hard real-time features
Hardware and software co-design

Learning objectives:
1. Recognise the potential of hardware-software co-design in circumstances in which this approach is pertinent
2. Apply hardware-software co-design principles in situations of modest complexity
Computer Architecture and Organization (CAO)

- CAO0. History and overview of computer architecture [core]
- CAO1. Fundamentals of computer architecture [core]
- CAO2. Computer arithmetic [core]
- CAO3. Memory system organization and architecture [core]
- CAO4. Interfacing and communication [core]
- CAO5. Interface subsystems [core]
- CAO6. Processor systems design [core]
- CAO7. Organization of the CPU [core]
- CAO8. Performance [core]
- CAO9. Performance enhancements [elective]
- CAO10. Multiprocessing [core]

Computer architecture is concerned with all aspects of the design and organization of the central processing unit and the integration of the CPU into the computer system itself. Architecture extends upward into computer software because a processor’s architecture must cooperate with the operating system and system software. It is impossible to design an operating system well without a knowledge of the underlying architecture. Moreover, the computer designer has to have an intimate understanding of software in order to implement the optimum architecture. Moreover, there is a tighter relationship between the computer architect and the compiler writer than at any time in the past.

Computer architecture is a key component of computer engineering and the practicing computer engineer should have a practical understanding of this topic. Consequently, computer architecture courses should include a laboratory component where students are able to evaluate alternative designs.

The term *architecture* implies structure and therefore computer architecture tells us something about the way in which the elements of a computer relate to each other. Computer architecture is generally thought of as the *programmer’s* view of a computer; that is, the *idealized* or *abstract* view of a computer. The *implementation* or *organization* of the computer is hidden from the programmer, although it would be wrong to divorce entirely architecture and implementation because each exerts a powerful influence on the other.

The computer architecture curriculum has to achieve multiple objectives. It must provide an overview of computer architecture and teach students the operation of a typical von Neumann machine. It must highlight the important issues facing today’s designers and give students the tools they will need to carry out research. Ideally it should reinforce topics that are common to other areas of computer science; for example, teaching register indirect addressing reinforces the concept of pointers in C.

As with any engineering program students must learn to design and build things. Therefore, the computer engineering student should be expected to design and build a
simple computer or computer system. This will most likely take place in a laboratory environment.

**CAO0. History and overview of computer architecture [core]**

*Suggested time:* 1 hour

**Topics:**
- Knowledge themes include system organization and architecture, memory, interfacing, microprocessors, system organization
- Contributors to the subject include Zuse, Atanasoff, von Neumann, Eckert, Mauchly, and Wilkes
- Contrasts between computer organization and computer architecture
- Purpose and role of computer architecture in computer engineering

**Learning objectives:**
1. Associate the themes involved with computer architecture.
2. Identify contributors to the subject area.
3. Articulate differences between computer organization and computer architecture.
4. Describe how computer engineering could make use of computer architecture.

**CAO1. Fundamentals of computer architecture [core]**

*Minimum core coverage time: 18 hours*

**Topics:**
1. Organization of the von Neumann machine
2. Instruction formats
3. The fetch/execute cycle; instruction decoding and execution
4. Registers and register files.
5. Instruction types and addressing modes
6. Subroutine call and return mechanisms
7. Programming in assembly language
8. I/O techniques and interrupts
9. Other design issues.

**Learning objectives:**
1. Be able to explain the organization of a von Neumann machine and its major functional units.
2. Be able to explain how an instruction is fetched from memory and executed.
3. Be able to articulate the strengths and weaknesses of the von Neumann architecture.
4. Be able to explain the relationship between the representation of machine level operation at the binary level and their representation by a symbolic assembler.
5. Be able to explain why a designer adopted a given different instruction formats, such as the number of addresses per instruction and variable length vs. fixed length formats.
6. Be able to write small programs and fragments of assembly language code to demonstrate an understanding of machine level operations.
7. Be able to implement some fundamental high-level programming constructs at the machine-language level.
8. Be able to use computer simulation packages to investigate assembly language programming.

**CA2. Computer arithmetic [core]**

*Minimum core coverage time: 3 hours*

**Topics:**
1. Representation of integers (positive and negative numbers)
2. Algorithms for common arithmetic operations (addition, subtraction, multiplication, division)
3. Significance of range, precision, and accuracy in computer arithmetic
4. Representation of real numbers (standards for floating-point arithmetic)
5. Algorithms for carrying out common floating-point operations
6. Converting between integer and real numbers
7. Multi-precision arithmetic
8. Hardware and software implementation of arithmetic unit.
9. The generation of higher order functions from square roots to transcendentals.

**Learning objectives:**
1. Appreciate how numerical values are represented in digital computers
2. Understand the limitations of computer arithmetic and the effects of errors on calculations.
3. Appreciate the effect of a processor’s arithmetic unit on its overall performance,

**CAO3. Memory system organization and architecture [core]**

*Minimum core coverage time: 9 hours*

**Topics:**
1. Memory systems hierarchy
2. Coding, data compression, and data integrity
3. Electronic, magnetic and optical technologies
4. Main memory organization and its characteristics and performance
5. Latency, cycle time, bandwidth, and interleaving
6. Cache memories (address mapping, line size, replacement and write-back policies)
7. Virtual memory systems
8. Memory technologies (DRAM, EPROM, FLASH, etc.)
9. Reliability of memory systems. Error detecting and error correcting systems.

**Learning objectives:**
1. Identify the main types of memory technology.
2. Explain the effect of memory latency and bandwidth on performance.
3. Explain the use of memory hierarchy to reduce the effective memory latency.
4. Describe the principles of memory management.
5. Appreciate how errors in memory systems arise and what can be done about them.
CAO4. Interfacing and communication [core]
Minimum core coverage time: 12 hours

Topics:
1. I/O fundamentals: handshaking, buffering,
2. I/O techniques: programmed I/O, interrupt-driven I/O, DMA
3. Interrupt structures: vectored and prioritised, interrupt overhead, interrupts and reentrant code.
4. Memory system design and interfacing.
5. Buses: bus protocols, local and geographic arbitration

Learning objectives:
1. Explain how interrupts are used to implement I/O control and data transfers.
2. Write small interrupt service routines and I/O drivers using assembly language.
3. Identify various types of buses in a computer system.
4. Describe data access from a magnetic disk drive.
5. Be able to analyse and implement interfaces.

CAO5. Interface subsystems [core]
Minimum core coverage time: 12 hours

Topics:
1. External storage systems; organization and structure of disk drives and optical memory
2. Basic I/O controllers, keyboard, mouse, etc.
3. RAID architectures
4. Video control
5. I/O Performance
6. SMART technology and fault detection
7. Processor to network interfaces

Learning objectives:
1. Compute the various parameters of performance for standard I/O types.
2. Explain the basic nature human computer interaction devices.
3. Describe data access from magnetic and optical disk drives.

CAO6. Processor systems design [core]
Minimum core coverage time: 12 hours

Topics:
1. The CPU interface: clock, control, data and address buses
2. Address decoding and memory interfacing
3. Basic parallel and serial interfaces
4. Timers
5. System firmware
Learning objectives:
1. Appreciate how a CPU chip is turned into a complete system.
2. Be able to design an interface to memory
3. Understand how to interface and use peripheral chips
4. Write sufficient EPROM-based system software to create a basic stand-alone system.
5. Be able to specify and design simple computer interfaces.

CAO7. Organization of the CPU [core]
Minimum core coverage time: 12 hours

Topics:
1. Implementation of the von Neumann machine
2. Single vs. multiple bus datapaths
3. Instruction set architecture; machine architecture as a framework for encapsulating design decisions.
4. Relationship between the architecture and the compiler
5. Implementing instructions
6. Control unit: hardwired realization vs. microprogrammed realization
7. Arithmetic units, for multiplication and division.
8. Instruction pipelining
9. Trends in computer architecture: CISC, RISC, VLIW
10. Introduction to instruction-level parallelism (ILP)
11. Pipeline hazards: structural, data and control
12. Reducing the effects of hazards

Learning objectives:
1. Compare alternative implementation of datapaths.
2. Discuss the generation of control signals using hardwired or microprogrammed implementations.
3. Explain basic instruction level parallelism using pipelining and the major hazards that may occur.
4. Explain what has been done to overcome the effect of branches
5. Discuss the way in which instruction sets have evolved to improve performance; for example, predicated execution.

CAO8. Performance [core]
Minimum core coverage time: 3 hours

Topics:
1. Metrics for computer performance; clock rate, MIPS, Cycles per instruction, benchmarks
2. Strengths and weaknesses of performance metrics  
3. Averaging metrics: arithmetic, geometric and harmonic  
4. The role of Amdahl’s law in computer performance

Learning objectives:  
1. Appreciate all the factors that contribute to computer performance.  
2. Understand the limitations of performance metrics  
3. Be able to select the most appropriate performance metric when evaluating a computer.  
4. Discuss the impact on control and datapath design for performance enhancements.

CAO9. Performance enhancements [elective]

Topics:  
1. Superscalar architecture  
2. Branch prediction  
3. Prefetching  
4. Speculative execution  
5. Multithreading  
6. Scalability  
7. Short vector instruction sets; Streaming extensions, AltiVec; relationship between computer architecture and multimedia applications.

Learning objectives:  
1. Discuss how various architectural enhancements affect system performance.  
2. Discuss how parallel processing approaches can be applied to the design of scalar and superscalar processors.  
3. Discuss how vector processing techniques can be applied to enhance instruction sets to be used for multimedia, signal processing, etc.  
4. Appreciate how each of the functional parts of a computer system affect its overall performance. Be able to estimate the effect on system performance of changes to functional units.

CAO9. Multiprocessing [elective] consider moving this to HPC section.

Minimum core coverage time: 3 hours

Topics:  
1. Systolic architectures  
2. Interconnection networks (hypercube, shuffle-exchange, mesh, crossbar)  
3. Shared memory systems  
4. Cache coherence  
5. Memory models and memory consistency

Learning objectives:
1. Discuss the concept of parallel processing beyond the classical von Neumann model.
2. Describe the limitations imposed by interconnections on multiprocessing systems.
3. Appreciate the problems caused by cache coherency and understand the ways in which the problem can be overcome.
Software Engineering (SWE)

SWE0. History and overview of software engineering [core]
SWE1. Software processes [core]
SWE2. Software requirements and specifications [core]
SWE3. Software design [core]
SWE4. Software testing and validation [core]
SWE5. Software evolution [elective]
SWE6. Software tools and environments [core]
SWE7. Software project management [elective]

Software engineering is the discipline concerned with 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.

The creation of programs benefits from concepts and practices from software engineering. There is scope for introducing fundamental ideas from software engineering into elementary programming and into early experience for software design.

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.

SWE0. History and overview of software engineering [core]

*Suggested time:* 1 hour

*Topics:*

- Knowledge themes include software processes, life cycle, requirements and specifications, and the importance of testing and validation
- Contributors to the subject include government agencies, industry, and academic institutions
- Software engineering vs. computer engineering
- Purpose and role of software engineering in computer engineering

*Learning objectives:*

- Associate the themes involved with software engineering.
- Identify contributors to the subject area.
- Articulate differences between computer engineering and software engineering.
- Describe how computer engineering could make use of software engineering.
SWE1. Software processes [core]

*Suggested time:* 2 hours

*Topics:*

- Software life cycle and process models
- Process assessment models
- Software process metrics

*Learning objectives:*

1. Select, with justification, the software development models most appropriate for the development and maintenance of diverse software products.
2. Explain the role of process maturity models.

SWE2. Software requirements and specifications [core]

*Suggested time:* 2 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. 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 (e.g., structured analysis or object-oriented-analysis).
3. Conduct a review of a software requirements document using best practices to determine the quality of the document.
4. Translate into natural language a software requirements specification written in a commonly used formal specification language.

SWE3. Software design [core]

*Suggested time:* 2 hours

*Topics:*

- Fundamental design concepts and principles
- Software architecture
- Structured design
- Object-oriented analysis and design
Component-level design
Design for reuse

Learning objectives:
1. Evaluate the quality of multiple software designs based on key design principles and concepts.
2. Using a software requirement specification and a common program design methodology and notation, create and specify the software design for a medium-size software product (e.g., using structured design or object-oriented design).
3. Using appropriate guidelines, conduct the review of a software design.

SWE4. Software testing and validation [core]
Suggested time: 2 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. Demonstrate the application of the different types and levels of testing (unit, integration, systems, and acceptance) to software products of medium size.
2. Undertake, as part of a team activity, an inspection of a medium-size code segment.
3. Describe the role that tools can play in the validation of software.

SWE5. Software evolution [elective]
Suggested time: 4 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. Outline the process of regression testing and its role in release management.
3. Estimate the impact of a change request to an existing product of medium size.
4. Develop a plan for re-engineering a medium-sized product in response to a change request.
5. Discuss the advantages and disadvantages of software reuse.
6. Demonstrate the ability to exploit opportunities for software reuse in a variety of contexts.

**SWE6. Software tools and environments [core]**

*Suggested time: 2 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 software 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.

**SWE7. Software project management [elective]**

*Suggested time: 4 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.
ESY. Embedded Systems

ESY0. History and overview of embedded systems [core]
ESY1. Fundamentals of embedded systems [core]
ESY2. Language issues [core]
ESY3. Hardware considerations [core]
ESY4. Mapping between languages and hardware [core]
ESY5. Real-time Operating Systems [elective]
ESY6. Classification of embedded systems [elective]
ESY7. Software engineering considerations [elective]
ESY8. Particular techniques and applications [elective]
ESY9. Tool support [elective]
ESY10. High integrity software systems [elective]

Embedded software forms an important component of many facets of computer engineering. Ultimately the software is one aspect of a system and it is this system that has typically to perform to a high level. With the rapid technological developments and the corresponding rise in the range of diverse applications, embedded systems are becoming ever more complex with ever greater functionality. There are many problems that are peculiar to the development of software for these systems. These are beyond the normal demands of a typical course in software engineering. Thus design often depends on a particularly specialist knowledge of the application domain; it may need a detailed understanding of input/output issues often including network protocols; methods of interaction are often paramount and dealing with timing issues can be an essential part of this. An additional challenge often faced by the designer in certain contexts is the provision of a reliable system based on the utilisation of unreliable resources. In addition the maintenance issues tend to exhibit an additional and different level of complexity with this being caused by possible changes to the hardware and the impact of this on such matters as timing as well as input and output.

ESY0. History and overview of embedded systems [core]

_Suggested time: 1 hour_

_Topics:_

- Knowledge themes include mapping between language and hardware, classifications, influence of software engineering, applications and techniques, and tool support
- Contributors to the subject include many individuals from a variety of industries
- Contrast between embedded systems and other computer systems
- Purpose and role of embedded systems in computer engineering

_Learning objectives:_

- Associate the themes involved with embedded systems.
- Identify contributors to the subject area.
- Describe how computer engineering could make use of embedded systems.
ESY1. **Fundamentals of embedded systems [core]**

*Minimum core coverage time: 4 hours*

**Topics:**
- Nature of embedded systems: typically seen as part of system development with requirements coming from the wider context; typically involves decisions about both hardware and software.
- Embedded software characteristics: reliability considerations and impact of this on system design and implementation. Data from sensors, output to actuators. Nature and properties of a range of these input and output devices; consequences for software.
- Illustrations and applications.
- Overview of applications of embedded systems as related to reliability, safety, security, fault tolerance, risk assessment, etc.

**Learning objectives:**
1. Understand the nature of embedded software systems and their characteristics
2. Recognise the implications for implementation.

ESY2. **Language issues [core]**

*Minimum core coverage time: 3 hours*

**Topics:**
- Programming requirements of embedded software systems including need to access hardware, e.g. to deal with input and output, to access a network
- The hardware context: micro-processors, micro-controllers; the implications for programming
- Performance requirements arising from single system considerations, multi-processor possibilities. Role of concurrency, fault tolerance, exception handlers
- Range of possible programming languages to support embedded software design: their capabilities, strengths and weaknesses
- Choice of programming languages for particular applications: principles to guide choice; role of tools and software support
- Multi-language possibilities

**Learning objectives:**
1. To explain the issues associated with the selection of programming language for a range of possible embedded system applications
2. To select with explanation a programming language appropriate for the implementation of a range of embedded system applications
ESY3. Hardware considerations [elective]

Topics
Input/output interfaces: the nature of these and their impact on programming; timing considerations. Interrupts. On-clip interfaces; overlapping input/output and programming.


Embedded motor controllers. Real-time clocks.

Learning objectives
1. To recognise the hardware issues associated with input and output and relevant to embedded systems development
2. To demonstrate the ability to write embedded systems software that communicates effectively with a range of hardware devices.

ESY4. Mapping between languages and hardware [core]
Minimum core coverage time: 3 hours

Topics:
• Mapping of language constructs onto hardware including interrupt handling and exception handling; analogue considerations
• Memory management issues including garbage collection and related techniques
• Multiple processes and associated priority considerations
• Run-time considerations, the nature and structure of run-time systems arising from a range of language constructs including concurrency and requests for priority treatment
• Concurrency mechanisms; performance issues and multiprocessor considerations, actors, interaction patterns
• Scheduling: the various options including pre-emptive and non pre-emptive scheduling
• Environment in which programs execute including application specific real time operating system considerations

Learning objectives:
1. Understand and explain the special mechanisms that exist in a programming language for meeting the special requirements of embedded software systems.
2. Choose the environment that best suits a range of applications, taking into account the characteristics and properties of application specific real-time operating systems and run-time systems.
3. Write software that uses interrupts, manages memory, contains multiple processes, etc.
**ESY5. Real-time Operating Systems [elective]**

**Topics**
The nature of real-time operating systems: the kernel and its role in satisfying real-time constraints; to include timing, fault tolerance, scheduling, input/output.

Functionality of the operating system: its role and its purpose.

Scheduling issues: the nature of the facilities typically provided.

Worst time execution calculations. Time and time stamping. Other alternatives

**Learning objectives**
1. To recognise that features expected of a real-time operating system and to be able to defend whether it is appropriate to use these in particular situations.
2. To demonstrate the effective and efficient use of real-time operating systems.

**ESY6. Classification of embedded systems [core]**

*Minimum core coverage: 6 hours*

**Topics:**
- Real-time systems, reactive systems, process control systems, high integrity systems, critical systems; the different nature of these and the implications for embedded systems.
- Characteristics of a range of devices; design of programs dealing with input and output including communication over a network; drivers for a range of devices which may exhibit strict timing regimes
- Systems dealing with multiple input/output, priority and timing considerations; run-time support; design conditioned or determined by requirements of other components whose needs have to be serviced
- Scheduling issues: real-time considerations, use of a monitor
- Reactive systems where the software reacts to changes in the environment of the system: applications and issues.
- Process control considerations: systems that exhibit feedback, control of these systems; model based systems and control
- Agent technology: definitions, applications, stateless systems, reactive systems, state-based systems, model-based systems, intelligent systems.
- Self test programs that check the proper performance of hardware including memory and devices
- The special nature and characteristics of critical systems

**Learning objectives:**
1. To recognise the different classes of embedded systems and the particular implementation issues associated with each such system.
2. To choose an approach appropriate to a range of possible applications.

**ESY7. Software engineering considerations** [elective]

*Minimum core coverage: 6 hours*

**Topics:**
- Requirements and specification: origins of these are typically the wider system considerations
- Design approaches to support embedded systems; continuous mode of operation, discrete mode of operation; design frameworks used in embedded systems design including synchronous and non-asynchronous, discrete and continuous, use of middleware
- Design patterns and the use of design notations and/or languages
- Error handling: error detection, recovery from errors, often automatically
- Testing in the context of embedded systems; the special issues; testing to achieve particular performance requirements
- Issues of version control and configuration management associated with maintenance, including update of hardware and portability considerations
- Reliability theory as it applies to software
- Metrics and their role in respect of the particular problems of managing embedded systems development

**Learning objectives:**
1. Recognise the particular features of embedded software systems and explain their impact on the software development process
2. Apply a range of technologies from software engineering to the particular problems of embedded software systems development

**ESY8. Particular techniques and applications** [elective]

**Topics:**
- Polling vs interrupt handling – the problems and benefits and choosing between them
- Common design patterns and frameworks used in embedded software systems design
- Interrupts, buffering strategies and error situations
- Techniques associated with downloading software automatically from a website
- Embedding databases in systems
- Embedding web sites in devices – the technical challenges and the benefits
- Security matters – interlocks and firewalls
- Principles of construction of software for a range of digital devices including digital cameras, hand-held systems; docking system software

**Learning objectives:**
1. Understand the range of techniques peculiar to the development of a range of embedded software systems
2. Choose a technique appropriate for one of a range of possible application and demonstrate its applicability in that context.

**ESY9. Tool support** [elective]

*Minimum core coverage : 3 hours*

**Topics:**
- The need for tool support for embedded software development – the nature of the requirements of these tools; impact on maintenance
- Tool choice and its impact on the development process including its quality; co-operating sets of tools
- Tool certification, measures of quality; run-time systems and the impact of this on code generation and licensing
- Compiler considerations: cross compilation, efficiency, portability
- Design tools
- Configuration management and version control support
- Debugging Tools

**Learning objectives:**
1. Recognise the benefits and advantages of tool support and the nature of this
2. Choose tools appropriate for a task.

**ESY10. High integrity software systems** [elective]

**Topics:**
- Nature and characteristics (including performance requirements) of a range of high integrity embedded software systems including safety and security critical systems
- Risk and hazard analysis and assessment; different approaches; their strengths and weaknesses. The safety case concept.
- Integrity levels: their impact on the software development process.
- Appropriate software life cycle models that include the allocation of responsibility for ensuring levels of integrity.
- Consideration of and evaluation of methods used throughout the life cycle in the development of high integrity software systems required to achieve a particular integrity level.
- Discussion of the role of formal methods and alternative strategies.
- Design approaches; the role of standards.
- Fault tolerance, diversity. Tools to support development
- To include language considerations, coding standards, approaches to verification and validation; the need to quantify reliability.
- Relevant verification and validation to address the non-functional requirements.
- Management issues, the particular problem of software, assessment of software quality, independence arguments, the role of metrics.
Learning objectives:
1. to understand the life cycle issues together with the methods, techniques and tool support appropriate to identifying the requirements for and specifying the functionality of high integrity software systems
2. to select approaches (and to justify that selection) that will result in the efficient and effective development of software systems of a particular integrity level, and the maintenance of these
3. to understand the context and the range of professional issues which should guide the activities of a software engineer involved in particular in the development of high integrity systems
4. to have demonstrated practical competence in the range of issues associated with high integrity systems development
OPERATING SYSTEMS (OPS)

OPS0. History and overview of operating systems [core]
OPS1. Operating system function and design [core]
OPS2. Operating system principles [core]
OPS3. Concurrency [core]
OPS4. Scheduling and dispatch [core]
OPS5. Memory management [core]
OPS6. Device management [core]
OPS7. Security and protection [elective]
OPS8. File systems [elective]
OPS9. System performance evaluation [core]

An operating system defines a software interface of the computer hardware and the architecture with which computer engineers can control and exploit the hardware to provide maximum benefit at the user level. It also manages sharing of resources (hardware and software) among the computer’s users (user programs and systems programs). This set of knowledge units explains the issues that influence the design of contemporary operating systems given the underlying hardware components and the architecture and includes a laboratory component to enable students to experiment with operating systems.

It is necessary that the student understands the basic principles and the purposes of an operating system prior to a study of digital instrumentation and embedded systems. It is necessary to addresses both the use of operating systems (externals) and their design and implementation (internals). Many of the ideas involved in operating system use have wider applicability across the field of computer engineering such as concurrent programming. Studying internal design has relevance in such diverse areas as fault tolerance, algorithm design and implementation, modern device development, building virtual environments, building secure and safe systems, network management, and many others.

OPS0. History and overview of operating systems [core]

*Suggested time: 1 hour*

*Topics:*

- Knowledge themes include principles concurrency, scheduling, dispatching, memory and device management
- Contributors to the subject include IBM, AT&T Bell Labs, Microsoft, and Sun
- Contrasts between different operating systems such as Windows, UNIX, and Mac OS
- Purpose and role of operating systems in computer engineering

*Learning objectives:*

- Associate the themes involved with operating systems.
- Identify contributors to the subject area.
- Articulate general differences between different operating systems.
Describe how computer engineering could make use of operating systems.

**OPS1. Operating system function and design [core]**

*Suggested time:* 2 hours

**Topics:**
- Functionality of a typical operating system
- Mechanisms to support client-server models, hand-held devices
- Design issues (efficiency, robustness, flexibility, portability, security, compatibility)
- Influences of security, networking, multimedia, windows

**Learning objectives:**
- Explain the range of requirements that a modern operating system has to address.
- Define the functionality that a modern operating system must deliver to meet a particular need.
- Articulate design tradeoffs inherent in operating system design.

**OPS2. Operating system principles [core]**

*Suggested time:* 3 hours

**Topics:**
- Structuring methods (monolithic, layered, modular, micro-kernel models)
- Abstractions, processes, and resources
- Concepts of application program interfaces (APIs)
- Applications needs and the evolution of hardware/software techniques
- Device organization
- Interrupts: methods and implementations
- Concept of user/system state and protection, transition to kernel mode

**Learning objectives:**
1. To demonstrate understanding of Operating Systems as an interface between user programs and the computer hardware.
2. To demonstrate understanding of the logical layers and the benefits of building these layers in a hierarchical fashion.
3. Relate system state to user protection.

**OPS3. Concurrency [core]**

*Suggested time:* 6 hours

**Topics:**
- States and state diagrams
- Structures (ready list, process control blocks, and so forth)
- Dispatching and context switching
- The role of interrupts
Concurrent execution: advantages and disadvantages
The “mutual exclusion” problem and some solutions
Deadlock: causes, conditions, prevention
Models and mechanisms (semaphores, monitors, condition variables, rendezvous)
Producer-consumer problems and synchronization
Multiprocessor issues (spin-locks, reentrancy)

Learning objectives:
1. Justify the presence of concurrency within the framework of an operating system.
2. Demonstrate the potential run-time problems arising from the concurrent operation of many (possibly a dynamic number of) tasks.
3. Summarize the range of mechanisms (at an operating system level) that can be employed to realize concurrent systems and be able to describe the benefits of each.
4. Explain the different states that a task may pass through and the data structures needed to support the management of many tasks.

OPS4. Scheduling and dispatch [core]

Suggested time: 3 hours

Topics:
- Preemptive and non-preemptive scheduling
- Schedulers and policies
- Processes and threads
- Deadlines and real-time issues

Learning objectives:
1. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems.
2. Describe relationships between scheduling algorithms and application domains.
3. Investigate the wider applicability of scheduling in such contexts as disk I/O, networking scheduling, and project scheduling.

OPS5. Memory management [core]

Suggested time: 5 hours

Topics:
- Review of physical memory and memory management hardware
- Overlays, swapping, and partitions
- Paging and segmentation
- Placement and replacement policies
- Working sets and thrashing
- Caching
Learning objectives:
1. Introduce memory hierarchy and cost-performance tradeoffs.
2. Explain what virtual memory is and how it is realized in hardware and software.
3. Examine the wider applicability and relevance of the concepts of virtual entity and of caching.
4. Evaluate the trade-offs in terms of memory size (main memory, cache memory, auxiliary memory) and processor speed.
5. Defend the different ways of allocating memory to tasks on the basis of the relative merits of each.

OPS6. Device management [core]
Suggested time: 4 hours

Topics:
- Characteristics of serial and parallel devices
- Abstracting device differences
- Buffering strategies
- Direct memory access
- Recovery from failures

Learning objectives:
1. Identify the relationship between the physical hardware and the virtual devices maintained by the operating system.
2. Differentiate the mechanisms used in interfacing a range of devices (including hand-held devices, networks, multimedia) to a computer and explain the implications of these for the design of an operating system.
3. Implement a simple device driver for a range of possible devices.

OPS7. Security and protection [elective]
Suggested time: 4 hours

Topics:
- Overview of system security
- Policy/mecchanism separation
- Security methods and devices
- Protection, access, and authentication
- Models of protection
- Memory protection
- Encryption
- Recovery management
Learning objectives:
1. Defend the need for protection and security, and the role of ethical considerations in computer use.
2. Summarize the features of an operating system used to provide protection and security, and describe the limitations of each of these.
3. Compare and contrast current methods for implementing security.

OPS8. File systems [elective]

Suggested time: 5 hours

Topics:
Files: data, metadata, operations, organization, buffering, sequential, nonsequential
Directories: contents and structure
File systems: partitioning, mount/unmount, and virtual file systems
Standard implementation techniques
Memory-mapped files
Special-purpose file systems
Naming, searching, access, backups

Learning objectives:
1. Summarize the full range of considerations that support file systems.
2. Compare and contrast different approaches to file organization, recognizing the strengths and weaknesses of each.

OPS9. System performance evaluation [elective]

Suggested time: 4 hours

Topics:
Why and what system performance needs to be evaluated
Policies for caching, paging, scheduling, memory management, security, and so forth
Evaluation models: deterministic, analytic, simulation, or implementation-specific
How to collect evaluation data (profiling and tracing mechanisms)

Learning objectives:
1. Describe the performance metrics used to determine how a system performs.
2. Explain the main evaluation models used to evaluate a system.
Circuits and Systems (CSY)

CSY0. History and overview of systems and circuits [core]
CSY1. Fundamental Electrical Quantities (core)
CSY2. Resistive Circuits and Networks (core)
CSY3. Reactive Circuits and Networks (core)
CSY4. Frequency Response (core)
CSY5. Sinusoidal Analysis (core)
CSY6. Convolution (core)
CSY7. Discrete Time Signals (core)
CSY8. Fourier Analysis (core)
CSY9. Filters (elective)
CSY10. Laplace Transforms (elective)
CSY11. z – Transforms (elective)
CSY12. Digital Filters (elective)

[This area certainly needs to be integrated with any DSP section that we include. I also think that additional discussion of the amount of signals that is in the core will be appropriate. I think leaving it this way will stimulate that discussion]

Systems and circuits is foundational material for computer engineering. These areas provide the basic knowledge for the design of the circuits used to implement computers. This is a very broad area and it is expected that there will be a great deal of variation between programs is the coverage of topics outside of the core.

CSY0. History and overview of systems and circuits [core]

*Suggested time:* 1 hour

*Topics:*
- Knowledge themes include RLC circuits, energy storage, frequency response, and sinusoidal analysis
- Contributors to the subject include Volta, Ohm, Hertz, Farad, and Henry
- Contrast between discrete-time and continuous-time signals
- Purpose and role of systems and circuits in computer engineering

*Learning objectives:*
- Associate the themes involved with systems and circuits.
- Identify contributors to the subject area.
- Articulate differences between discrete and continuous time signals.
- Describe how computer engineering could make use of systems and circuits.

CSY1. Fundamental Electrical Quantities (core)

*Suggested time:* 3 hours

*Topics:*
- Charge
Current
Voltage
Energy
Power

**Learning Objectives:**
1. To be able to understand the concept of and to represent basic electrical quantities.
2. To understand the relationships between basic electrical quantities.

**CSY2. Resistive Circuits and Networks (core)**

*Suggested time:* 20 hours

**Topics:**
- Ohm’s Law
- Kirchoff’s laws
- Independent and dependent sources
- Series and parallel elements
- Voltage and Current measurement
- Mesh and Nodal analysis
- Superposition
- Thevenin’s and Norton’s theorems
- Maximum power transfer

**Learning Objectives:**
1. To be able to represent and manipulate basic resistive circuit equations.
2. To be able to analyze and simplify basic resistive circuits.
3. To understand and use network analysis tools for resistive circuits.

**CSY3. Reactive Circuits and Networks (core)**

*Suggested time:* 14 hours

**Topics:**
- Inductance
- Capacitance
- Mutual inductance
- Time constants for RL and RC circuits
- Transient response of RL, RC, and RLC circuits
- Damping
- Transformers

**Learning Objectives:**
1. To be able to represent basic energy storage devices.
2. To understand how to combine various combinations of inductors and capacitors.
3. To understand simple transient response of various R, L, and C circuits.
4. To be able to analyze and design simple R, L, and C circuits.

**CSY4. Frequency Response (core)**  
*Suggested Time:* 10 hours

**Topics:**  
- Response of RL, RC, and RLC circuits  
- Transfer functions  
- Two-port circuits  
- Parallel and series resonance

**Learning Objectives:**  
1. To understand the frequency domain characteristics of electrical circuits.  
2. To be able to analyze and design frequency selective circuits.

**CSY5. Sinusoidal Analysis (core)**  
*Suggested time:* 7 hours

**Topics:**  
- Phasor representation of voltage and current  
- Forced response to sinusoidal functions  
- Impedance and Admittance  
- Nodal and Mesh analysis  
- Thevenin’s and Norton’s theorems  
- Phasor diagrams  
- Superposition  
- Source transformations

**Learning Objectives:**  
1. To understand response of electrical circuits to sinusoidal signal excitation.  
2. To be able to analyze circuits using the techniques given.

**CSY6. Convolution (core)**  
*Suggested time:* 4 hours

**Topics:**  
- Impulse response  
- Convolution integral  
- Physically realizable systems  
- Graphical methods

**Learning Objectives:**  
1. To be able to use the convolution technique to analyze circuits.  
2. To be able to represent convolution using graphical techniques.
CSY7. Discrete Time Signals (core)
*Suggested time:* 8 hours

*Topics:*
- Representation of signals
- Sampling of signals
- Quantizing
- Aliasing
- Difference Equations

*Learning Objectives:*
1. To understand the discrete-time representation of signals.
2. To understand errors introduced by sampling and quantizing.

CSY8. Fourier Analysis (core)
*Suggested time:* 6 hours

*Topics:*
- Signal representation by Fourier series
- Trigometric Fourier series
- Exponential Fourier series
- Definition of the Fourier transform
- Properties of the Fourier transform
- Circuit analysis using the Fourier transform

*Learning Objectives:*
1. To be able to represent signals using Fourier series.
2. To understand the Fourier transform and its properties.
3. To apply Fourier transform techniques to circuit analysis.

CSY9. Filters (elective)
*Suggested time:* 5 hours

*Topics:*
- Frequency selective circuits
- Transfer functions
- Passive filters
- Active filters

*Learning Objectives:*
1. To understand frequency selective circuits.
2. To be able to design filters that have specified frequency characteristics.

CSY10. LaPlace Transforms (elective)
*Suggested Time:* 8 hours
Topics:

- LaPlace transform integral
- Impulse response
- Step functions
- Ramp functions
- Inverse transforms
- Poles and zeroes
- Initial valve theorem
- Final valve theorem
- Circuit analysis using LaPlace transforms

Learning Objectives:
1. To understand the LaPlace transform technique and its mathematical representation.
2. To be able to represent circuits and signals by their LaPlace transforms.
3. To be able to use the LaPlace transform to describe electrical circuits and compute their behavior.

CSY11. z – transforms (elective)
Suggested time: 4 hours

Topics:

- Properties of the z – transform
- Inverse z – transforms
- Difference equations

Learning Objectives:
1. To understand the properties and uses of the z – transform.

CSY12. Digital Filters (elective)
Suggested time: 4 hours

Topics:

- Frequency response of discrete – time systems
- Recursive filter design
- Nonrecursive filter design
- Windowing

Learning Objectives:
1. To understand frequency selective circuits in the z – transform domain.
2. To be able to design digital filters that have specified frequency characteristics.
Networks (NWK)

NWK0. History and Overview of Networks [core]
NWK1. Communications Network Architecture [core]
NWK2. Communications Network Protocols [core]
NWK3. Local and Wide Area Networks [core]
NWK4. The web as an example of client-server computing [core]
NWK5. Data Security and Integrity [elective]
NWK6. Performance Evaluation [elective]
NWK7. Data Communications [elective]
NWK8. Wireless and Mobile Computing [elective]

NWK0. History and Overview of Networks [core]
Suggested time: 1 hour

Topics:
- Knowledge themes include network architectures and protocols, network types such as LAN, WAN, MAN, and wireless
- Contributors to the subject include Hamming, Huffman, and Cerf
- Contrasts between network architectures and protocols
- Purpose and role of networks in computer engineering

Learning objectives:
- Associate the themes involved with networks.
- Identify contributors to the subject area.
- Describe how computer engineering could make use of networks.

NWK1. Communications Network Architecture [core]
Suggested time: 4 hrs

Topics:
- Network Line Configuration (Point-to-point, Multipoint)
- Networking and Internetworking Devices
  - Repeaters, Bridges, Switches, Routers, Gateways
- Network Topologies (Mesh, Star, Tree, Bus, Ring)
- Connection-oriented and Connectionless services

Learning Objectives:
1. To be able to understand fundamental concepts of networks & their topologies.
2. To understand the concept of network architecture and its hardware components.

NWK2. Communications Network Protocols [core]
Suggested time: 6 hours

Topics:
- Network Protocol (Syntax, Semantics, Timing)
Protocol Suites (TCP/IP)
Layered Protocol Software (Stacks)
  Physical layer networking concepts
  Data link layer concepts
  Internetworking and routing
Network Standards and standardization bodies

Learning Objectives:
1. To demonstrate understanding of the elements of a protocol, and the concept of layering.
2. To be able to recognize the importance of networking standards, and their regulatory committees.
3. To be able to describe the seven layers of the OSI model.
4. To be able to compare and contrast the OSI model with the TCP/IP model.
5. To demonstrate understanding of the differences between circuit switching and packet switching.

NWK3. Local and Wide Area Networks [core]
Suggested time: 10 hours

Topics:
LAN Topologies (Bus, Ring, Star)
LAN Technologies (Ethernet, Token Ring, Gigabit Ethernet)
Error Detection and Correction
Carrier Sense Multiple Access Networks (CSMA)
Large Networks and Wide Areas
Circuit Switching, Packet Switching
Protocols (addressing, congestion control, virtual circuits, quality of service)

Learning Objectives:
1. To demonstrate understanding of basic concepts of LAN and WAN technologies and topologies.
2. To demonstrate understanding of different components and requirements of network protocols.
3. To demonstrate understanding of basic concepts of error detection and correction at the data link layer and below.

NWK4. The web as an example of client-server computing [core]
Suggested 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
Support tools for web site creation and web management

**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).

**NWK5. Data Security and Integrity [elective]**

*Suggested time: 9 hours*

**Topics:**
- Secure Networks, Cryptography
- Encryption and Privacy
  - Public Key, Private Key, Symmetric Key
- Authentication Protocols
- Packet Filtering
- Firewalls
- Virtual Private Networks
- Transport Layer Security

**Learning Objectives:**

1. To understand common barriers to network security and the major issues involved in implementing proper security measures.
2. To be able to describe the purpose of encryption and the function of public and private keys.
3. To be able to compare and contrast the various types of firewalls.
4. To generate and distribute a PGP key pair and use the PGP package to send an encrypted e-mail message.
5. To be able to explain the concept of and necessity for transport layer security.

**NWK6. Performance Evaluation [elective]**

*Suggested time: 12 hours*

**Topics:**
- Privacy and Public Networks
Virtual Private Networks
Service Paradigms
  Connection-Oriented Service
  Connectionless Service
Network Performance Characteristics
  Delay, Throughput

Learning Objectives:
1. To be able to define performance metrics and describe how each affects a particular network and/or service paradigm.

NWK7. Data Communications [elective]
Suggested time: 12 hours
Topics:
  Signals (Analog, Digital, Periodic, A-periodic)
  Time and Frequency Domains
  Encoding and Modulating
    A/D and D/A Conversion
  Interfaces and Modems
  Transmission Media
  Multiplexing
  Error Detection and Correction

Learning Objectives
1. To demonstrate understanding of the fundamental concepts of data communications, from signals and signal encoding methods to communication service methods and data transmission modes.

NWK8. Wireless and mobile computing [elective]
Suggested time: 6 hours

Topics:
  The special problems of wireless and mobile computing
  Wireless local area networks and satellite-based networks
  Mobile Internet protocol
  Extending the client-server model to accommodate mobility
  Mobile data access: server data dissemination and client cache management
  The software packages to support mobile and wireless computing
  The role of middleware and support tools
  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. Be aware of the many areas of interest that lie within this area, including networking, multimedia, wireless and mobile computing, and distributed computing.
Electronics (ELE)

- ELE0. History and overview of electronics [core]
- ELE1. Electronic properties of materials [core]
- ELE2. Diodes and diode circuits [core]
- ELE3. MOS transistors and biasing [core]
- ELE4. MOS logic families [core]
- ELE5. Bipolar transistors and logic families [core]
- ELE6. Design parameters and issues [core]
- ELE7. Storage elements [core]
- ELE8. Interfacing logic families and standard buses [core]
- ELE9. Operational amplifiers [core]
- ELE10. Data conversion circuits [core]
- ELE11. SPICE circuit simulation [core]
- ELE12. Electronic voltage and current sources [elective]
- ELE13. Linear amplification and biasing [elective]
- ELE15. Multistage transistor amplifiers [elective]
- ELE16. Power circuits [elective]
- ELE17. Feedback in electronics [elective]
- ELE18. Active filters [elective]
- ELE19. Integrated circuit building blocks [elective]
- ELE20. Circuits for wireless applications [elective]

Electronics is foundational material for computer engineering. These areas provide the basic knowledge for the design of the electronic circuits used to implement computers. This is a very broad area and it is expected that there will be a great deal of variation between programs in the coverage of topics outside of the core.

**ELE0. History and overview of electronics [core]**

*Suggested time: 1 hour*

**Topics:**

- Knowledge themes include diodes, transistors, propagation, buses, amplifiers, and simulators
- Contributors to the subject include Bardeen, Brattain, and Shockley
- Purpose and role of electronics in computer engineering

**Learning objectives:**

- Associate the themes involved with electronics.
- Identify contributors to the subject area.
- Describe how computer engineering could make use of electronics.
ELE1. Electronic properties of materials [core]

Suggested time: 4 hours

Topics:
- Solid-state materials
- Electrons and holes
- Doping, acceptors and donors
- p- and n-type material
- Conductivity and resistivity
- Drift and diffusion currents, mobility and diffusivity

ELE2. Diodes and diode circuits [core]

Suggested time: 5 hours

Topics:
- Diode operation and i-v characteristics
- Regions of operation, models, and limitations
- Schottky, Zener, variable capacitance diodes
- Single diode circuits, the load line
- Multi-diode circuits
- Rectifiers
- dc/dc converters
- Diode logic – AND and OR functions

ELE3. MOS Transistors and biasing [core]

Suggested time: 3 hours

Topics:
- NMOS field-effect transistor operation
- i-v characteristics
- Regions of operation, models, and limitations
- Enhancement and depletion-mode devices
- PMOS devices
- Transfer characteristic of FET with load resistor
- Biasing for logic and amplifier applications

ELE4. MOS logic families [core]

Suggested time: 9 hours

Topics:
- Logic level definitions
- NMOS logic design – Inverter, NOR, NAND, SOP, POS, complex gates
- PMOS logic
- CMOS logic - Inverter, NOR, NAND, SOP, POS, complex gates
- Dynamic logic
- CVS logic
Cascade buffers
NMOS and CMOS power/delay scaling

**ELE5. Bipolar transistors and logic families [core]**

*Suggested time: 5 hours*

*Topics:*
- npn and pnp transistor operation
- i-v characteristics
- Regions of operation, models, and limitation
- Transfer characteristic of BJT with load resistor
- Biasing for logic and amplifier applications
- Logic level definitions
- The differential pair as a current switch
- Transistor-transistor logic – inverters, NAND, other functions
- Emitter-coupled logic – OR/NOR gate, other functions
- Low voltage bipolar logic families

**ELE6. Design parameters and issues [core]**

*Suggested time: 5 hours*

*Topics:*
- Switching energy, power-delay product comparison,
- Propagation delay, rise time, fall time
- Fan-in and fan-out
- Power dissipation, noise margin
- Power supply distribution
- Sources of signal coupling and degradation
- Transmission line effects; passive, active, dc and ac termination
- Element tolerances
- Worst-case analysis of circuits
- Monte Carlo analysis
- Monte Carlo analysis in SPICE
- Six-sigma design

**ELE7. Storage elements [core]**

*Suggested time: 3 hours*

*Topics:*
- Latches
- Flip-flops
- Static RAM cells
- Dynamic RAM cells
- Sense amplifiers
ELE8. Interfacing logic families and standard buses [core]

*Suggested time: 3 hours*

**Topics:**
- Terminal characteristics of various logic families
- Standard interface characteristics
- Level translations – TTL/CMOS, TTL/ECL, CMOS/ECL, etc.
- Single-ended to differential and differential to single-ended conversion
- Transmission line characteristics, reflections
- Bus termination – passive, active, dc, ac
- 4-20 mA current interfaces
- RS-XXX buses
- IEEE-XXXXX buses
- Low-level differential signaling
- RAMBUS
- DDR

ELE9. Operational amplifiers [core]

*Suggested time: 6 hours*

**Topics:**
- Ideal op-amps and circuit analysis
- Ideal op-amp circuits – inverting and non-inverting amplifiers, summing amplifier, difference amplifier, integrator, low pass filter
- Non-ideal op-amps – dc errors, CMRR, input and output resistances, frequency response, output voltage and current limitations
- Circuits with non-ideal amplifiers
- Multi-stage op-amp circuits

ELE10. Data conversion circuits [elective]

*Suggested time: 5 hours*

**Topics:**

**D/A Converters**
- Definitions – Codes, LSB, MSB, etc
- Linearity, differential linearity, offset and gain errors
- Weighted resistor D/A converter
- R/2R ladders and D/A converters
- Weighted current source converters
- Delta-sigma converters

**A/D Converters**
- Definitions – Codes, LSB, MSB, missing etc
- Linearity, differential linearity, offset and gain errors, missing codes
- Counting converter
- Successive approximation
Single and dual slope converters
Flash converters
Delta-sigma converters
Sample-and-hold circuits

**ELE11. SPICE circuit modeling and simulation [core]**

*Suggested time:* 3 hours

*Topics:*
- Dc analysis
- Ac analysis
- Transient analysis
- Simulation control options
- Built-in solid-state device models
- Device parameter control
- Libraries
- Mixed-mode simulation

**ELE12. Electronic voltage and current sources [elective]**

*Suggested time:* 3 hours

*Topics:*
- Electronic voltage sources
  - Ideal voltage source characteristics
  - Voltage references
  - Emitter followers
  - Voltage sources utilizing operational amplifiers
- Electronic current sources
  - Ideal current source characteristics
  - Transistor current sources; Common-emitter, cascode, regulated cascode circuits
  - Current sources utilizing operational amplifiers

**ELE13. Linear amplification and biasing [elective]**

*Suggested time:* 2 hours

*Topics:*
- Characteristics of a linear amplifier
- Voltage gain, current gain, power gain, dB scale
- Frequency domain characteristics
- Distortion
- Biasing for linear amplification

*Suggested time:* 12 hours

**Topics:**
- Definition of a small-signal in a diode
- Definition of a small-signal in the BJT
- BJT amplifiers: common-emitter, common-base, common-collector
- Definition of a small-signal in the MOSFET
- MOSFET amplifiers: common-source, common-gate, common-drain
- Bias circuits; voltage, current, power gain; input and output resistance; signal range
- Coupling/dc blocking/bypass capacitors
- Low frequency response
- High frequency device models
- High Frequency response
- Short-circuit and open-circuit time constant techniques

ELE15. Multistage transistor amplifiers

*Suggested time:* 6 hours

**Topics:**
- Ac and dc coupled amplifiers
- Frequency response
- Differential pairs – MOSFET & BJT
- Current sources and biasing
- Current mirrors
- Active loads
- Elementary two- and three-stage op-amp circuits
- Classical op-amp input stages
- μA741 op-amp

ELE16. Power circuits [elective]

*Suggested time:* 3 hours

**Topics:**
- Class A output stages
- Class B and class B push-pull output stages
- Cross over distortion
- Class AB amplifiers
- Power semiconductor devices
- Switching (boost and buck) converters
ELE17. Feedback in electronic circuits [elective]

Suggested time: 5 hours

Topics:
- Feedback theory
- Nyquist plots and stability
- Bode plots and stability
- Feedback in op-amp circuits
- Two-port feedback theory
- Transistor feedback amplifiers
- Simulation of feedback amplifier loop-gain

ELE18. Active filters [elective]

Suggested time: 4 hours

Topics:
- Continuous time filters
  - Bandwidth, Q, etc.
- Single op-amp active filters
- Multi op-amp filters
  - Q and cutoff/center frequency sensitivity
- Switched capacitor filters

ELE19. Integrated circuit building blocks

Suggested time: 8 hours

Topics:
- Oscillators:
  - Barkhausen criteria for oscillation
  - RC oscillators
  - LC oscillators – Colpitts, Hartley
  - Crystal oscillators
  - Multivibrators
- Operational amplifiers and circuits
- Comparators
- PTAT circuits
- Band-gap references
- Voltage regulators
- Gilbert multipliers
ELE20. Circuits for wireless applications [elective]

*Suggested time:* 9 hours

*Topics:*
- Noise
- Passive components
- Low noise amplifiers
- Frequency conversion and mixers
- Power amplifiers – Class B, Class C
Digital Logic (DIG)

DIG0. History and overview of digital logic [core]
DIG1. Switching theory [core]
DIG2. Combinational logic circuits [core]
DIG3. Modular design of combinational circuits [core]
DIG4. Memory elements [core]
DIG5. Sequential logic circuits [core]
DIG6. Register Transfer Logic [core]
DIG7. Digital Systems Design [core]

The logic design area covers the digital building blocks, tools, and techniques in digital design. Emphasis is on a building-block approach. This subject area comprises approximately 3 semester hours.

DIG0. History and overview of digital logic [core]

Suggested time: 1 hour

Topics:
- Knowledge themes include logic circuits, switching, memory, registers, and digital systems
- Contributors to the subject include Boole, Kleene, Burkes, Goldstine, and von Neumann
- Purpose and role of digital logic in computer engineering

Learning objectives:
- Associate the themes involved with digital logic.
- Identify contributors to the subject area.

Describe how computer engineering could make use of digital logic.

DIG1. Switching theory [core]

Suggested time: 6 hours

Topics:
- Number systems and codes
- Binary arithmetic
- Boolean and switching algebra
- Representation and manipulation of switching functions
- Minimization of switching functions
- Incompletely specified switching functions

Learning objectives:
1. To be able to work with binary number systems and arithmetic.
2. To be able to derive and manipulate switching functions, which form the basis of digital circuits.
3. To be able to reduce switching functions to simplify circuits used to realize them.
DIG2. Combinational logic circuits [core]

Suggested time: 4 hours

Topics:
- Basic logic gates (AND, OR, NOT, NAND, NOR, XOR)
- Realization of switching functions with networks of logic gates
  - 2-level networks: AND-OR, OR-AND, NAND-NAND, NOR-NOR
  - Multi-level networks
- Physical properties of logic gates (technology, fan-in, fan-out, propagation delay)
- Elimination of timing hazards/glitches

Learning Objectives:
1. To be able to realize switching functions with networks of logic gates.
2. To be able to explain and apply fundamental characteristics of relevant electronic technologies, such as propagation delay, fan-in, fan-out, and power dissipation and noise margin.

DIG3. Modular design of combinational circuits [core]

Suggested time: 6 hours

Topics:
- Design of medium scale combinational logic modules
  - Multiplexers, demultiplexers, decoders, encoders, comparators
  - Arithmetic functions (adders, subtracters, carry lookahead)
  - Multipliers, dividers
  - Arithmetic and logic units (ALUs)
- Hierarchical design of combinational circuits using logic modules

Learning Objectives:
1. To be able to analyze and explain uses of small- and medium-scale logic functions as building blocks.
2. To be able to analyze and design combinational logic networks in a hierarchical, modular approach, using standard and custom logic functions.

DIG4. Memory Elements

Suggested time: 3 hours

Topics:
- Unclocked and clocked memory devices (latches, flip flops)
- Level vs. edge-sensitive, and master-slave devices
- Basic flip flops (SR, D, JK, T)
- Asynchronous flip flop inputs (preset, clear)
- Timing constraints (setup time, hold time) and propagation delays
- Data registers (selection, clocking, timing)
- Random-access memory (RAM)
Learning Objectives:
1. To be able to design and describe the operation of basic memory elements.
2. To be able to analyze circuits containing basic memory elements.
3. To apply the concepts of basic timing issues, including clocking, timing constraints, and propagation delays during the design process.

DIG5. Sequential Logic Circuits
Suggested time: 12 hours

Topics:
- Finite state machines (FSMs) - clocked and unclocked
- Mealy vs. Moore models of FSMs
- Modeling FSM behavior
  - State diagrams and state tables
  - Timing diagrams
  - Algorithmic state machine charts
- Analysis of synchronous and asynchronous circuits
- Design of synchronous sequential circuits
  - State minimization
  - State assignment
  - Next state and output equation realization
- Sequential functional units
  - Data registers
  - Shift registers
  - Counters
  - Sequence detectors
  - Synchronizers
  - Debouncers
  - Controllers

Learning Objectives:
1. To be able to analyze the behavior of synchronous and asynchronous machines
2. To be able to synthesize synchronous and asynchronous sequential machines

DIG6. Register Transfer Logic [core]
Suggested time: 6 hours

Topics:
- Register transfer notation
  - data flow components
  - control flow components
- Conditional and unconditional transfers
- ALU control
- Bus structures
Logic sequencers

Learning Objectives:
1. To introduce the notion of register transfer as a higher-level way to describe system behavior.

DIG7. Digital Systems Design [core]
Suggested time: 12 hours

Topics:
- Hierarchical, modular design of digital systems
- Digital circuit modeling
  - Block diagrams
  - Timing diagrams
  - Register transfer languages
  - Algorithmic state machines
  - HDL (VHDL, Verilog)
- Simulation of digital circuit models (HDL, schematic, etc.)
- Synthesis of digital circuits from HDL models
- Design principles and techniques
  - Bridging conceptual levels – top down/bottom up
  - Divide and conquer
  - Iteration
  - Satisfying a behavior with a digital structure
- Functional units, building blocks and LSI components
  - Bus, adder, shifter, register, ALU, and control circuits
  - Tristate devices and buses
- Clock generation and distribution
- Control concepts
  - Major control state
  - Sequence of microoperations
  - Event schematic for macrosequences
  - Conditional execution of microoperations
- Timing concepts
  - System timing dependencies
  - Sequencing
  - Clock skew
- Programmable logic devices (PLDs) and field-programmable gate arrays (FPGAs)
  - PLAs, ROMs, PALs, complex PLDs
- Digital-to-analog and analog-to-digital conversion
- Testing

Learning Objectives:
1. To be able to apply digital system design principles and descriptive techniques.
2. To be able to analyze and design functional building blocks and control and timing concepts of digital systems.
Computing Curricula - Computer Engineering Body of Knowledge

Strawman Draft (06/12/02)

**Programming Fundamentals (PRF)**

- PRF0. History and overview of programming fundamentals [core]
- PRF1. Fundamental programming constructs [core]
- PRF2. Algorithms and problem-solving [core]
- PRF3. Fundamental data structures [core]
- PRF4. Programming Paradigms [core]
- PRF5: Recursion [core]
- PRF6. Object-oriented programming
- PRF7. Event-driven and concurrent programming
- PRF8. Using APIs

Competency in a programming language is prerequisite to the study of computer engineering. Undergraduate programs must teach students how to use at least one programming language. The difficulty of achieving the necessary level of fluency in a programming language is further complicated by the need to include more advanced techniques than were necessary a few years ago. The core topics in this unit cover the basic pieces that should be covered independent of a particular programming language as programming languages tend to come and go over the years.

Over the past decade, object-oriented programming, event-driven applications, and the use of extensive APIs (application programming interfaces) have become fundamental tools that some computer engineering students need early in their academic program. These concepts may be included in a program that only teaches an object oriented language such as C++ or Java.

**PRF0. History and overview of programming fundamentals [core]**

*Suggested time: 1 hour*

*Topics:*
- Knowledge themes include programming constructs and paradigms, data structures, problem solving, and recursion
- Contributors to the subject include Backus, McCarthy, Dahl, Myrhaag, Kay
- Contrasts between different programming paradigms
- Purpose and role of programming fundamentals in computer engineering

*Learning objectives:*
1. Associate the themes involved with programming fundamentals.
2. Identify contributors to the subject area.
3. Articulate differences between various programming paradigms.
4. Describe how computer engineering could make use of programming fundamentals.

**PRF1. Fundamental programming constructs [core]**

*Suggested 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

Learning objectives:
1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.
2. Write 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 procedures and functions.
3. Apply the techniques of structured decomposition to break a program into smaller pieces.

PRF2. Algorithms and problem-solving [core]
Suggested 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. Define the basic properties of an algorithm.
2. Develop algorithms for solving simple problems.
3. Use a suitable programming language to implement, test, and debug algorithms for solving simple problems.

PRF3. Fundamental data structures [core]
Suggested 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. Identify data structures that could be used to represent specific types of information and discuss the tradeoffs among the different possibilities.
2. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables.
3. Describe how these data structures are allocated and represented in memory.

PRF4. Programming Paradigms [core]
Suggested time: 8 hours

Topics:
- Procedural programming
- Functional programming
- Object-oriented design
- Encapsulation and information-hiding
- Separation of behavior and implementation
- Classes, subclasses, and inheritance
- Event-Driven programming

Learning objectives:
1. Be able to identify the paradigm used by pseudo-code snippets.
2. Be able to identify the appropriate paradigm for a given programming problem.

PRF5. Recursion [core]
Suggested time: 6 hours

Topics:
- The concept of recursion
- Recursive mathematical functions
- Simple recursive procedures
- Divide-and-conquer strategies
- Recursive backtracking
- Implementation of recursion

Learning objectives:
1. Explain the concept of recursion.
2. Explain the structure of the divide-and-conquer approach.
3. Write, test, and debug simple recursive functions and procedures.
4. Describe how recursion can be implemented using a stack.
PRF6. Object-oriented programming

Topics:
- Polymorphism
- Class hierarchies
- Collection classes and iteration protocols
- Fundamental design patterns

Learning objectives:
1. Outline the philosophy of object-oriented design and the concepts of encapsulation, subclassing, inheritance, and polymorphism.
2. Design, code, test, and debug simple programs in an object-oriented programming language.
3. Select and apply appropriate design patterns in the construction of an object-oriented application.

PRF7. Event-driven and concurrent programming

Topics:
- Event-handling methods
- Event propagation
- Managing concurrency in event handling
- Exception handling

Learning objectives:
1. Design, code, test, and debug simple event-driven programs that respond to user events.
2. Defend the need for concurrency control and describe at least one method for implementing it.
3. Develop code that responds to exception conditions raised during execution.

PRF8. Using APIs

Topics:
- API programming
- Class browsers and related tools
- Programming by example
- Debugging in the API environment
- Component-based computing
- Middleware
Learning objectives:
1. Explain the value of application programming interfaces (APIs) in software development.
2. Design, write, test, and debug programs that use large-scale API packages.
Algorithms and Complexity (ALG)

ALG0. History and overview of algorithms and complexity [core]
ALG1. Basic algorithmic analysis [core]
ALG2. Algorithmic strategies [core]
ALG3. Fundamental computing algorithms [core]
ALG4. Distributed algorithms [core]
ALG5. Basic computability theory [core]
ALG6. The complexity classes P and NP [elective]

Algorithms are fundamental to computer engineering. The real-world performance of any software or hardware system depends on 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 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, 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.

ALG0. History and overview of algorithms and complexity [core]

*Suggested time:* 1 hour

*Topics:*
- Knowledge themes include analysis and complexity, algorithmic strategies, and basic known algorithms
- Contributors to the subject include Euclid, Harmanis, Hoare, Knuth, and Dijkstra
- Contrast complexities of different algorithmic strategies
- Purpose and role of algorithms in computer engineering

*Learning objectives:*
- Associate the themes involved with algorithms and complexity.
- Identify contributors to the subject area.
- Describe how computer engineering could make use of algorithms and complexity.

ALG1. Basic algorithmic analysis [core]

*Suggested 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
Empirical measurements of performance
Time and space tradeoffs in algorithms
Using recurrence relations to analyze recursive algorithms

Learning objectives:
1. Use big O, omega, and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.
2. Determine the time complexity of simple algorithms.
3. Deduce the recurrence relations that describe the time complexity of recursively defined algorithms, and solve simple recurrence relations.

ALG2. Algorithmic strategies [core]
Suggested time: 8 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. Design algorithms using the brute-force, greedy, and divide-and-conquer strategies.
2. Design algorithms using at least one other algorithmic strategy from the list of topics for this unit.

ALG3. Fundamental computing algorithms [core]
Suggested 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. Use and implement the fundamental abstract data types—specifically including hash tables, binary search trees, and graphs—necessary to solve algorithmic problems efficiently.
2. Solve problems using sequential search, binary search, \(O(N \log N)\) sorting algorithms, and 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.
3. Demonstrate the following abilities: to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in simple programming contexts.

ALG4. Distributed algorithms [core]

Suggested time: 3 hours

Topics:
- Concurrency
- Scheduling
- Fault tolerance

Learning objectives:
1. Explain the distributed paradigm.
2. Distinguish between logical and physical clocks.
3. Describe the relative ordering of events.
4. Explain one simple distributed algorithm.

ALG5. Basic computability theory [core]

Suggested time: 6 hours

Topics:
- Finite-state machines
- Deterministic finite Automata (DFA)
- Non-deterministic finite Automata (NFA)
- Equivalence of DFA’s and NFA’s
- Context-free grammars
- Pushdown automata (PDA)
- Tractable and intractable problems
- Uncomputable functions
- The halting problem
Implications of uncomputability

**Learning objectives:**
1. Explain the idea that some problems may have no algorithmic solution.
2. Provide examples that illustrate the implications of uncomputability.

**ALG6. The complexity classes P and NP [elective]**

*Suggested time: 6 hours*

**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.
Discrete Structures (DSC)

DSC0. History and overview of discrete structures
DSC1. Functions, relations, and sets [core]
DSC2. Basic logic [core]
DSC3. Proof techniques [core]
DSC4. Basics of counting [core]
DSC5. Graphs and trees [core]
DSC6. Discrete probability [core]
DSC7. Recursion [elective]

Discrete structures is foundational material for computer engineering. By *foundational* we mean that relatively few computer engineers will be working primarily on discrete structures, but that many other areas of computer engineering require the ability to work with concepts from discrete structures. Discrete structures includes important material from such areas as set theory, logic, graph theory, combinatorics, and probability. The material in discrete structures is pervasive in the areas of data structures and algorithms. As the field of computer engineering 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.

It is important to remember that discrete structures must be placed in the context of computer engineering. Wherever possible, reference should be made to engineering situations or settings. Topics in discrete structures should be integrated with themes from computer engineering. It is important to emphasize application rather than just theory. That is, teachers should refrain from teaching discrete structure topics strictly from a mathematical perspective and should instill relevance to application areas of computing and computer engineering.

Finally, we note that while areas often have somewhat fuzzy boundaries, this is especially true for discrete structures. We have gathered together a body of material of a mathematical nature that must be included in a computer engineering education and that computer engineering educators know well enough to specify in great detail. However, the decision about where to draw the line between particular knowledge areas on the one hand, and topics left only as supporting mathematics on the other, 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.

DSC0. History and overview of discrete structures [core]

*Suggested time:* 1 hour

*Topics:*

- Knowledge themes include sets, logic, functions, and graphs
Contributors to the subject: Aristotle, Euler, Boole, Venn, and DeMorgan
Purpose and role of discrete structures in computer engineering
Contrasts between discrete-time models vs. continuous-time models

Learning objectives:
- Associate the themes involved with discrete structures.
- Identify contributors to the subject area.
- Articulate differences between discrete and continuous models.
- Describe how computer engineering could make use of discrete structures.

DSC1. Functions, relations, and sets [core]
Suggested time: 6 hours

Topics:
- Functions (one-to-one, onto, inverses, composition)
- Relations (reflexivity, symmetry, transitivity, equivalence relations)
- Discrete versus continuous functions and relations
- Sets (Venn diagrams, complements, Cartesian products, power sets)
- Pigeonhole principle
- Cardinality and countability
- Applications of functions and relations to computer engineering

Learning objectives:
- Illustrate by examples the basic terminology of functions, relations, and sets.
- Illustrate by examples, both discrete and continuous, the operations associated with sets, functions, and relations.
- Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
- Apply functions and relations to problems in a computer engineering setting.

DSC2. Basic logic [core]
Suggested time: 10 hours

Topics:
- Propositional logic
- Logical connectives
- Truth tables
- Use of logic gates to illustrate connectives
- Normal forms (conjunctive and disjunctive)
- Predicate logic
- Universal and existential quantification
- Limitations of predicate logic
- Boolean algebra
- Applications of logic to computer engineering
Learning objectives:
- Manipulate formal methods of symbolic propositional and predicate logic.
- Use formal tools of symbolic logic.
- Demonstrate knowledge of formal logic proofs and logical reasoning through solving problems such as puzzles.
- Apply logic gates to problems in logic.

DSC3. Proof techniques [core]
Suggested time: 6 hours

Topics:
- Notions of implication, converse, inverse, negation, and contradiction
- The structure of formal proofs
- Direct proofs
- Proof by counterexample, contraposition, and contradiction
- Mathematical induction and strong induction
- Examples of proof in a computer engineering setting

Learning objectives:
- Outline basic proofs for theorems using the techniques of proof by contradiction and mathematical induction.
- Solve problems by different methods of proof
- Apply proof techniques to problems in a computer engineering setting.

DSC4. Basics of counting [core]
Suggested time: 3 hours

Topics:
- Counting arguments
- The pigeonhole principle
- Permutations and combinations
- Applications of counting to computer engineering

Learning objectives:
- Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
- State the definition of the Master theorem.
- Apply counting principles to problems in a computer engineering setting.

DSC5. Graphs and trees [core]
Suggested time: 4 hours
Topics:
- Trees
- Undirected graphs
- Directed graphs
- Spanning trees
- Traversal strategies
- Applications of graphs to computer engineering

Learning objectives:
- Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each.
- Model problems in computing using graphs and trees.
- Relate graphs and trees to data structures, algorithms, and counting.
- Apply graphs and trees to problems in a computer engineering setting.

DSC6. Discrete probability [core]
Suggested time: 6 hours

Topics:
- Finite probability space, probability measure, events
- Conditional probability, independence, Bayes’ rule
- Integer random variables, expectation
- Applications of probability to computer engineering

Learning objectives:
- Calculate probabilities of events and expectations of random variables for problems arising from games of chance.
- Demonstrate how to apply the tools of probability to average case analysis of simple algorithms.
- Apply probability to problems in a computer engineering setting.

DSC7. Recursion [elective]
Suggested time: 6 hours

Topics:
- Recursive mathematical definitions
- Developing recursive equations
- Solving recursive equations
- Applications of recursion to computer engineering

Learning objectives:
- Solve a variety of basic recursive equations
- Analyze a problem to create relevant recurrence equations or to identify important counting questions.
Relate the ideas of mathematical induction to recursion and recursively defined structures.

Apply recursion to problems in a computer engineering setting.
VLSI and ASIC Design (VLS)

VLS1. MOS Transistor Fundamentals
VLS2. Processing and Layout
VLS3. Function of the Basic Inverter Structure
VLS4. Circuit Characterization and Performance
VLS5. Combinational Logic Circuits
VLS6. Sequential Logic Circuits
VLS7. Alternative Circuit Structures/Low Power Design
VLS8. Semiconductor Memories and Array Structures
VLS9. Chip Input/Output Circuits
VLS10. Semi custom Design Technologies
VLS11. ASIC Design Methodology

MOS transistor fundamentals, CMOS logic circuits; VLSI fabrication and design rules; clocking strategies and sequential design; performance estimation; memories and programmable arrays; standard cell design methodologies; computer aided design (CAD) tools, top-down design of application-specific integrated circuits (ASICs).

VLS1. MOS Transistor Fundamentals

*Suggested time:* 6 hours

*Topics:*
- Semiconductor materials and characteristics
- Semiconductor doping and its effects
- Diode construction, function, and modeling
- MOS transistor characteristics and equations
- Sub-micro MOS transistor characteristics
- MOS transistor parasitic
- Device modeling and simulation
- Bipolar devices (optional)

*Learning Objectives:*
1. Understand the current carrying mechanism and I/V characteristics of intrinsic and doped semiconductor materials.
2. Understand the behavior and I/V characteristics of a reverse-biased and forward-biased PN junction.
3. Understand the function of a PMOS and NMOS field effect transistor (FET) and how to model that function using the device equations.
4. Understand the effect of sub-micron device sizes on the function of MOSFETs.
5. Understand the origin and effect of parasitic resistances and capacitances within the transistor itself.
6. Understand the basics of how MOSFET simulation is performed by SPICE (and optionally BSIM) and what parasitic are included in the device model itself.
7. (Optionally) Understand the function of a Bipolar Junction Transistor (BJT) and how to model that function using the device equations.

VLS2. Processing and Layout
Suggested time: 2 hours
Topics:
- Processing steps for patterning SiO₂ on a silicon wafer
- CMOS processing technology steps and their results
- Layout design rules and their objectives
- Scalable (λ-based) design rules
- Design-rule checking

Learning Objectives:
1. Understand the basic steps of photolithography, its limitations, and how that determines minimum line width and device sizes.
2. Understand the processing steps required for fabrication of CMOS devices and the general results of each step.
3. Understand the physical defects that can arise in silicon processing and how design rules attempt to minimize their effects.
4. Understand the spacing and minimum device sizes specified by a typical set of design rules.
5. Understand the benefits and tradeoffs of a λ-based scalable design rule.
6. Understand the process and tools used for design rule checking.

VLS3. Function of the Basic Inverter Structure
Suggested time: 5 hours
Topics:
- Connectivity, layout, and basic functionality of a CMOS inverter
- The CMOS inverter voltage transfer characteristic (VTC)
- Analysis of the CMOS VTC for switching threshold, \( V_{OH}, V_{OL}, V_{IH}, V_{IL} \), and Noise Margins
- Effect of changing the inverter configuration on the CMOS VTC
- Connectivity and basic functionality of a Bipolar ECL inverter (optional)
- Connectivity and basic functionality of a Bipolar TTL inverter (optional)

Learning Objectives:
1. Understand the basic functionality of the CMOS inverter.
2. Understand how the VTC of a CMOS inverter is derived from the PMOS and NMOS characteristic \( I_D \) vs. \( V_{DS} \) family of curves.
3. Be able to analyze the VTC to determine switching threshold, $V_{OH}$, $V_{OL}$, $V_{IH}$, $V_{IL}$, and Noise Margins.
4. Understand how these quantities reflect the ability of the inverter to operate in the presence of noise.
5. Understand how changing the configuration of the inverter and the MOSFETS that make it up changes the VTC and thus the inverter’s operation.
6. (Optionally) Understand the functionality of Bipolar-based logic gates

**VLS4. Circuit Characterization and Performance**

*Suggested time: 3 hours*

*Topics:*
- Switching characteristics (rise and fall times, gate delays)
- Power dissipation
- Resistance and capacitance estimation
- CMOS transistor sizing
- Conductor sizing

*Learning Objectives:*
1. Understand the basic causes of propagation delay and power dissipation in CMOS logic.
2. Understand the techniques for estimating parasitic resistance and capacitance for various layers on a CMOS integrated circuit.
3. Understand the effects of changing (and optimizing) the transistor widths in CMOS logic.
4. Understand the effects of changing (and optimizing) the conductor widths on a CMOS integrated circuit.

**VLS5. Combinational Logic Circuits**

*Suggested time: 4 hours*

*Topics:*
- Basic CMOS gate design
- Layout techniques for combinational logic structures
- Transistor sizing for complex CMOS logic devices
- Transmission gates
- Architectural building blocks (multiplexers, decoders, adders, counters, multipliers)

*Learning Objectives:*
1. Understand how the circuit design for CMOS logic gates is performed.
2. Understand the techniques, such as Euler paths and stick diagrams, which are used to optimize the layout of CMOS logic circuits.
3. Understand how the size for each transistor in a CMOS logic gate can be determined.
4. Understand the functionality of the CMOS transmission gate and how it is used in several logic functions (multiplexers, transmission gate-based XOR gates, etc.)
5. Understand the functionality of several of the more important architectural building blocks identified above and how they can be optimized for CMOS implementation.
VLS6. Sequential Logic Circuits

*Suggested time: 5 hours*

**Topics:**
- Storage mechanisms in CMOS logic
- Dynamic latch circuits
- Static latch and flip-flop circuits
- Sequential circuit design
- Single and multiphase clocking
- Clock distribution, clock skew

**Learning Objectives:**
1. Understand how charge storage (capacitance) and feedback can be used to store values in CMOS logic.
2. Understand the circuit design, functionality and advantages and disadvantages of dynamic latches in CMOS.
3. Understand the circuit design, functionality and advantages and disadvantages of static latches and flip-flops (including edge-triggered) in CMOS.
4. Understand the concepts of bi-stability and metastability in static flip-flops.
5. Understand how latches and flip-flops are used in the design of state machines and data paths.
6. Understand the functionality and advantages and disadvantages of single phase clocking, both level sensitive and edge triggered.
7. Understand the functionality and advantages and disadvantages of multi (two) phase clocking.
8. Understand the problems arising from clock skew and how clock distribution schemes (including the use of PLLs) can be used to solve it.

VLS7. Alternative Circuit Structures/Low Power Design

*Suggested time: 3 hours*

**Topics:**
- NMOS, Pseudo-NMOS, Domino-CMOS, CVSL
- Low power design

**Learning Objectives:**
1. Understand how MOSFET-based logic families other than CMOS are implemented.
2. Understand the advantages and disadvantages of these logic families.
3. Understand the reasons for dynamic and static leakage power.
4. Understand how to design CMOS circuits for low power.
VLS8. Semiconductor Memories and Array Structures

*Suggested time: 4 hours*

*Topics:*
- Memory system organization
- Read-only memory circuits
- EPROM/EEPROM/Flash memory circuits
- Static read-write memory (SRAM) circuits
- Dynamic read-write memory (DRAM) circuits
- Programmable Logic Array (PLA) circuits

*Learning Objectives:*
1. Understand how memory systems are organized and why they are not typically organized in the most simplistic arrangement – that of a one-dimensional word array.
2. Understand the circuit-level implementations possible for read-only memory (ROM) organizations.
3. Understand the layout and function of the specialized transistors used in non-volatile ROM devices and how their characteristics influence the circuit-level implementations of ROMs using them.
4. Understand the functionality and layout of cells used to implement static RAM (SRAM) memories.
5. Understand how SRAMs are typically organized and how their associated peripheral circuitry (sense amps, decoders, address translation detectors, etc.) is organized and functions.
6. Understand how a typical 3-transistor and 1-transistor DRAM cell functions and how they are typically laid out.
7. Understand how DRAMs are typically organized and accessed, and how their associated peripheral circuitry (sense amps, decoders, etc.) is organized and functions.
8. Understand how PLAs function, how they can be implemented in CMOS, and how logic functions are mapped to them.

VLS9. Chip Input/Output Circuits

*Suggested time: 2 hours*

*Topics:*
- General I/O pad issues
- Bonding pads
- ESD Protection circuits
- Input, Output, Bidirectional, and analog pads
- VDD and VSS pads

*Learning Objectives:*
1. Understand the unique functions that I/O circuits must perform and their general circuit-level implementations.
2. Understand the functions of signal I/O pads and their general transistor-level implementations.
3. Understand the functions of VDD and VSS pads for both the core and padframe, and their general transistor-level implementations.

VLS10. Semi custom Design Technologies

Suggested time: 2 hours

Topics:
- Full custom methodology
- Standard cell methodology
- Gate array technologies
- Programmable logic technologies
- Field-programmable gate arrays (FPGAs)
- Time to market and design economics

Learning Objectives:
1. Understand the different design techniques, methodologies, and implementation technologies available to implement a function on a single integrated circuit.
2. Understand the advantages and disadvantages of each and how a designer might go about selecting a specific one for his or her current project.

VLS11. ASIC Design Methodology

Suggested time: 4 hours

Topics:
- ASIC design flow (custom, semicustom)
- Design hierarchy
- Computer-aided design (CAD)
  - Design modeling and capture (schematic, HDL)
  - Design verification (formal, simulation, timing analysis)
  - Automated synthesis
  - Layout, floorplanning, place and route
  - Back annotation
- Semi-custom design with programmable logic devices and programmable gate arrays
- System-on-chip (SOC) design and intellectual property (IP) cores
- Testing and design for testability
- Verification

Learning Objectives:
1. Understand the more detailed design issues present in implementing a given digital system on an Application-Specific Integrated Circuit (ASIC)
2. Understand the function, capabilities, and disadvantages of the various Computer-Aided Design (CAD) tools available to the ASIC designer to automate portions of the design process.
3. Understand the issues that come with implementing a real-world, complex design in an ASIC for a production environment.
4. Understand the basic principles of test generation and design for testability.
5. Understand the difference between testing and verification.
Inevitably the pursuit of enhanced computing capability has resulted in a wide range of developments in the sphere of computing. Applications that benefit from such developments include weather forecasting, aerodynamic modeling, aspects of artificial intelligence, computer animation and many more. Some of these advances have focused on improving the performance of single processors, but many have been associated with advances in parallel computing. These hardware developments have been accompanied by complementary advances in the associated software as well as in the theoretical insights and understanding of these matters. These have included providing clear definitions of the semantics of constructs, identifying limitations on particular approaches, and so on.

For this knowledge unit within the context of Computer Engineering, the main focus will be three fold: firstly there are architectural considerations which hardware designers need to debate and address; secondly there is the nature of the basic software systems which provide a user with the means of expressing parallel algorithms in a manner that utilizes and exploits the hardware capability. Of course, these two aspects are intimately related. But thirdly, it is important to have a sound basis for measuring performance and so comparing the effectiveness of different machines and machine architectures. The examples presented under the various headings are those that are common now, but in the future these topics will be replaced by those that have become available.

ACP1. Overview/History [core]
Suggested time: 3 hours

Topics:

Brief history of the development of high performance computers: the trends, relationships between speed, store size and other machine parameters.

Review of application areas that benefit from high performance: the significance and benefits of high performance in areas such as weather forecasting, scientific applications that rely on the solution of differential or partial differential equations, applications that rely on the use of matrix algorithms, business applications. Databases, artificial intelligence. Applications for simulation and modeling. Typical characteristics.
Quantifying performance; the mathematical and statistical basis for measuring and comparing performances of machines. Speed and size and their influence on performance. Inclusion of communication costs in multi-processor systems. Uni-processor: mechanisms for improving performance within a single processor; to include interleaving, cache memory, instruction look ahead, pipelining of instructions, special performance arithmetic and logic units.

Multi-processors: pipelined vector processors, processor arrays, multiprocessors, multi-computer systems such as clusters of workstations - control driven nature of these.

Dataflow machines: with decentralized control; the Grid concept; massively parallel machines. The mechanisms, the performance issues, the trends.

Learning objectives:
1. Understand and explain the trends that the high performance computing field has experienced.
2. Understand the design behind a range of high performance computer systems and explain how the various designs improved upon their predecessors.
3. Analyze the design weaknesses inherent in each implementation.

ACP2. Models [core]
Suggested time: 3 hours

Topics:
Classification of models and the basis for this: parallel machine models (SIMD, MIMD, SISD, MISD): Flynn’s taxonomy, Handler’s classification. The parallel random access machine, message passing.

Granularity issues: fine grained, medium grained, and coarse-grained granularity. Different levels of parallelism – the options and what guides selection.

System organizations: to include uni-processor, parallel architectures, and distributed architectures. Advantages and disadvantages of each.

Multiprocessors and multi-computers: the range of possible topologies, tightly coupled and loosely coupled options.

Multi-user models: shared memory vs. distributed memory; threads / processes models; multi-flow models; distributed processing models.

Learning objectives:
1. Explain and defend the basis for the classification of different models of high performance computing.
2. Outline the main features of the classical models used within processing architectures
3. Compare and contrast the various models and demonstrate an ability to make an appropriate selection for one of a set of applications.

ACP3. Architectures

*Suggested time: 5 hours*

**Topics:**
- Design of high performance processing units: role of parallelism, use of special purpose chips (e.g. for graphics, input/output), pipelining issues, special purpose arithmetic and logic units. VLSI considerations and impact on performance.
- Processor organizations: the options, the popular options and the reasons for their popularity.
- Processor to memory interconnections: bus, cross-bar switch, multi-stage interconnection networks. Illustrations of successful designs.
- Networks of workstations: various approaches to interconnection, appropriateness of each possible approach.
- High performance I/O systems; input/output overlap direct memory access, special purpose input/output processors.
- Other architectures: systolic arrays, dataflow machines and hybrid architectures, e.g. very long instruction word (VLIW) machines, multiple SIMD (MSIMD) machines.

**Learning objectives:**
1. Explain the features of computer architectures that contribute to high performance.
2. Discuss how each of the features of the architectures improves performance.
3. Present how components can be combined to produced high performance computer systems that meet particular requirements.

ACP4. Operating systems

*Suggested time: 3 hours*

**Topics:**
- Operating system features for high performance machines: separate supervisors, master-slave or client server, symmetric organization (e.g. floating mater) possibilities. Examples and history of development. The special issues, extensions to existing
operating systems or special purpose systems, the nature of any extensions, how extensions affect operating systems performance. The range of possible approaches and comparisons between them, ensuring that the power is realized, exploiting parallelism, possibly massive parallelism.

Concurrency within operating systems: problem of deadlock and deadlock avoidance; other problems – starvation, liveness, etc.; synchronization.

Scheduling: the range of possible algorithms, their strengths and weaknesses of the different approaches. The influence of scheduling on performance.

Features of operating systems that allow networks of computers to co-operate to collectively realize high performance.

Possibilities of having parallelism in the operating system or in the programming language; the benefits of each.

**Learning objectives:**

1. Understand and explain the features of operating systems that are designed to support high performance and to recognize the limitations of these.
2. Select from the range of possibilities a computer system most appropriate for one of a range of application areas.
3. Demonstrate an ability to predict the performance of different architectures.

**ACP5. Software**

*Suggested time: 5 hours*

**Topics:**

The need for concurrency, parallelism features in programming languages (as an alternative to using the facilities of the operating system). Languages for expressing concurrency and parallelism as well as synchronization in algorithms: the range of possibilities including semaphores, remote procedure calls, message passing. Problems of deadlock, liveness, etc.; steps to avoid this. Termination detection.

Compilation issues including parallelising compilers and super-compilers; the nature of the challenge for the compiler writer; typical solutions and the generality of these for different architectures. Code re-organisation, transformations. The problems of identifying parallelism automatically.

High performance Fortran (HPF) as an extension of Fortran 90; nature of extension.

Developing software for high performance: message passing interface (MPI) - a library of 129 functions that can be called from C or Fortran programs; Bulk synchronous parallel (BSP) model, super-step concept; the Linda system, having processes as items in a tree structured memory.
Typical approaches to writing applications. The software engineering issues.

Software infrastructures utilized for communication.


HCI issues – the possibilities, the benefits

**Learning objectives:**

1. Compare and contrast the different approaches to the development of software systems to achieve high performance.
2. Select an approach appropriate to the range of typical applications and demonstrate an ability to implement such an application.

**ACP6. Algorithms**

*Suggested time: 5 hours*

**Topics:**

Design of parallel algorithms: trade-offs, measurement, including communications costs. Algorithm design: strategies, insights, and patterns. Influence of scheduling on performance. The issues involved in adapting serial algorithms and applications to high performance systems. Termination detection and fault tolerance.

Crucial relationship between algorithms and architectures – match and measurement thereof; measurement of processor utilization.


Theoretical considerations: performance considerations, complexity measures, limits to speed-up; factors contributing to this.

Particular algorithms: sorting, fast Fourier transform, graph algorithms, combinatorial problems, numerical algorithms, problems from logic. Committee coordination, parallel garbage collection.

**Learning objectives:**

1. Recognise the strategies applicable to the design of efficient parallel algorithms.
2. Explain the major developments in algorithm design for high performance.
3. Understand the limitations of parallelism for improvement.
**TFT - Testing and Fault Tolerance**

**TFT1 – Faults and Fault Models in Digital Circuits**
- Logical (stuck-at) faults (single and multiple)
- Bridging faults and opens
- Delay faults
- Fault equivalence and dominance
- Yield and defect levels
- Test coverage
- Fault simulation and fault grading

**TFT2 - Test generation methods**
- The D algorithm
- PODEM, FAN, Learning Algorithms
- Automatic Test Pattern Generation (ATPG)
  - Pseudorandom techniques
  - Deterministic test pattern generation
- Test generation algorithms for sequential circuits
- Memory testing
- PLA testing

**TFT3 - Design for testability**
- Testability measures (controllability, observability)
- Scan and partial scan design
- BIST and other design for testability techniques
- Boundary scan and the IEEE 1149.1 testability standard
- Ad-hoc methods

Fault models in digital circuits, test generation algorithms, test generation for sequential circuits, fault simulation, testability measures, fault coverage, yield and defect levels, design-for-testability approaches, scan and boundary scan, IDDQ testing, mixed signal testing. Architecture and design of fault tolerant computer systems using protective redundancy, estimation of the reliability and availability of fault tolerant systems, error recovery, and fault diagnosis.
TFT4 – Testing non-stuck-at faults
  CMOS opens testing
  Performance and delay testing
  IDDQ and other current based tests
  Mixed signal testing

TFT5 - System-level test and diagnosis
  Printed circuit board testing
  MCM and core based testing
  System testing
  Verification Testing

TFT6 - Reliability and fault tolerance definitions
  Reliability and availability modeling

TFT7 - Error detecting and correcting codes
  Single Error Detection
  Single Error correction/Double Error Detection
  Burst Error detection codes
  Hamming and Reed/Solomon Codes
  Test compaction and aliasing

TFT8 – Fault Tolerant System Design
  Hardware redundancy approaches
  Fault tolerance in VLSI devices
  Aerospace systems
  Telecommunications systems
  Industrial control applications
  Fault-tolerant transaction processing systems
  Software approaches

TFT9 – Software fault tolerance
  Software reliability models
  Software fault-tolerance methods (N-version programming, recovery blocks, rollback and recovery )
  Fault tolerance in operating systems and data structures
  Fault tolerance in database and distributed systems

TFT10 – Software Testing (this may belong in Software Engineering)
  Test specifications
  Black box testing
  White box testing
  Random tests
  Test coverage
Digital System Verification (DSV)

DSV0. History and Overview Including Pentium Bugs and other Horror Stories, Verification vs. Validation. Relationship of Good Design Practice to Verification (3 hours)
DSV2. Comparison of Simulation, Testing, and Formal Verification (Timing Analysis) for Timing (4 hours)
DSV3. Formal Verification: Model Checking (10 hours)
DSV4: Formal Verification: Proofs (6 hours)
DSV5: Formal Verification: Equivalence Checking (3 hours)
DSV6: Verification by Simulation and Testbenches (4 hours)
DSV7: Verification by Assertions and Verification Languages (4 hours)
DSV8: Verification by Testing (2 hours)
DSV9: Other Verification: Signal Integrity, Specification, Reliability, Safety, Power, Cooling, ASIC Physical Design, … (3 hours)
DSV10: Comparison and Contrast of Verification, Testing, and Reliability (1 hour)
DSV11: Configuration Control, Bug Tracking, Regression Testing (2 hours)
DSV12: Economics of Verification (2 hours)

Possible texts and references:


For further reading, see Randy Bryant's web page: [http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bryant/www/home.html](http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bryant/www/home.html)


Mike Gordon Notes/Transputer Paper on IEEE Arithmetic


Two sites on open vera verification language

www.open-vera.com
www.smartverification.com
Digital Systems Engineering: Signal Integrity (DSI)

[elective] (44 hours) - Strawman, this needs lots of work.

- **DSI0**: History and Overview, Motivation, Importance, Horror Stories (2 hours)
- **DSI1**: Signals (4 hours)
- **DSI2**: Lossless Transmission Lines (12 hours)
- **DSI3**: Coupled Lines (6 hours)
- **DSI4**: Measurement (3 hours)
- **DSI5**: Simulation (3 hours)
- **DSI6**: Signaling (8 hours)
- **DSI7**: Power distribution (6 hours)
- **DSI8**: EMI/EMC(?)

**DSI0**: History and Overview, Motivation, Importance, Horror Stories (2 hours)

**DSI1**: Signals (4 hours)
- Representation of data
- Noise margin
  - Definition (real definition, not typical text definition)
  - Calculate from transfer characteristics
- Noise immunity
- AC characterization, delay, setup, hold times

**DSI2**: Lossless Transmission Lines (12 hours)
- Propagation of signals, forward and backward waves
- Characteristic impedance
- Thevenin equivalent circuits
- Reflections
- Terminations
  - series and shunt
  - interaction between driver/receiver characteristics and termination type
- Lumped L and C
- Return path requirements, connectors, packages, capacitors
- Nonlinear terminations, bergeron method
- Lossy lines
  - R and skin effect
  - Dielectric loss
- ISI
- Equalization and balanced codes

**DSI3**: Coupled Lines (6 hours)
- Odd and even modes
Differential signals and termination
Reflections on coupled lines
Crosstalk
Weak coupling
Odd-even mode coupling

DSI4: Measurement (3 hours)
TDR theory and measurement
VNA theory and measurement

DSI5: Simulation (3 hours)
HSPICE simulation of reflections, loss, coupling, power distribution, …

DSI6: Signaling (8 hours)
Overview: meso, plesio, ...
Timing: common clock
Skew and Jitter
Clock distribution
PLLs
Metastability
Mesochronous clocking
Plesiochronous clocking

DSI7: Power distribution (6 hours)
Power distribution system
Package parasitics
PC board characteristics
Impedance vs frequency
Capacitor models RLC, distributed, ...
PC board models for power distribution and resonance

DSI8: EMI/EMC(?).
Intelligent Systems and Automation (ISA)

Proposal for New Knowledge Unit

During the Atlanta meeting there was discussion about the scope and breadth of this topic of Computer Engineering. On several occasions questions were raised about where to draw the boundary with a number of comments related to this being made. Having given the matter some additional thought can I suggest that we include the following and see this as the limit of the discipline. I suspect that if we fail to include this then that omission will only attract comment in the review process.

Unfortunately I have not had time to develop this further. Of course, there is no point in doing so unless the proposal attracts agreement. With this in mind can I suggest that the text below gets included in the Body of Knowledge for Computer Engineering.

Still to be developed.

To include topics such as robotics, speech processing, vision systems and image processing.