# Computer Science Curriculum 2008:

An Interim Revision of CS 2001

Report from the Interim Review Task Force

(Date here)

Association for Computing Machinery

IEEE Computer Society

## Composition of the CS2008 Review Taskforce

The CS2008 Review Taskforce is a joint taskforce of the ACM and IEEE Computer Society, with members from both associations.

They are:

**Members of Special Advisory Board**

Endorsed by the ACM Education Council, November 15, 2008

Endorsed by IEEE Computer Society Educational Activities Board, November 16, 2008

# Contents

# Preface to the Interim Review

In recent times, the ACM and the IEEE Computer Society have sought to provide curriculum guidance on computing at approximately ten-year intervals. Thus 1968, 1978, 1991, and 2001 were the dates of publication of previous guidance on Computer Science.

Around the time of the publication of the most recent Computer Science volume, in December 2001, a commitment was made by the ACM and the Computer Society to provide curriculum guidance on a more regular basis. This was to recognize the rapid rate of change in the discipline and the consequent need for guidance to the community. It was felt that after a period of around 5 years steps should be taken to address this. Yet such guidance should not be seen to create revolution or confusion in the community; rather it should help and support. This present volume is provided in that spirit.

Since 2001, much has happened in computing. Today there is talk of a crisis, with enrollments having plummeted in many countries, often by as much as 60 – 70% from the peak of 2001. This fall in numbers has come at a time when there is increased recognition of the role of computing in innovation across engineering, in science, in business, in education, in entertainment and indeed in all walks of life. At the same time, the number of jobs in computing has risen while the supply of good graduates has fallen and some data suggests is failing to meet the demand in certain countries. The reasons for this are many and complex. However, many argue that the traditional curriculum in computing is unattractive to present-day students and that creates a challenge.

Part of the CC 2001 endeavor was to create documents that would complement the Computer Science guidance document. This resulted in the publication, over recent years, of volumes in Computer Engineering, Information Systems, Information Technology, and Software Engineering. An Overview volume has also been published; this sought to highlight the differences and draw out the similarities, but basically to provide a framework within which the various volumes could be seen to fit. This creates a different kind of environment in which to review the Computer Science volume.

Taking all these various matters into consideration, this review of the computer science volume comes at a crucial time. In addition, there is wide recognition that a considerable amount of work is needed to discover better and more effective ways of presenting the discipline of computing.  This has enormous importance, economic and strategic. Yet it would be misleading to recommend ideas that were not regarded as sound advice and best practice based on appropriate trials and testing.

This interim review has benefited from input from many (from industry, academia, etc.) through consultation and through discussion. It should be seen as a necessary updating of the influential CS2001 volume. The process has lead to wide recognition of the need to find new and better ways to present and portray the discipline of computer science; that remains a challenge for us all.

# Chapter 1 - Introduction

The Review Task Force (RTF), commissioned by the ACM Education Board and the IEEE Computer Society Education Activities Board, was given a mandate to conduct an interim review of the Computer Science 2001 volume (hereafter referred to as CS2001) that had been published on 15th December, 2001. During the work on the CC2001 series of volumes, it was recognized that there was a need to produce curricular guidance for the community on a regular basis; this was interpreted as being roughly every 5 years. This is necessitated by the pace of change in the discipline and the consequent need to support the meaningful and effective evolution of programs of study.

The idea of an interim review was a new concept in the activities of ACM / Computer Society curricular guidance. It was not envisioned that a completely new set of curricular advice for Computer Science would be produced, but that the work on this occasion would absorb significantly less resources than a full review. Rather it had been anticipated that a variant of, or addendum to, the earlier document would be produced with significant matters being updated as necessary. Part of the mandate given to the RTF would involve carrying out a modest but hopefully effective consultation with the community, taking account of relevant industrial views. For instance, the structure might be expected to remain largely unaltered but the review would provide the opportunity to update the guidance and to address issues of concern to the community.

## 1.1 Consultation Process

As part of the consultation process, the views of the community were sought and used as important input to the review process. Several approaches to obtaining these were adopted

- A web site was created to capture comments of any kind on the existing CS 2001 document; an announcement was then circulated to interested groups:

  *The ACM and IEEE Computer Society invite the community to contribute input to reviewing the 2001 Computer Science Curriculum Volume. A web site has been created to capture community comments and contributions. See*
  *[…]. The review period extends to 30th June 2007.*

- In excess of 8000 e-mail messages were sent out to interested parties drawing attention to the opportunity to provide comments; an alternative possibility offered to recipients of the e-mail was to attend an open public meeting to discuss the CS 2001 volume
- Public meetings were held
- Members of the RTF attended other meetings and drew attention to the opportunities to provide feedback
- Individual perspectives were obtained

As a result of the public consultation, feedback was received from 68 professionals and academics on the CS2001 website, with approximately 163 comments on various aspects of the curriculum. An in-depth discussion of the Computer Science volume was also held with fifteen participants at a meeting in Colorado, and this brought together representatives of the ACM, the IEEE Computer Society as well as representatives from industry and the two-year college community. Additional valuable feedback has also been received from individuals.

## 1.2 Indicators for change

As a result of the consultation several indicators for change were identified

*Experience of using the original*

The feedback certainly confirmed the view that it was timely to be undertaking a review of the Computer Science volume. In addition comments and suggestions pointed to the desirability of a mechanism whereby the community could be more deeply involved on an ongoing basis with the evolution of curricular guidance. In fact the RTF had recognized that issue and saw that as an important part of their work.

*Feedback from consultation*

A recurring theme was that industry involvement was essential and the needs of industry had to be reflected in the review. In addition, the following were identified for attention:

- Security, including the need to address this systematically and not just in operating systems and networking, but also in programming was repeatedly identified as being of major concern; some argued that a substantial element of security should form part of the core and so be compulsory for all graduating students
- All topics had to be updated, but specific mention was made of concurrency, net-centric computing, human computer interaction, software engineering, management information systems, systems issues and professional practice
- Learning outcomes had to receive more systematic and careful attention; these were deemed to be very important since they had to capture what students could reasonably be expected to do on completion of the program of study, module, etc.
- Communication and other transferable skills had to be addressed
- There were undercurrents about the current "crisis" in computing, generally, and the need to take steps to address this
- The final set of comments (from the final consultation) were dominated by attention to the teaching of programming / programming languages, and to the topic of multiple paradigms

*Industrial perspectives*

As might be expected, feedback from industry was diverse but invariably it was supplied willingly (even enthusiastically) and with a deep sense of conviction. It tended to confirm the importance of enlisting high quality students who then had to be attuned to a good work ethic and receive sound education in the fundamentals of the subject.

The following topics tended to receive attention in the dialogue with industry:

- security
    - seen to be of increasing importance, to the extent that all graduates should have a general awareness of security issues
    - should embrace the issues associated with access, encryption, networking, etc.
    - importantly this should also include a general awareness about how to write safe and secure software – the latter was deemed to be very important
- quality issues
    - testing, debugging and bug tracking
    - checking on code readability and documentation
    - code reviews
- software engineering principles and techniques
    - this included such matters as basic release management principles and basic source control principles
    - best practices for developing software in teams
- code archeology
    - delving into big, ill-documented code bases and making sense of them
- performance tuning
    - back-of-the-envelope calculation in design, etc.

Particular mention was made of the perceived benefits of employing students who had contributed to open source software projects, who had experienced industry internships, or who had done undergraduate systems software projects as part of their coursework or research.

Several industrialists passed very positive comment about compiler courses. Although many companies do not engage in anything related to compilers, compiler writing tended to be seen as a microcosm for realistic software development. So good compiler writers are often seen as desirable; they tend to be good software engineers.

An emphasis was placed on the problems of students having been indoctrinated in particular tools or processes that they then have to unlearn. In an ideal world, candidates need to have an appreciation for why particular topics (such as those mentioned above) are important and they ought to be taught some guidelines and best practices that will help them think about the craft of modern software development. A quote from one industrial commentator captures many of the concerns:

> *The thing that we can't afford to do […] is teach candidates how to think critically, to be effective problem solvers, and to have basic mastery of programming languages, data structures, algorithms, concurrency, networking, computer architecture, and discrete math / probability / statistics. I can't begin to emphasize the importance of algorithms and data structures to the work we do here […]. With multi-terabyte disks, bigger broadband pipes, etc. on the way, the big data problems that demand these skills […] are quickly going to be in need in a huge number of programming contexts.*

## 1.3 Important contextual considerations

Since the publication of the original CS2001 report (http://www.sigcse.org/cc2001), the landscape in terms of curricular guidance has changed considerably. The full CC2001 series (including the five volumes on Computer Engineering, Computer Science, Information Systems, Information Technology and Software Engineering) has now been published together with an Overview volume. The five volumes were produced by independent teams of experts who were charged with providing the best advice possible for the community. A common framework of a body of knowledge, with knowledge units and learning outcomes, etc., was used to provide a level of uniformity in approach throughout the series.

The Overview volume sought to take a high level perspective. To quote from the introductory Summary section in that volume:

> *This report summarizes the body of knowledge for undergraduate programs in each of the major computing disciplines, highlights their commonalities and differences and describes the characteristics of graduates from each kind of undergraduate degree program.*

Of course, there have been other important developments since 2001. For instance, the notion of *computational thinking* (see [Wing, 2006]), an idea intended to capture the kind of thinking that characterizes a study of computing, has become important; and some have sought to identify key principles on which the teaching of computing should be based. Also, there has been a crisis in terms of the numbers of students wishing to study in the discipline and that must not be ignored. The latter has given rise to the recent CPATH initiatives, by the US National Science Foundation, which aim to transform and revitalize computing education.

## 1.4 Structure of the interim CS2008 computer science report

The main body of the report consists of seven chapters. In Chapter 2, we articulate a set of principles that have guided the work; chapter 3 outlines developments that have taken place in computing and how these have affected change, chapters 4 and 5 are about changes to the curriculum in the broad sense, chapter 6 includes thoughts on the crisis and the final chapter offers some concluding thoughts.

The bulk of the material in the report appears in three appendices. Appendix A contains an overview of the revised body of knowledge, Appendix B contains the detail of the body of knowledge for undergraduate computer science and Appendix C consists of descriptions of certain new recommended courses that can be included in curricula. It is intended that, by providing both the body of knowledge and course descriptions and using these in conjunction with the original CS 2001 report, departments can create effectively updated curricula easily.

# Chapter 2 - Principles

Based on the analysis of past curriculum reports prior to 2001 and the changes in the discipline outlined in the preceding chapters, the CS2008 Review Task Force articulated a set of principles to guide its work; many of these are similar to or an evolution of similar principles that appeared in CS2001.

## On Computing

1. *Computing is a broad field that extends well beyond the boundaries of computer science.* A single report that covers only computer science cannot address the full range of issue that colleges and universities must consider as they seek to address their computing curricula. Additional reports in Computer Engineering, Software Engineering, and Information Systems have been produced in this series; a further report on Information Technology is at an advanced stage of preparation. These address major sub-disciplines, but additional possibilities still exist. Moreover an Overview volume has been produced in an attempt to characterize these sub-disciplines and also to characterize graduates from such degree programs, including specifically computer science.

2. *Like CS2001, CS2008 should seek to identify the fundamental skills and knowledge that all computing students must possess.* Despite the enormous breadth of computer science, there are nonetheless concepts and skills that are common to computing as a whole. CS2008 must attempt to identify and articulate the common themes of the discipline and make sure that all undergraduate programs include this material.

3. *CS2008 must continue to strive to be international in scope.* Despite the fact that curricular requirements differ from country to country, CS2001 was intended to be useful to computing educators throughout the world; CS2008 should further endorse that perspective. Although it will be strongly influenced by educational practice in the United States, every effort is made to ensure that the curriculum recommendations are sensitive to national and cultural differences so that they will be widely applicable throughout the world.

4. *CS2008 must include updated professional practice as an integral component of the undergraduate curriculum.* These practices encompass a wide range of activities including management, ethics and values, written and oral communication, working as part of a team, and remaining current in a rapidly changing discipline. In CS2008 cultural issues now merit inclusion. We further continue to endorse the position articulated in the CC1991 report that “mastery of the discipline includes not only an understanding of basic subject matter, but also an understanding of the applicability of the concepts to real-world problems.”

5. The Interim Review Task Force needs to recognize that, since around 2001, student enrolment numbers have been falling to the extent that a crisis now exists in the Computing discipline. *The Interim Review needs to present the Computer Science discipline in as positive a light as possible, as seen by current applicants.* Since around 2001 student numbers have fallen by around 50% in the US and in many other countries. Now there are not sufficient numbers of graduates to fill the many important positions that exist.

## On Computer Science

6. *Computer science continues to draw its foundations from a wide variety of disciplines.* Undergraduate study of computer science requires students to utilize concepts from many different fields. All computer science students must learn to integrate theory and practice, to recognize the importance of abstraction, and to appreciate the value of good engineering design.

7. Development of a computer science curriculum must be sensitive to changes in technology, new developments in pedagogy, and the importance of lifelong learning. In a field that evolves as rapidly as computer science, educational institutions must continue to adopt explicit strategies for responding to change. Institutions, for example, must recognize the importance of remaining abreast of progress in both technology and pedagogy, subject to the constraints of available resources. *Computer science education, moreover, must seek to prepare students for lifelong learning that will enable them to move beyond today’s technology to meet the challenges of the future.*

8. *The required body of knowledge, i.e. the core, should be updated from CS2001 to reflect changes in the discipline.* Yet it should still be made as small as reasonably possible. As computer science has grown, the number of topics required in the undergraduate curriculum has also grown. Since 2001, there have been changes in key areas such as computer architecture, communications, human computer interaction and web technologies. But in addition topics such as security have gained a greater degree of prominence. These considerations have caused some adjustments to the core in CS2008. As before, coverage of the core is not limited to introductory courses, but extends throughout the curriculum. At the same time, it remains important to recognize that this core does not constitute a complete undergraduate curriculum, but must be supplemented by additional courses that may vary by institution, degree program, or individual student.

9. *The development of CS2008 must remain broadly based.* To be successful, the process of creating the CS2008 recommendations must include participation from many different constituencies including industry, government, and the full range of higher educational institutions involved in computer science education.

10. The Interim Review Task Force must take into account relevant feedback from industry and seek to address their current needs.

## Course design and implementation

11. *CS2008 should follow the style of CS2001 and go beyond knowledge units to offer significant guidance in terms of individual course design in particular areas.* Articulating a set of well-defined models will make it easier for institutions to share pedagogical strategies and tools. It will also provide a framework for publishers who provide the textbooks and other materials for those courses.

12. *CS2008 must include discussions of strategies and tactics for implementation along with high-level recommendations.* Although it is important to articulate a broad vision of computing education, the success of any curriculum depends heavily on implementation details. CS2001 provided institutions with advice on the practical concerns of setting up a curriculum by including sections on strategy and tactics along with technical descriptions of the curricular material. CS2008 must do likewise in important areas.

13. In seeking to address the Computing crisis, the Interim Review Task Force has been mindful of the need to protect the discipline of Computer Science. *It has taken the view that the crisis should be addressed mainly through pedagogical considerations.*

## The Review Process

14. *The rapid evolution of computer science requires an ongoing review of the corresponding curriculum.* Given the pace of change in this discipline, the process of updating the curriculum once a decade has become unworkable. The professional associations have now established an ongoing review process that allows individual components of the curriculum recommendations to be updated on a recurring basis. A joint group formed by representatives of ACM and the Computer Society has been formed for this purpose.

# Chapter 3 Developments

The review had to consider a range of issues. There have been significant changes in technology, changes in the perceived importance and relevance of certain topics, and often changes in the context in which computer science is taught. The purpose of this chapter is to provide a brief overview of the main influences and to indicate the main changes that have occurred.

Early in its history, the original CS2001 Task Force identified a set of 14 areas that together represented the body of knowledge for computer science at the undergraduate level, as shown in Figure 3-1. This structure remains in this interim report.

## 3.1 Recent Trends

Since the development of CS2001, some relevant trends in the evolution of the discipline of computer science have become apparent. These include

*The emergence of security as a major area of concern*

The amount of malicious software in the form of viruses, worms, etc is causing huge concerns to the extent that it is now seen as a major threat to the industry. The Review Task Force has received urgent requests from industrialists requesting that substantial attention to security matters be regarded as compulsory for all computing graduates.

Often the issue of security is seen as being related to an improved view of the use of passwords, access controls of various kinds, the use of firewalls, employing cryptographic techniques, data security and so on. However, many leaks come about through inadequate attention to very basic matters such as programming techniques. Failure to program properly can result, for instance, in exploitable software defects such as buffer overflow and related problems. In addition web-related security breaches are increasingly common, and again need to be addressed appropriately.

*The growing relevance of concurrency*

The development of multi-core processors has been a significant recent architectural development. To exploit this fully, software needs to exhibit concurrent behavior; this places greater emphasis on the principles, techniques and technologies of concurrency.

Some have expressed the view that all major future processor developments will include concurrent features, and with even greater emphasis on the concurrency elements. Such a view implies the increased emphasis on currency will not be a passing fashion but rather it represents a fundamental shift towards greater attention to concurrency matters.

The increased ubiquitous nature of computing and computers represents a further factor in heightening the relevance of this topic and again there is every indication that this will only gain further momentum in the future.
It is expected that these observations will have implications for many knowledge areas in future curriculum guidelines.

*The pervasive nature of net-centric computing*

The rise in the use of the web has been one of the phenomena of the 21st century. Now its use is pervasive. From the perspective of the computer scientist, there are implications for the knowledge areas that address programming, software engineering, data management, mobility, human computer interaction, security and intelligent systems. Web engineering has now emerged as a new discipline, and web science is emerging.

These developments have served to blur the traditional distinction between the areas of operating systems and networking / communications.

One of the characteristics of these developments is that they are not focused on any one of the CS2001 knowledge areas. Rather they tend to cut across the existing knowledge areas and even start to call into question the datedness of the existing structure of the body of knowledge areas. Other topics such as games computing or entertainment computing, as well as simulation and modeling tend to do likewise.

An additional important observation is the stronger emergence of the concept of *systems issues*; by this is meant a number of entities working in harmony to achieve a desired effect. This concept was present in CS2001 but topics such as

- The development of computer systems where different components are required to co-operate in order to achieve a particular level of performance for a particular cost.
- Security typically requires attention to many different facets of the computer and its software; this often applies also to such matters as reliability.
- Software engineering tools (or even user applications) have to work together seamlessly

The Review Task Force considered these observations in some detail and with some care. Indeed new knowledge areas for security and concurrency were contemplated to ensure that these important topics were looked at in a systematic and disciplined manner. In the end the various topics were distributed throughout the existing knowledge areas. An important consideration in the thinking of the RTF was the intensive argument, debate and even tension involved in assigning hours to the core topics during the development of the original CS2001. To ignore that without going through a similar extensive exercise carried dangers.

The nature of an interim review was seen as largely an updating exercise, rather than an activity that would cause a major rethink about the discipline and how to present it; however, it was recognized that the need for the latter was growing daily.

## 3.2 Summarizing the Main Changes

The purpose of this section is to capture in a succinct form the major changes that appear as a consequence of this interim review of the CS2001 Computer Science volume. In summary, this new report

- recognizes the existence of additional curricular advice that has been published since around 2001
- incorporates a general updating of the body of knowledge
- includes advice on new courses or course fragments that are provided as exemplars.

All this is to be expected. But beyond that several additional steps were taken. Many of these have been driven by the increased attention to security and concurrency. But other alterations have occurred.

### 3.2.1 Knowledge areas

Figure 3-1. The 14 knowledge areas

| | |
|---|---|
| Discrete Structures (DS) | Human-Computer Interaction (HC) |
| Programming Fundamentals (PF) | Graphics and Visual Computing (GV) |
| Algorithms and Complexity (AL) | Intelligent Systems (IS) |
| Architecture and Organization (AR) | Information Management (IM) |
| Operating Systems (OS) | Professional Issues |
| Net-Centric Computing (NC) | Software Engineering (SE) |
| Programming Languages (PL) | Computational Science (CN) |

There has been no change to the set of knowledge areas. In large part this has been to reduce the possibility of the interim report leading to widespread change, though it is fully recognized that a certain level of reappraisal is desirable. The need for rethinking the set of knowledge areas and the manner in which the discipline is portrayed has been recognized and discussed.

Changes have been made to the individual knowledge areas and these are detailed in Appendix A. Often these reflect the greater emphasis on security, as well as a necessary updating of net-centric computing. The main changes are summarized below.

*Discrete Structures*

There is a suggestion for placing less emphasis on the notion of purely formal (symbolic) proof, to be replaced by a greater focus on student's ability to produce rigorous and sound proof arguments. For instance, it might be effective for students to get more practice using mathematically rigorous (yet not entirely symbolic) proof argument strategies for reasoning about predicate calculus sentences, replacing some practice with fully symbolic approaches for reasoning about propositional sentences; for clarification, see the learning objectives for DS/BasicLogic in the Body of Knowledge.

*Programming Fundamentals*

Attention is drawn to the need for increased levels of care and attention in the teaching of basic programming. In part this is to address security concerns, so that students become aware of the need to program in such a way that they do not create security loopholes.

*Computer Architecture*

This has been revised and there is now less attention to the logic design level, with architectural issues now being updated (e.g. to include multi-core processors) and receiving greater prominence. A section on devices has also been added. It is anticipated that the latter will provide greater opportunities for motivating students.

*Net-centric computing*

Given the rate of developments in this general area, it was inevitable that considerable changes would occur here. Several topics were given new, more currently meaningful names. Topics deemed less relevant, such as circuit switching and packet switching, streams and datagrams, and common gateway interface programs were removed. Emerging topics, including service-oriented-architecture and grid computing were added to the core.

*Intelligent Systems*

The idea of perception is introduced and there is greater emphasis on planning, on ontologies, etc. The area of games or entertainment software can be seen as part of this knowledge area and is now mentioned since for some it can provide an important motivational dimension to the curriculum.

*Professional Issues*

There is increased attention to the concept of identity theft, to cultural issues and to the apparent conflict between the demands of data protection and the requirements of freedom of information

*Computational Science*

Numerical methods have been dropped from the Computational Science and Numerical Methods knowledge area (though it is still referred to as CN). This is to recognize the fact that in most institutions numerical analysis is not seen to form a major part of the computer science curriculum though, of course, the topic remains important.

Typically such courses can be found in offerings from mathematics departments. Matters such as rounding errors, iteration, recurrence relations, etc. remain important; these concepts now appear elsewhere.

Simulation and modeling have been given greater prominence. In part this is to recognize the growing importance of the area in many applications (including science where the computer can be seen as the laboratory of the future). The link between simulation and modeling and object-oriented programming is important.

The unit on high performance computing was deemed to be dated and has been recast as a unit on parallelism. An intention of this change has been to better capture the notion of computational science. For clarification, see the learning objectives for DS/BasicLogic in the Body of Knowledge.

### 3.2.2 Additional Adjustments

Underpinning many of these changes a number of more subtle alterations have occurred. In particular:

- Certain ideas that have matured since 2001 have been identified and care has been taken to include these in the relevant knowledge areas and elsewhere. So there are certain principles, e.g. the principles of locality, that have been identified and the notion of computational thinking has become important
- It is suggested that there should be increased attention to a careful and considered approach to all aspects of technical work.
- There should be greater attention to software re-use and greater awareness of open source possibilities as well as greater familiarity with source control systems.
- There is mention of a more systematic approach to the treatment of learning outcomes. In part this is about recognizing the various levels of granularity but also about recognizing that learning outcomes can be seen as a starting point for the development of curricula.
- To increase the international appeal and relevance of the work, some comments on this are included
- A comment is made about the role of electives, reinforcing a similar comment in the earlier CS2001 volume. Increasingly computing students gain employment in application areas. To be effective and to be able to play a leadership role this often entails gaining some sophisticated domain knowledge. There is now a recommendation that students should be advised of this as they make choices of electives to accompany their computer science courses.

### 3.2.3 Renaming of Knowledge Units

In the Interim Review Report a new convention has been adopted for the naming of individual knowledge units. The original aim behind this renaming proposal was to avoid introducing ambiguity in the meaning of unit designators such as PF1 or NC3. In the updated body of knowledge, there have been changes to the knowledge areas, some rather small. But these are sufficiently significant to create possible problems. If the "<area><number>" naming convention were to be strictly maintained then

- removal or shifting of a knowledge area could result in holes in the numerical sequencing (an NC3 with no NC1 or NC2), or
- change the meaning of the existing designators.

Since those names do appear in various places on the web (the Commerce Department, for example, reprinted the CS2001 Body of Knowledge in a 2003 publication and uses those designators in the body of the report), it seemed best to introduce a new scheme that eliminates any possible ambiguity.

The convention adopted has been to retain the two-letter codes for the knowledge areas but to replace the sequence numbers with semantically meaningful identifiers, written as single lexical units using the traditional programming convention of marking the word boundaries with uppercase letters, so that, for instance,

DS5. Graphs and Trees

becomes

DS/GraphsAndTrees

This scheme has two significant advantages beyond avoiding ambiguity with the older scheme. First, the meaning of a knowledge unit is now obvious from the designator, and no one has to remember what the designator "DS5" means. Second, the new scheme eliminates the ordering constraint, which has simplified the revision process considerably. A slight disadvantage is that it is unclear where, for instance, DS/GraphsAndTrees appears in the sequencing within DS but an improved indexing of the knowledge units has sought to address that comment.

### 3.2.4 The Programming Languages Issue

The issue of programming languages and paradigms caused considerable debate at the time of the final public consultation. In brief, following a SIGPLAN conference in Harvard at the end of May, we received a request to reconsider the teaching of programming. That request generated a great deal of debate within the review committee and was welcomed as drawing attention to an important matter.

The review committee was divided in its response to the SIGPLAN proposal. On the positive side, the committee was convinced that students need exposure to more than one programming paradigm, for precisely the reasons outlined in the proposal. At the same time, there was no consensus within the review committee that the functional programming paradigm needed to be required in all undergraduate computer science curricula. There are certainly many successful curricula that do not require students to learn functional programming; making the specific changes in the knowledge units recommended by SIGPLAN would force those curricula to change to remain in compliance with the guidelines. We did not believe that we could justify making so far-reaching a change, particularly at the level of an interim review.

Our consensus recommendation is therefore to add a new requirement that students acquire facility with more than one programming paradigm to the set of general requirements in Chapter 9 ("Completing the Curriculum") of the CS2001 report. The new requirement follows the existing section 9.1.5:

> **9.1.5 Exposure to different programming paradigms**
>
> Computer science professionals frequently use different programming languages for different purposes and must be able to learn new languages over their careers as the field evolves. As a result, students must recognize the benefits of learning and applying new programming languages. It is also important for students to recognize that the choice of programming paradigm can significantly influence the way one thinks about problems and expresses solutions of these problems. To this end, we believe that all students must learn to program in more that one paradigm.
>
> Although we believe it is essential for all students to acquire experience with more than one paradigm, we believe that the choice of an appropriate secondary paradigm will depend significantly on the specific character and educational goals of each institution. Universities that seek to prepare students for positions in academia, research, and advanced development would be well advised to introduce functional programming so that students can take advantage of the mental discipline that those languages encourage. Programs that seek to prepare students to develop web-based applications might choose instead to use scripting languages. Such languages seem to be growing in commercial importance and are increasingly adopting features that were typically associated with functional languages in the past.

The review committee also plans to forward both the SIGPLAN proposal and the notes from our discussions to the next full-scale curriculum committee, which should be appointed sometime in the next year or two. As is clear from the interest that has been generated by this discussion, programming and programming language issues will be central discussion items in the next round of curriculum revisions.

## 3.3 Addressing the Crisis

The impact of the crisis in computing has exercised the RTF and it felt compelled to take certain steps that might offer encouragement towards a solution. At any time, of course, it must be important to find better ways of educating students. This typically involves appealing to them, providing programs that they find interesting and challenging, and yet motivating. Particularly when computing generally is going through difficult periods due to image, perception and so on this is an extremely important matter. The RFT recognized that there are ongoing issues here that merit further investigation. Accordingly a new chapter has been supplied to address this.

Two particular topics were addressed in some detail:

- finding new and better ways of teaching programming, and
- trying to place computing in a context that would serve to motivate and inspire students.

Ultimately the community needs to experiment to find ways of overcoming each crisis-du-jour and of providing a healthier environment in which teaching and learning in computing can flourish and be widely respected and regarded.

## 3.4 Concluding Comment

Standing back, there do seem to be a number of fundamental changes afoot. For the most part, it is too early to be definite about their impact and influence in the long term, but we have tried to draw attention to them.

# Chapter 4 Program of Study Specification

Increasingly learning objectives are seen as central to curriculum design. They are often seen as a starting point for deliberations on particular courses, but their influence typically runs through an entire program of study.

The original CS2001 document used learning outcomes, but its treatment of them could have been more uniform. The purpose of this chapter is to address related issues including some key matters that have an influence throughout an entire curriculum.

## 4.1 On Learning Objectives

Learning objectives are central components of any body of knowledge; basically they capture important elements that are typically absent from a mere list of knowledge topics. They are intended to capture what students are able to *do* with knowledge. This typically varies from simple recall to include, for instance, more sophisticated process-oriented skills. Given the rate of change of knowledge, it is often argued with considerable justification that these skills ought to form the basis of curriculum development as they are fundamental to life-long learning. If properly selected, they encourage the student to make effective use of new knowledge as it becomes available.

There are subtle arguments about whether the term ‘learning objectives’ or ‘learning outcomes’ should be used. The underlying difference is normally explained in the following way: objectives are aspirational, whereas outcomes are mandatory in some sense. Since they are softer as a target, here the focus will be on objectives.

Learning objectives are relevant at various stages. For the purposes of this interim review, it is convenient to classify them as

- *benchmark learning objectives* are intended to capture the characteristics that describe what is expected from a student on any (for instance) computer science program of study
- *program learning objectives* are associated with programs of study and so capture at a high level the anticipated behavioral characteristics of graduates
- *class learning objectives* are associated with individual classes and so indicate the contribution that each class makes to the program learning objectives though usually in sufficient detail to guide intended participants (lecturer, students, etc.)
- *instructional learning objectives* are associated with knowledge units or topics and so include in great detail the expectations relating to individual parts of the class

In a properly designed program of study there will be important relationships between the various levels of objectives. In brief, the lower level objectives ought to imply the higher level objectives.

### 4.1.1 On the Nature of Learning Objectives

A set of educational objectives was first captured in Bloom’s taxonomy back in 1956. This has undergone a number of revisions though the elements of the original insights are still dominant. A slight variation on Bloom’s taxonomy is summarized below to take into account the meaning of specific terms (such as *execute* or *implement*) in a computing context. In this discussion we follow the views outlined in [Anderson et al, 2001].

Formally, typically a learning objective contains a *verb* and a *noun*:

the verb describes the intended cognitive process; the latter includes the elements of Bloom’s taxonomy and so words like remember, understand, analyze, design are all highly relevant.

the noun describes the knowledge that the student is intended to acquire; now knowledge itself can be categorized in various ways, with a spectrum running from the concrete to the abstract.

| Level | Category | Cognitive Processes |
|---|---|---|
| 1. | Remember | recognizing, recalling, describing, stating |

| | | |
|---|---|---|
| 2. | Understand | interpret, exemplify, classify, infer, compare, explain, paraphrasing, summarizing |
| 3. | Apply | execute (i.e. carry out), implement (i.e. use), compute, manipulate, solve |
| 4. | Analyze | differentiate, organize, attribute, discriminate, distinguish, sub-divide |
| 5. | Evaluate | check, critique, assess, compare, contrast |
| 6. | Create | generate, plan, produce, innovate, devise, design, organize |

Benchmark and program objectives tend to be associated with the higher levels of this taxonomy, whereas instructional objectives tend to be associated with the lower levels.

## 4.2 Characteristics of Graduates

As previously described, learning objectives can occur at program of study level, class or module level and knowledge unit or topic level; other possibilities also exist but these would seem to be the most commonly used in the CS2001 context. In any particular case, a certain consistency between these has to be established; thus the learning outcomes for the various classes should typically imply the learning outcomes for the program of study itself, and that will reflect the expected characteristics of graduates.

At a broad level, these expected characteristics of computer science graduates can be expressed as follows (and this constitutes an updating of a similar list in CS2001):

- *System-level perspective.* The objectives associated with individual units in the body of knowledge tend to emphasize isolated concepts and skills that can lead to a fragmented view of the discipline. Graduates of a computer science program must develop a high-level understanding of systems as a whole. This understanding must transcend the implementation details of the various components to encompass an appreciation for the structure of computer systems and the processes involved in their construction and analysis.
- *Appreciation of the interplay between theory and practice.* A fundamental aspect of computer science is the balance between theory and practice and the essential link between them. Graduates of a computer science program must understand not only the theoretical underpinnings of the discipline but also how that theory influences practice.
- *Familiarity with common themes and principles.* In the course of an undergraduate program in computer science, students will encounter many recurring themes such as abstraction, complexity, and evolutionary change. They will also encounter principles, e.g. those associated with caching, (e.g. the principle of locality), with sharing a common resource, with security, with concurrency, and so on. Graduates should recognize that these themes and principles have broad application to the field of computer science and must not compartmentalize them as relevant only to the domains in which they were introduced.
- *Significant project experience.* To ensure that graduates can successfully apply the knowledge they have gained, all students in computer science programs must be involved in at least one substantial software project. Such a project (usually positioned late in a program of study) demonstrates the practical application of principles learned in different courses and forces students to integrate material learned at different stages of the curriculum. Student need to appreciate the need for domain knowledge for certain applications, and that this may necessitate study within that domain.
- *Attention to rigorous thinking.* This may be formal but need not be but should include discipline epitomized by the use of sound practices which include planning, tracking progress, measuring and generally managing quality; this needs to be seen to complement sound design and sound choice of techniques.
- *Adaptability.* One of the essential characteristics of computer science over its relatively brief history has been an enormous pace of change. Graduates of a computer science program must possess a solid foundation that allows and encourages them to maintain their skills as the field evolves.

The expected capabilities and skills for computer science graduates can be captured under three headings:

*Cognitive capabilities and skills relating to computer science*

- Knowledge and understanding. Demonstrate knowledge and understanding of essential facts, concepts, principles, and theories relating to computer science and software applications.
- Modeling. Use such knowledge and understanding in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoff involved in design choices.
- Requirements. Identify and analyze criteria and specifications appropriate to specific problems, and plan strategies for their solution.

- Understanding the elements of computational thinking. This includes recognizing its broad relevance in everyday life as well as its applicability within other domains, and being able to apply it in appropriate circumstances.
- Critical evaluation and testing. Analyze the extent to which a computer-based system meets the criteria defined for its current use and future development.
- Methods and tools. Deploy appropriate theory, practices, and tools for the specification, design, implementation, and maintenance as well as the evaluation of computer-based systems.
- Professional responsibility. Recognize and be guided by the social, professional, legal and ethical as well as cultural issues involved in the use of computer technology. Increasingly cultural issues are also relevant.

*Practical capabilities and skills relating to computer science*

- Design and implementation. Specify, design, and implement computer-based systems.
- Evaluation. Evaluate systems in terms of general quality attributes and possible tradeoffs presented within the given problem.
- Information management. Apply the principles of effective information management, information organization, and information-retrieval skills to information of various kinds, including text, images, sound, and video. This must include managing any security issues.
- Human-computer interaction. Apply the principles of human-computer interaction to the evaluation and construction of a wide range of materials including user interfaces, web pages, multimedia systems and mobile systems.
- Risk assessment. Identify any risks (and this includes any safety or security aspects) that may be involved in the operation of computing equipment within a given context.
- Tools. Deploy effectively the tools used for the construction and documentation of software, with particular emphasis on understanding the whole process involved in using computers to solve practical problems. This should include tools for software control including version control and configuration management.
- Software reuse. Be aware of the existence of publicly available software (such as APIs or open source materials) and engage effectively in open-source projects.
- Operation. Operate computing equipment and software systems effectively.

*Additional transferable skills*

- Communication. Make succinct presentations to a range of audiences about technical problems and their solutions. This may involve face-to-face, written communication or electronic communication.
- Teamwork. Be able to work effectively as a member of a development team.
- Numeracy. Understand and explain the quantitative dimensions of a problem.
- Self management. Manage one’s own learning and development, including time management and organizational skills
- Professional development. Keep abreast of current developments in the discipline to continue one’s own professional development.
- Software reuse, open source issues. Separate compilation.

## 4.3 International Considerations

With the increased globalization of computing and the international nature of the workforce it is increasingly important to pay attention in the curriculum to certain related matters. There are a number of very basic issues and concerns that ought to be reflected in the design and development of the curriculum.

### 4.3.1 Some Basic Issues

*International competitiveness*

It must be the responsibility of all countries to ensure that their students are internationally competitiveness in terms of the abilities and skills they acquire during their education. This has implications for the whole range of technical skills as well as non-technical skills (so covering personal, professional, etc) and ensuring that the standards of the education provided are excellent. To do otherwise is to disadvantage the young and to open up the possibility of others coming in to fill local jobs.

Additionally some would place a degree of emphasis on attention to innovation in the curriculum; this tends to equip students with an attitude of mind that ultimately stimulates new developments and provides a set of incentives and a flair likely to lead to job creation.

In this context, competitions such as the The ACM Programming Contest provide important indications of health although the messages they send out must not be over-emphasized.

*Legal, social, etc issues*

There is reasonably wide agreement that this topic of legal, social, professional and ethical should feature in all computing degrees. Comments relating to it exist in all the curriculum guidelines and are seen as important. It needs to be observed that many aspects of this (e.g. the legal dimension) will vary from one country to another. So one view that emerges naturally is that some attention needs to be given to the separation between the national issues and international issues. Such a separation would serve to facilitate student mobility.

Separations of this kind have not normally been prescribed. Indeed taking a truly global perspective here may involve expert guidance from an international lawyer, and that is beyond the individual skills of most computing faculty. So some reasonable compromise seems inevitable and justified. One approach might be to focus on national perspectives for depth, but then to take a broad view of the truly significant matters in the international scene.

It seems important to recognize that such issues do impinge on technical areas such as human computer interaction, the design of interactive systems, as well as in information gathering activities of various sorts (e.g. of personal data).

*Cultural Issues*

Issues of a cultural nature are often ignored in computing degrees but increasingly they are important. These concerns straddle topics such as:

- Significance of certain colors, symbols, etc.; since there is often considerable sensitivity associated with these, it is important to be aware of them.
- Likewise dates, times of the year, etc. have special significance and often these can impact on working relations. It is beneficial to show sensitivity in such matters.
- Attitude to authority – this varies from one culture to another. It can imply an unwillingness to question authority, for instance. So what does this mean in the context of effective team building? Are alternative models then appropriate and if so what are these?

### 4.3.2 Additional Concerns

A number of additional matters can be considered beyond the more basic elements.

***Articulation Considerations***

Articulation of courses and programs between academic institutions is a process that facilitates transfer by students from one institution to another. The goal is to enable students to transfer in as seamless a manner as possible. Efficient and effective articulation requires accurate assessment of courses and programs as well as meaningful communication and cooperation. Both students and faculty have responsibilities and obligations for successful articulation. Ultimately, students are best served when educational institutions establish well defined articulation agreements that actively promote transfer.

Articulation agreements often guide curriculum content as well, and are important considerations in the formulation of transfer-oriented programs of study. Institutions are encouraged to work collaboratively to design compatible and consistent programs of study that enable students to transfer, in the United States from associate-degree programs into baccalaureate-degree programs, and in other countries from post-secondary colleges into universities. A two-year college must develop transition and articulation strategies for the colleges and universities to which its students most often transfer, recognizing that it may be necessary to modify course content to facilitate transfer credit and articulation agreements.

A student's program of study must also take into consideration the general education requirements at both the initial college and the anticipated transfer institution. Faculty must ensure that they clearly define program goals, address program learning objectives, and evaluate students effectively against defined course objectives. Articulation agreements should specify one or more well-defined exit points for students to matriculate from the post-secondary college to the transfer institution. In turn, faculty at the receiving institution must provide any transitional preparation necessary to enable transfer students to continue their academic work on par with students at their institution. Hence, students must expect to complete programs in their entirety up to well-defined exit points (e.g., completion of a defined course sequence or program) at one institution before transferring to another institution; one cannot expect articulation to accommodate potential transfers in the middle of a carefully designed curriculum. Acting on these considerations, all post-secondary institutions of higher education will foster student success and best serve their students' academic and career aspirations.

*Student mobility and related matters*

In some parts of the world (notably Europe at the present time with its Bologna process) there are positive incentives to encourage student mobility. There are two principal aspects to this

- equipping students so that they can readily move and take up study in other countries
- having a policy of welcoming suitable students from outside to come and study locally

There are implications from both of these aspects for education, not just on the computing front but also in terms of language skills, acquiring an understanding of other practices, and so on. But in both cases there is a need for educationalists to keep abreast of what is happening elsewhere and to respond accordingly.

*Economic concerns*

In any computing degree students ought to have some feeling for the financial and economic imperatives. Thus which approaches are expensive and is this expense justified? Which approaches are less expensive and is this sensible?

With the advent of outsourcing and off-shoring these matters become more complex and take on new dimensions. Then economic issues of an international dimension come to the fore but associated with these there are often related ethical issues concerning exploitation but also ownership; these are often difficult fine lines of demarcation in this area. Such matters ought to feature in courses on legal, ethical and professional practice.

For anyone producing software there is a requirement that they can guarantee to some level the functionality and related reliability of what the software will do. But thinking of errors, and other malicious or dangerous possibilities, there are also requirements in terms of what it should *not* do (e.g. it should not spread viruses). The practice of off-shoring has raised complications in such situations. In particular (sensitive) application areas this was particularly significant.

*Global Company Considerations*

Some organizations take the view that they will tailor or customize their products for use in other countries. This can involve them in merely translating an interface by expressing text in the language of that country. But, for instance, if the software provides information this may entail considerable work in gathering local information and in making that available. It may even involve legal concerns about what can be made available. In such situations considerable local knowledge and expertise may be required. Then other issues to do with best practice in teamwork activity come into play.

When local information is stored within a particular country, the best way of accessing that is often through the mother tongue. The availability of automatic translation mechanisms can facilitate access by someone from another country. Increasingly there are huge possibilities in this regard. So natural language processing becomes highly relevant.

### 4.3.3 Concluding Remarks

The comments made above must be seen as just a start. They are provided to raise awareness of the extent to which international considerations can potentially feature in the curriculum.

# Chapter 5 Course Considerations

## 5.1 The Core

In the original CS2001 report the concept of the core was seen as referring to 'those units required of all students in all computer science degree programs'. This was not seen as a complete curriculum but it identified those topics deemed to provide an essential base on which other courses could build. A key requirement was to keep the core as small as reasonably possible; in the event it ran to some 280 hrs. Keeping it small would facilitate a greater element of choice for students as they sought to complete their programs of study.

In this revision (CS2008) the concept of the core remains. As a result of changes, there are indeed adjustments to the content and to the size of the core. Some material has been removed and other considerations (such as the attention to security matters) have resulted in material being added. The core remains at a comparable size of 280 hours. (In keeping with CS2001 and earlier curriculum reports, an *hour* corresponds to the in-class time required to present the material and represents the minimum level of coverage required.)

## 5.2 Introductory Courses

The original CS2001 report outlines a range of approaches to the important early introductory courses. In particular it outlined various possible models including imperative first, objects first, functional first, breadth first, algorithms first and hardware first. These tended to reflect different philosophies and priorities about computing, especially approaches to programming.

Although the Interim Review Group did not undertake a systematic analysis of uptake or popularity of the different models, experience has been gained on the use of these. It seems that most of these approaches do have their advocates with the choice often depending of the particular orientation of the program of study. Of these, the algorithms first approach - in which 'basic concepts of computer science are introduced using pseudo-code rather than an executable language - seems to have received less favor.

There also seemed to be merit in providing some guidance on introductory courses on security. Historically this has often been seen as an advanced topic, and yet the inclusion of aspects of this in the core now implies that certain security material needs to be accessible to all students.

To address these two matters, guidance is offered in Appendix C.

## 5.3 Relationship to Two-Year College Programs

According to the American Association for Community Colleges, nearly one-half of all undergraduate students in the United States are enrolled in two-year colleges. One segment of the two-year college student population begins their academic careers intending to transfer to a university in order to continue their academic studies. These students must be served by well-defined articulation agreements, programs of study consistent with those found in university settings, and faculty sensitive to the special issues associated with such students and programs. Articulation considerations are detailed in chapter four of this report, where it is noted that curriculum content plays a crucial role in articulation success. The two computer science curriculum reports, CS2001: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science and the 2003 Guidelines for Associate- Degree Programs in Computer Science, established transfer equivalencies between four-year and two-year programs to foster articulation. Both reports have been updated to include recent developments in the discipline of computing, while maintaining the equivalencies previously established.

## 5.4 Organizing Principles

### 5.4.1 The Original Report

In the original CS2001 document (see section 8.2 of that report), a number of principles for organizing the curriculum beyond the introductory courses was enunciated. In summary these were identified as

*A traditional topic-based approach*

To quote:

The most common approach to the intermediate courses is simply to apportion the material into units based on the traditional divisions of the field. Thus, in this approach, students take separate courses in each of the core areas: a course in architecture, a course in operating systems, a course in algorithms, and so on. […] This model is close to what many departments and programs currently do, so it has the advantage of being well tested. The classes will generally be familiar to most faculty and instructional resources—such as syllabi, texts, and problem sets—are readily available.

*A compressed approach*

Again, to quote:

[…] For institutions that need to reduce the number of intermediate courses, the most straightforward approach is to combine individual topics into thematic courses that integrate material from related areas of computer science. […] this strategy also begins to address the problem of classes that focus too narrowly on “software artifacts.”

*A systems-based approach*

“Ultimately, the theories and practices of computer science find expression in the development of high-quality computer systems. This section defines a computer science curriculum that uses systems development as a unifying theme. It includes more technical and professional material than the other models, while retaining a reasonable level of coverage of the theoretical topics. Computer science theory remains essential, both as a foundation for understanding practice and to provide students with a lasting base of knowledge that remains valid despite changes in technology. […] This system perspective must permeate all aspects of the curriculum and include a combination of theory, practice, application, and attitudes.”

*A web-based approach*

“This model has grown out of a grass-roots demand for curricular structures that focus more attention on the Internet and the World-Wide Web, using these domains to serve as a common foundation for the curriculum as a whole.”

The original report went on to observe that it is possible to have approaches that are hybrid of the above models and it provided examples such as an illustration of a cross-cutting and a traditional model, a web-based and a compressed model as well as a hybrid of a traditional and compressed.

### 5.4.2 Alternative Organizing Principles

A number of alternative approaches used to organize the curriculum also exist. One set of possibilities stems from taking themes that are cross-cutting or systems wide; perhaps these can be seen as less abstract in concept and more closely related to particular topics. As such they can be used to motivate and possibly suggest or imply career directions.

A number of possibilities exist:

*Providing a security (or safety) focus*

Essentially security is a systems matter; for the system itself to be regarded as secure all major components typically need to be secure. Considerations of this kind have implications for programming (even elementary programming), information management, networks and communications, interfaces of all kinds, net-centric computing, tool building, and so on. So this theme can be used as a unifying concept for the curriculum. A similar set of comments can be made about the concept of safety.

*Focus on games or entertainment software*

The entertainment industry increasingly employs computing in many ways and the topic is of considerable interest to many young people. Developments in this world typically involve specialist hardware, specialist programming skills, deep attention to graphics and visualization, sophisticated interfaces often characterized by interaction and multi-media, aspects of artificial intelligence, etc. In many cases there are applications of simulation and modeling (e.g. of aspects of the physical world) and a reliance on topics such as games theory. These various considerations can be used to provide a theme on which to base an entire curriculum

*Focus on concurrency*

The increased presence of concurrency in hardware heightens the profile of this topic across the curriculum. Increasingly there is a need to pay attention to concurrent programming, models of computation, concurrent algorithms, the influence on compiler courses, on performance in searching, and so on. So again this can be used as theme running through the curriculum.

Other possibilities might include a focus on such topics as information management, graphics, human computer interaction, simulation and modeling, computing education, computer communications, mobility, etc. Again the idea of hybrid approaches can be adopted and opens up additional alternatives.

All of the approaches described in section 5.4 — the specific models and the various hybrids—all have a common goal: to present the fundamental ideas and enduring concepts of computer science that every student must learn to work successfully in the field. In doing so, these intermediate courses lay the foundation for more advanced work in computer science.

# Chapter 6 Reflections on the Computing Crisis

## 6.1 Background

Today's computer science programs grew out of the development of the mainframe computer in the US and Europe in the 1940s and 50s immediately after the Second World War. Computing received a boost with the advent of the microprocessor in the late 1970s and then the number of courses in computer science rapidly expanded.

Education in computer science has expanded in recent years, particularly in India and China. However, the number of students enrolling on computer science programs in countries where computer science education had hitherto been expanding (e.g., the US, Canada, the UK, Mexico) has begun to decline. This decline in student enrolment is taking place at a time when the demand for skilled professionals is increasing. The reduction in enrolment and the increase in demand have become so severe, that it is now referred to as the *Computing Crisis*.

In undertaking their work the Interim Review Task Force gave a great deal of consideration to the current crisis in computing, characterized by falling enrolments since 2000 and a shortfall in the number of graduates required to meet the national needs of some countries. The reasons for this situation are complex and involve many factors.

Future trends and developments in the enrolment of students cannot be predicted, nor can the demands of industry. The crisis may be resolved in two or three years or it may become worse in some countries. The Task Force has taken the view that it should comment on these trends in order to support and help the community. The problem just cannot be ignored.

The often-quoted observation that today's students find the computing curriculum unattractive is painful to those involved with higher education. Many factors play a role in the public image of computer science. In particular, students may develop a negative attitude towards computing as a result of poor pre-university computer science education which can give students the impression that computer science is synonymous with keyboard skills, word processing and the use of spreadsheets. Some students may expect computer science to be all about music and film production, web development and entertainment and find it difficult to relate to the mainstream computer science curriculum. Whatever the causes in declining enrolment in some countries, we cannot ignore this issue.

The often-quoted observation that today's students find the computing curriculum unattractive is painful to those involved with higher education. Many factors play a role in the public image of computer science. In particular, students may develop a negative attitude towards computing as a result of poor pre-university computer science education which can give students the impression that computer science is synonymous with keyboard skills, word processing and the use of spreadsheets. Some students may expect computer science to be all about music and film production, web development and entertainment and find it difficult to relate to the mainstream computer science curriculum. Whatever the causes in declining enrolment in some countries, we cannot ignore this issue.

There were a number of factors to be taken into account when considering the reduction in enrolment; some of which are:

- not to change the fundamental nature of the discipline, and not to over-simplify the curriculum and reduce the academic level of the discipline.
- to provide a curriculum that is interesting and attractive to current students, and equips them with skills and abilities to undertake a career in computing.
- to recognize that the computing crisis is a key issue within the computing community and this underpins much current research and experimentation. Moreover, it is important to encourage and support further endeavors in this regard.
- not to offer advice that is known to work in one specific environment but to offer advice of a generic nature so that different groups in the community will be encouraged to seek solutions appropriate for them and their students.

One way of tacking the decline in enrolment is to seek better and more innovative ways of delivering the curriculum. It is vital that we motivate and engage with the students in order to improve both recruitment and retention. The purpose of this chapter is of offer some general suggestions about this matter.

## 6.2 Application Domains

The range of computer applications is enormous and they cover every sphere of activity. Moreover, the range of applications grows wider with each day.

- computing and computers have been responsible for as much as 75% in the economic growth in the US since around 1995.
- the computer, through simulation and modeling, is the laboratory of the future for scientists. Computer simulation is found in applications ranging to the design of new drugs to the creation of energy efficient aircraft.
- education and entertainment have been heavily influenced by developments in computing. In a very short time, the digital camera has replaced an entire industry based on chemicals and specialized photographic processing. Similarly, the Internet has become a primary source of information for many and has replaced books and the written word.
- in the world of business computers play a huge role and now developments in enterprise architecture are tending to dictate the way in which organizations have to be organized.

All these observations imply that there is a rich and diverse set of evidence in the form of applications and developments that can be used to enliven the curriculum and so motivate students.

These applications of computers illustrate the importance of innovation in computing. Teaches have scope for building on innovation and incorporating it in their teaching materials. By drawing attention to innovation students can be motivated towards thinking about exciting developments for the future and the related benefits and challenges. There are related issues here about the important role that undergraduate research can play in the curriculum

## 6.3 Contextual Issues

The notion of context is important from a motivational and educational perspective. For instance, context can provide a framework in which to embed the teaching of computing. This can be at the level of

- a program of study such as games computing, multimedia computing, robotics, mobile computing, forensic computing, etc
- individual courses, that address topics such as safety, security, games programming, robotics, etc
- a group or cluster of courses that tackle a common theme such as ‘the computerized home’

Suitably choosing labels and titles can significantly influence the look and the feel of the curriculum.

Context can also help significantly in addressing issues such as innovation, reflecting the view that computing developments drive innovation and seeking to build on that. Again, there are different levels at which this might be addressed during lifetime of a program of study. A widely held view is that innovation is best taught in context, i.e. provide a student with a particular context in which to innovate.

As well as formal lectures, a course of study may include exercises, coursework, project work, etc. By applying a suitable context student may be motivated to invest effort and energy in their work. Similarly, students may be encouraged to undertake internships in different industries or organizations in order to increase their overall motivation.

## 6.4 Pedagogical Considerations

As in all disciplines there are approaches to teaching that inspire and encourage students. Such inspiration is particularly relevant in the early years of a course at a time when students are often coming to terms with the choices they face. In later years it is necessary to strike a balance between challenging students and yet building up their confidence and skill levels. Successful courses should emphasize the relevance of the taught work to career opportunities, providing exciting and imaginative exercises, giving regular and helpful feedback as well as encouragement are all crucial in influencing students.

Even in the context of teaching the syllabus there is scope for including observations about the wider applicability and the relevance of ideas from computing. The term ‘computational thinking’ has been coined to illustrate this concept. We need to develop and refine the notion of computational thinking. In some cases this may be a small step but yet it can serve over time to change perceptions about the discipline and to place it in a much more central position in terms of its wider relevance and its position in education for today's society

## 6.5 Final Observations

The Interim Review Task Force has adopted the view that it would be dangerous to be more prescriptive when considering solutions to the 'crisis in computing'. It was felt undesirable to select particular approaches here for special attention. However, the feeling is that we should encourage 'different flowers to blossom', that is, the community should be encouraged to experiment in their own particular local context so that they can find solutions that work for them.

Consequently, the Task Force has decided to leave some room in course prescriptions to allow individuals to customize, to provide this context, to experiment, and to motivate students

# Chapter 7 Concluding Comments

The Interim Review Task Force has wrestled with many aspects of its remit, some of the pressures creating conflict situations. The Task Force has been at pains not to suggest wholesale and radical change but to suggest revisions or modifications to the curriculum in certain key areas.

Within its various deliberations the Task Force identified the need for further deep thinking about the curriculum, about the manner in which that was presented and the manner in which computer science is taught. This reflected a perspective that there was considerable scope for presenting computer science in new and different ways that could be more in tune with the interests and the aspirations of today's generation of young people. It remains important to engage them, to challenge them, to direct them and to provide them with an education that would encourage them to be innovative and influential, and to contribute to the discipline and its well-being in some sense. But there is a need for greater clarity over what this means in practice and how the community should be advised.

The main value of a curriculum guideline document is that it is a guideline, not a prescription. That is, its main value is as a reference document that can help guide the development of a curriculum to meet local needs, or as a reference for evaluating an existing curriculum for possible improvement.

In short, many challenges remain for the curriculum designer.

## References

[Wing, 2006]

Wing, Jeannette M. “Computational Thinking”, Communications of the ACM, 49, 3, 2006, pp 33-35.

[Le Blanc and Sobel, 2006]

Rich Le Blanc, and Ann Sobel (chairs) et al., Software Engineering 2004: Curriculum guidelines for undergraduate degree programs in computer engineering, a volume of the Computing Curricula Series, copyright ACM and IEEE, published by the IEEE Computer Society, 2006.

[Soldan, et al., 2006]

Dave Soldan (chair) et al., Computer Engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering, a volume of the Computing Curricula Series copyright ACM and IEEE, published by the IEEE Computer Society, 2006.

[Shackelford, et al., 2005]

Russ Shackelford (chair) et al., Computing Curricula 2005: The Overview Report, a volume of the Computing Curricula series produced by the Joint Task Force for Computing Curricula 2005, copyright ACM and IEEE, published by the Association for Computing Machinery, 2006.

[Furst, et al., 2007]

Furst, M., Isbell, C., Guzdial, M. “THREADS: How to restructure a computer science curriculum for a flat world”, Proceedings of the Thirty-Eighth SIGCSE Technical Symposium on Computer Science Education, March 7 -10, 2007, pp 420-424.

[Marion and Baldwin, 2007]

Marion, B., and Baldwin, D. (chairs), SIGCSE Committee Report on the Implementation of a Discrete Mathematics Course, ACM, April 2007.

[Howe, et al., 2004]

Howe, E., Thornton, M., and Weide, B.W., "Components-First Approaches to CS1/CS2: Principles and Practice", Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, ACM Press, 2004, pp 291-295.

[Pedroni and Meyer, 2006]

Pedroni, M., and Meyer, B., "The Inverted Curriculum in Practice", Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, ACM Press, 2006, pp 481-485.

# Appendix A Overview of the Body of Knowledge

**DS. Discrete Structures (43 core hours)**
DS/FunctionsRelationsAndSets (6)
DS/BasicLogic (10)
DS/ProofTechniques (12)
DS/BasicsOfCounting (5)
DS/GraphsAndTrees (4)
DS/DiscreteProbability (6)

**PF. Programming Fundamentals (33 core hours)**
PF/FundamentalConstructs (9)
PF/AlgorithmicProblemSolving (6)
PF/DataStructures (10)
PF/Recursion (4)
PF/EventDrivenProgramming (4)
PF/ObjectOriented (8)
PF/FoundationsInformationSecurity (2)
PF/SecureProgramming (4)

**AL. Algorithms and Complexity (31 core hours)**
AL/BasicAnalysis (4)
AL/AlgorithmicStrategies (6)
AL/FundamentalAlgorithms (12)
AL/DistributedAlgorithms (3)
AL/BasicComputability (6)
AL/PversusNP
AL/AutomataTheory
AL/AdvancedAnalysis
AL/CryptographicAlgorithms
AL/GeometricAlgorithms
AL/ParallelAlgorithms

**AR. Architecture and Organization (36 core hours)**
AR/DigitalLogic (7)
AR/DataRepresentation (9)
AR/AssemblyLevelOrganization (3)
AR/MemoryArchitecture (5)
AR/FunctionalOrganization (6)
AR/Multiprocessing (6)
AR/PerformanceEnhancements
AR/DistributedArchitectures
AR/Devices
AR/DirectionsInComputing

**OS. Operating Systems (18 core hours)**
OS/OverviewOfOperatingSystems (2)
OS/OperatingSystemPrinciples (2)
OS/Concurrency (6)
OS/Scheduling and dispatch (3)
OS/MemoryManagement (5)
OS/DeviceManagement
OS/SecurityAndProtection
OS/FileSystems
OS/RealTimeAndEmbeddedSystems
OS/FaultTolerance
OS/SystemPerformanceEvaluation
OS/Scripting
OS/DigitalForensics
OS/SecurityModels

**NC. Net-Centric Computing (18 core hours)**
NC/Introduction(2)
NC/NetworkCommunication (7)
NC/NetworkSecurity (6)
NC/WebOrganization
NC/NetworkedApplications
NC/NetworkManagement
NC/Compression
NC/MultimediaTechnologies
NC/MobileComputing

**PL. Programming Languages (21 core hours)**
PL/Overview(2)
PL/VirtualMachines(1)
PL/BasicLanguageTranslation(2)
PL/DeclarationsAndTypes(3)
PL/AbstractionMechanisms(3)
PL/ObjectOrientedProgramming(10)
PL/FunctionalProgramming
PL/LanguageTranslationSystems
PL/TypeSystems
PL/ProgrammingLanguageSemantics
PL/ProgrammingLanguageDesign

**HC. Human-Computer Interaction (8 core hours)**
HC/Foundations (6)
HC/BuildingGUIInterfaces (2)
HC/UserCcenteredSoftwareEvaluation
HC/UserCenteredSoftwareDevelopment
HC/GUIDesign
HC/GUIProgramming
HC/MultimediaAndMultimodalSystems
HC/CollaborationAndCommunication
HC/InteractionDesignForNewEnvironments
HC/HumanFactorsAndSecurity

**GV. Graphics and Visual Computing (3 core hours)**
GV/FundamentalTechniques (2)
GV/GraphicSystems (1)
GV/GraphicCommunication
GV/GeometricModeling
GV/BasicRendering
GV/AdvancedRendering
GV/AdvancedTechniques
GV/ComputerAnimation
GV/Visualization
GV/VirtualReality
GV/ComputerVision
GV/ComputationalGeometry
GV/GameEngineProgramming

**IS. Intelligent Systems (10 core hours)**
IS/FundamentalIssues (1)
IS/BasicSearchStrategies (5)
IS/KnowledgeBasedReasoning (4)
IS/AdvancedSearch
IS/AdvancedReasoning
IS/Agents
IS/NaturaLanguageProcessing
IS/MachineLearning
IS/PlanningSystems
IS/Robotics
IS/Perception

**IM. Information Management (11 core hours)**
IM/InformationModels (4)
IM/DatabaseSystems (3)
IM/DataModeling (4)
IM/Indexing
IM/RelationalDatabases
IM/QueryLanguages
IM/RelationalDatabaseDesign
IM/TransactionProcessing
IM/DistributedDatabases
IM/PhysicalDatabaseDesign
IM/DataMining
IM/InformationStorageAndRetrieval
IM/Hypermedia
IM/MultimediaSystems
IM/DigitalLibraries

**SP. Social and Professional Issues (16 core hours)**
SP/HistoryOfComputing (1)
SP/SocialContext (3)
SP/AnalyticalTools (2)
SP/ProfessionalEthics (3)
SP/Risks (2)
SP/SecurityOperations
SP/IntellectualProperty (3)
SP/PrivacyAndCivilLiberties (2)
SP/ComputerCrime
SP/EconomicsOfComputing
SP/PhilosophicalFrameworks

**SE. Software Engineering (31 core hours)**
SE/SoftwareDesign (8)
SE/UsingAPIs (5)
SE/ToolsAndEnvironments (3)
SE/SoftwareProcesses (2)
SE/RequirementsSpecifications (4)
SE/SoftwareValidation (3)
SE/SoftwareEvolution (3)
SE/SoftwareProjectManagement (3)
SE/ComponentBasedComputing
SE/FormalMethods
SE/SoftwareReliability
SE/SpecializedSystems
SE/RiskAssessment

**CN. Computational Science (no core hours)**
CN/ModelingAndSimulation
CN/OperationsResearch
CN/ParallelComputation

Note: The numbers in parentheses represent the minimum number of hours required to cover this material in a lecture format. It is always appropriate to include more.

# Appendix B Detailed Body of Knowledge

# Discrete Structures (DS)

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

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

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

Finally, we note that while areas often have somewhat fuzzy boundaries, this is especially true for discrete structures. We have gathered together here a body of material of a mathematical nature that computer science education must include, and that computer science educators know well enough to specify in great detail. However, the decision about where to draw the line between this area and the Algorithms and Complexity area (AL) on the one hand, and topics left only as supporting mathematics on the other hand, was inevitably somewhat arbitrary. We remind readers that there are vital topics from those two areas that some schools will include in courses with titles like "discrete structures" and "discrete mathematics"; some will require one course, others two. In April 2007, the SIGCSE Committee on the Implementation of a Discrete Mathematics Course released a report detailing three models for a one-semester discrete mathematics course to meet the criteria articulated in CS2001; these models remain applicable under the slightly revised suggestions in this interim report. See SIGCSE Committee Report On the Implementation of a Discrete Mathematics Course.

**DS. Discrete Structures (43 core hours)**

**DS/FunctionsRelationsAndSets [core]**
**DS/BasicLogic [core]**
**DS/ProofTechniques [core]**
**DS/BasicsOfCounting [core]**
**DS/GraphsAndTrees [core]**
**DS/DiscreteProbability [core]**

## DS/FunctionsRelationsAndSets [core]

*Minimum core coverage time: 6 hours*

*Topics:*

- Functions (surjections, injections, inverses, composition)
- Relations (reflexivity, symmetry, transitivity, equivalence relations)
- Sets (Venn diagrams, complements, Cartesian products, power sets)
- Pigeonhole principle
- Cardinality and countability

*Learning objectives:*

1. Explain with examples the basic terminology of functions, relations, and sets.
2. Perform the operations associated with sets, functions, and relations.
3. Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
4. Demonstrate basic counting principles, including uses of diagonalization and the pigeonhole principle.

## DS/ProofTechniques [core]

*Minimum core coverage time: 12 hours*

*Topics:*
- Notions of implication, converse, inverse, contrapositive, negation, and contradiction
- The structure of mathematical proofs
- Direct proofs
- Proof by counterexample
- Proof by contraposition
- Proof by contradiction
- Mathematical induction
- Strong induction
- Recursive mathematical definitions
- Well orderings

*Learning objectives:*
1. Outline the basic structure of and give examples of each proof technique described in this unit.
2. Discuss which type of proof is best for a given problem.
3. Relate the ideas of mathematical induction to recursion and recursively defined structures.
4. Identify the difference between mathematical and strong induction and give examples of the appropriate use of each.

## DS/BasicsOfCounting [core]

*Minimum core coverage time: 5 hours*

*Topics:*
- Counting arguments
- Sum and product rule
- Inclusion-exclusion principle
- Arithmetic and geometric progressions
- Fibonacci numbers
- The pigeonhole principle
- Permutations and combinations
- Basic definitions
- Pascal's identity
- The binomial theorem
- Solving recurrence relations
- Common examples
- The Master theorem

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

## DS/GraphsAndTrees [core]

*Minimum core coverage time: 4 hours*

*Topics:*
- Trees
- Undirected graphs
- Directed graphs
- Spanning trees/forests
- Traversal strategies

*Learning objectives:*

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

## DS/GraphsAndTrees [core]
*Minimum core coverage time: 4 hours*

*Topics:*
- Trees
- Undirected graphs
- Directed graphs
- Spanning trees/forests
- Traversal strategies

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

## DS/DiscreteProbability [core]
*Minimum core coverage time: 6 hours*

*Topics:*
- Finite probability space, probability measure, events
- Conditional probability, independence, Bayes' theorem
- Integer random variables, expectation
- Law of large numbers

*Learning objectives:*
1. Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance.
2. Differentiate between dependent and independent events.
3. Apply the binomial theorem to independent events and Bayes' theorem to dependent events.
4. Apply the tools of probability to solve problems in areas such as the Monte Carlo method, the average case analysis of algorithms, and hashing.

## Programming Fundamentals (PF)

Fluency in a programming language is prerequisite to the study of most of computer science. Undergraduate computer science programs must teach students how to use at least one programming language well; furthermore, computer science programs should teach students to become competent in languages that make use of the object-oriented and event-driven programming paradigms.

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

**PF. Programming Fundamentals (33 core hours)**

**PF/FundamentalConstructs [core]**
**PF/AlgorithmicProblemSolving [core]**
**PF/DataStructures [core]**
**PF/Recursion [core]**
**PF/EventDrivenProgramming [core]**
**PF/ObjectOriented [core]**
**PF/FoundationsInformationSecurity [core]**
**PF/SecureProgramming [core]**

### PF/FundamentalConstructs [core]

*Minimum core coverage time: 9 hours*

*Topics:*

Basic syntax and semantics of a higher-level language
Variables, types, expressions, and assignment
Simple I/O
Conditional and iterative control structures
Functions and parameter passing
Structured decomposition

*Learning objectives:*

Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.
Modify and expand short programs that use standard conditional and iterative control structures and functions.
Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.
Choose appropriate conditional and iteration constructs for a given programming task.
Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
Describe the mechanics of parameter passing.

### PF/AlgorithmicProblemSolving [core]

*Minimum core coverage time: 6 hours*

*Topics:*

- Problem-solving strategies
- The role of algorithms in the problem-solving process
- Implementation strategies for algorithms

- Debugging strategies
- The concept and properties of algorithms

*Learning objectives:*
1. Discuss the importance of algorithms in the problem-solving process.
2. Identify the necessary properties of good algorithms.
3. Create algorithms for solving simple problems.
4. Use pseudocode or a programming language to implement, test, and debug algorithms for solving simple problems.
5. Describe strategies that are useful in debugging.

## PF/DataStructures [core]
*Minimum core coverage time: 10 hours*

*Topics:*
- Representation of numeric data
- Range, precision, and rounding errors
- Arrays
- Representation of character data
- Strings and string processing
- 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. Describe the representation of numeric and character data.
2. Understand how precision and round-off can affect numeric calculations.
3. Discuss the use of primitive data types and built-in data structures.
4. Describe common applications for each data structure in the topic list.
5. Implement the user-defined data structures in a high-level language.
6. Compare alternative implementations of data structures with respect to performance.
7. Write programs that use each of the following data structures: arrays, strings, linked lists, stacks, queues, and hash tables.
8. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
9. Choose the appropriate data structure for modeling a given problem.

## PF/Recursion [core]
*Minimum core coverage time: 4 hours*

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

*Learning objectives:*
1. Describe the concept of recursion and give examples of its use.
2. Identify the base case and the general case of a recursively defined problem.
3. Compare iterative and recursive solutions for elementary problems such as factorial.
4. Describe the divide-and-conquer approach.
5. Implement, test, and debug simple recursive functions and procedures.
6. Determine when a recursive solution is appropriate for a problem.

## PF/EventDriven [core]
*Minimum core coverage time: 4 hours*

*Topics:*
- Event-handling methods
- Event propagation
- Exception handling

*Learning objectives:*
1. Explain the difference between event-driven programming and command-line programming.
2. Design, code, test, and debug simple event-driven programs that respond to user events.
3. Develop code that responds to exception conditions raised during execution.

## PF/ObjectOriented [core]
*Minimum core coverage time: 8 hours*

*Topics:*
- Object-oriented design
- Encapsulation and information-hiding
- Separation of behavior and implementation
- Classes and subclasses
- Inheritance (overriding, dynamic dispatch)
- Polymorphism (subtype polymorphism vs. inheritance)

*Learning objectives:*
1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism.
2. Design, implement, test, and debug simple programs in an object-oriented programming language.
3. Describe how the class mechanism supports encapsulation and information hiding.
4. Design, implement, and test the implementation of "is-a" relationships among objects using a class hierarchy and inheritance.
5. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.

## PF/FoundationsInformationSecurity [core]
*Minimum core coverage time: 4 hours*

*Topics:*
- Role and purpose of computer and network security
- Security goals: confidentiality, integrity, availability triad
- Security standards and policies
- Security mindset
- Defense in depth
- Common threats: worms, viruses, trojans, denial of service
- Risk assessment and cost-benefit analyses
- Security versus usability, time, and/or  money tradeoffs

*Learning objectives:*
1. Explain the objectives of information security
2. Analyze the tradeoffs inherent in security
3. Explain the importance and application of each of confidentiality, integrity, and availability
4. Understand the basic categories of threats to computers and networks
5. Discuss issues for creating security policy for a large organization
6. Defend the need for protection and security, and the role of ethical considerations in computer use
7. Add a very simple risk-assessment learning outcome here

## PF/SecureProgramming [core]
*Minimum core coverage time: 2 hour*

*Topics*
- Important of checking for and avoiding array and string overflows
- Programming language constructs to avoid and alternatives

- How attackers use overflows to smash the run-time stack

*Learning objectives:*
1. Rewrite a simple program to remove a simple vulnerability
2. Explain why or why not a buffer overflow is possible in the programming language you know best
3. Explain why one or more language constructs may lead to security problems such as overflows.

# Algorithms and Complexity

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

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

With the emergence of multicore processors, issues related to parallel algorithms have become more relevant. While the parallelism topics remain listed here as elective, the committee believes that the role of parallelism throughout the curriculum needs to be considered.

**AL. Algorithms and Complexity (31 core hours)**

**AL/BasicAnalysis [core]**
**AL/AlgorithmicStrategies [core]**
**AL/FundamentalAlgorithms [core]**
**AL/DistributedAlgorithms [core]**
**AL/BasicComputability [core]**
**AL/PversusNP [elective]**
**AL/AutomataTheory [elective]**
**AL/AdvancedAnalysis [elective]**
**AL/CryptographicAlgorithms [elective]**
**AL/GeometricAlgorithms [elective]**
**AL/ParallelAlgorithms [elective]**

## AL/BasicAnalysis [core]

*Minimum core coverage time: 4 hours*

*Topics:*

- Asymptotic analysis of upper and average complexity bounds
- Identifying differences among best, average, and worst case behaviors
- Big O, little o, omega, and theta notation
- Standard complexity classes
- Empirical measurements of performance
- Time and space tradeoffs in algorithms
- Using recurrence relations to analyze recursive algorithms

*Learning objectives:*

1. Explain the use of big O, omega, and theta notation to describe the amount of work done by an algorithm.
2. Use big O, omega, and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.
3. Determine the time and space complexity of simple algorithms.
4. Deduce recurrence relations that describe the time complexity of recursively defined algorithms.
5. Solve elementary recurrence relations.

## AL/AlgorithmicStrategies [core]

*Minimum core coverage time: 6 hours*

*Topics:*

- Brute-force algorithms
- Greedy algorithms
- Divide-and-conquer

- Backtracking
- Branch-and-bound
- Heuristics
- Pattern matching and string/text algorithms
- Numerical approximation algorithms

*Learning objectives:*
1. Describe the shortcoming of brute-force algorithms.
2. For each of several kinds of algorithm (brute force, greedy, divide-and-conquer, backtracking, branch-and-bound, and heuristic), identify an example of everyday human behavior that exemplifies the basic concept.
3. Implement a greedy algorithm to solve an appropriate problem.
4. Implement a divide-and-conquer algorithm to solve an appropriate problem.
5. Use backtracking to solve a problem such as navigating a maze.
6. Describe various heuristic problem-solving methods.
7. Use pattern matching to analyze substrings.
8. Use numerical approximation to solve mathematical problems, such as finding the roots of a polynomial.

## AL/FundamentalAlgorithms [core]
*Minimum core coverage time: 12 hours*

*Topics:*
- Simple numerical algorithms
- Sequential and binary search algorithms
- Quadratic sorting algorithms (selection, insertion)
- O(N log N) sorting algorithms (Quicksort, heapsort, mergesort)
- Hash tables, including collision-avoidance strategies
- Binary search trees
- Representations of graphs (adjacency list, adjacency matrix)
- Depth- and breadth-first traversals
- Shortest-path algorithms (Dijkstra's and Floyd's algorithms)
- Transitive closure (Floyd's algorithm)
- Minimum spanning tree (Prim's and Kruskal's algorithms)
- Topological sort

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

## AL/BasicComputability [core]
*Minimum core coverage time: 6 hours*

*Topics:*
- Finite-state machines
- Context-free grammars
- Tractable and intractable problems
- Uncomputable functions
- The halting problem
- Implications of uncomputability

*Learning objectives:*

1. Discuss the concept of finite state machines.
2. Explain context-free grammars.
3. Design a deterministic finite-state machine to accept a specified language.
4. Explain how some problems have no algorithmic solution.
5. Provide examples that illustrate the concept of uncomputability.

## AL/PversusNP [elective]

*Topics:*
- Definition of the classes P and NP
- NP-completeness (Cook's theorem)
- Standard NP-complete problems
- Reduction techniques

*Learning objectives:*
1. Define the classes P and NP.
2. Explain the significance of NP-completeness.
3. Prove that a problem is NP-complete by reducing a classic known NP-complete problem to it.

## AL/AutomataTheory [elective]

*Topics:*
- Deterministic finite automata (DFAs)
- Nondeterministic finite automata (NFAs)
- Equivalence of DFAs and NFAs
- Regular expressions
- The pumping lemma for regular expressions
- Push-down automata (PDAs)
- Relationship of PDAs and context-free grammars
- Properties of context-free grammars
- Turing machines
- Nondeterministic Turing machines
- Sets and languages
- Chomsky hierarchy
- The Church-Turing thesis

*Learning objectives:*
1. Determine a language's location in the Chomsky hierarchy (regular sets, context-free, context-sensitive, and recursively enumerable languages).
2. Prove that a language is in a specified class and that it is not in the next lower class.
3. Convert among equivalently powerful notations for a language, including among DFAs, NFAs, and regular expressions, and between PDAs and CFGs.
4. Explain at least one algorithm for both top-down and bottom-up parsing.
5. Explain the Church-Turing thesis and its significance.

## AL/AdvancedAnalysis [elective]

*Topics:*
- Amortized analysis
- Online and offline algorithms
- Randomized algorithms
- Dynamic programming
- Combinatorial optimization

*Learning objectives:*
1. Use the potential method to provide an amortized analysis of previously unseen data structure, given the potential function.
2. Explain why competitive analysis is an appropriate measure for online algorithms.

3. Explain the use of randomization in the design of an algorithm for a problem where a deterministic algorithm is unknown or much more difficult.
4. Design and implement a dynamic programming solution to a problem.

## AL/CryptographicAlgorithms [elective]

*Topics:*
- Historical overview of cryptography
- Private-key cryptography and the key-exchange problem
- Public-key cryptography
- Digital signatures
- Security protocols
- Applications (zero-knowledge proofs, authentication, and so on)

*Learning objectives:*
1. Describe efficient basic number-theoretic algorithms, including greatest common divisor, multiplicative inverse mod n, and raising to powers mod n.
2. Describe at least one public-key cryptosystem, including a necessary complexity-theoretic assumption for its security.
3. Create simple extensions of cryptographic protocols, using known protocols and cryptographic primitives.

## AL/GeometricAlgorithms [elective]

*Topics:*
- Line segments: properties, intersections
- Convex hull finding algorithms

*Learning objectives:*
1. Describe and give time analysis of at least two algorithms for finding a convex hull.
2. Justify the Omega(N log N) lower bound on finding the convex hull.
3. Describe at least one additional efficient computational geometry algorithm, such as finding the closest pair of points, convex layers, or maximal layers.

## AL/ParallelAlgorithms [elective]

*Topics:*
- PRAM model
- Exclusive versus concurrent reads and writes
- Pointer jumping
- Brent’s theorem and work efficiency

*Learning objectives:*
1. Describe implementation of linked lists on a PRAM.
2. Use parallel-prefix operation to perform simple computations efficiently in parallel.
3. Explain Brent’s theorem and its relevance.

See also NetCentric computing for algorithm topics related to security, cryptography and compression.

# Architecture and Organization

The computer lies at the heart of computing. Without it most of the computing disciplines today would be a branch of theoretical mathematics. A professional in any field of computing should not regard the computer as just a black box that executes programs by magic. All students of computing should acquire some understanding and appreciation of a computer system's functional components, their characteristics, their performance, and their interactions. Students need to understand computer architecture in order to make best use of the software tools and computer languages they use to create programs. In this introduction the term architecture is taken to include instruction set architecture (the programmer's abstraction of a computer), organization or microarchitecture (the internal implementation of a computer at the register and functional unit level), and system architecture (the organization of the computer at the cache, and bus level). Students should also understand the complex tradeoffs between CPU clock speed, cache size, bus organization, number of core processors, and so on. Computer architecture also underpins other areas of the computing curriculum such as operating systems (input/output, memory technology) and high-level languages (pointers, parameter passing).

The learning outcomes specified for these topics correspond primarily to the core and are intended to support programs that require only the minimum 36 hours of computer architecture. For programs that want to teach more than the minimum, the same topics (AR1-AR6) can be treated at a more advanced level by implementing a two-course sequence. For programs that want to cover the elective topics, those topics can be introduced within a two-course sequence and/or be treated in a more comprehensive way in a third course

**AR. Architecture and Organization (36 core hours)**

- **AR/DigitalLogic [core]**
- **AR/DataRepresentation [core]**
- **AR/AssemblyLevelOrganization[core]**
- **AR/MemoryArchitecture [core]**
- **AR/FunctionalOrganization [core]**
- **AR/Multiprocessing [core]**
- **AR/PerformanceEnhancements [elective]**
- **AR/DistributedArchitectures [elective]**
- **AR/Devices [elective]**
- **AR/DirectionsInComputing [elective]**

## AR/Digital Logic and Data Representation [core]

*Minimum core coverage time: 7 hours*
(was Digital logic and digital systems AR1 and Machine level representation of data AR2. )

*Topics:*

- Introduction to digital logic (logic gates, flip-flops, circuits)
- Logic expressions and Boolean functions
- Representation of numeric data
- Signed and unsigned arithmetic
- Range, precision, and errors in floating-point arithmetic
- Representation of text, audio, and images
- Data compression

*Learning objectives:*

1. Design a simple circuit using fundamental building blocks.
2. Appreciate the effect of AND, OR, NOT and EOR operations on binary data
3. Understand how numbers, text, images, and sound can be represented in digital form and the limitations of such representations
4. Understand how errors due to rounding effects and their propagation affect the accuracy of chained calculations.
5. Appreciate how data can be compressed to reduce storage requirements including the concepts of lossless and lossy compression.

## AR/Computer Architecture and Organization [core]

*Minimum core coverage time: 9 hours*

(was Assembly level machine organization AR3)

*Topics:*

- Overview of the history of the digital computer
- Introduction to instruction set architecture, microarchitecture and system architecture
- Processor architecture – instruction types, register sets, addressing modes
- Processor structures – memory-to-register and load/store architectures
- Instruction sequencing, flow-of-control, subroutine call and return mechanisms
- Structure of machine-level programs
- Limitations of low-level architectures
- Low-level architectural support for high-level languages

*Learning objectives:*

1. Describe the progression of computers from vacuum tubes to VLSI.
2. Appreciate the concept of an instruction set architecture, ISA, and the nature of a machine-level instruction in terms of its functionality and use of resources (registers and memory).
3. To understand the relationship between instruction set architecture, microarchitecture, and system architecture and their roles in the development of the computer.
4. Be aware of the various classes of instruction: data movement, arithmetic, logical, and flow control.
5. Appreciate the difference between register-to-memory ISAs and load/store ISAs.
6. Appreciate how conditional operations are implemented at the machine level.
7. Understand the way in which subroutines are called and returns made.
8. Appreciate how a lack of resources in ISPs has an impact on high-level languages and the design of compilers.
9. Understand how, at the assembly language level, how parameters are passed to subroutines and how local workplace is created and accessed.

## AR/Interfacing and I/O Strategies [core]

*Minimum core coverage time: 3 hours*

(was Interfacing and communication AR5)

*Topics:*

- I/O fundamentals: handshaking and buffering
- Interrupt mechanisms: vectored and prioritized, interrupt acknowledgment
- Buses: protocols, arbitration, direct-memory access (DMA)
- Examples of modern buses: e.g., PCIe, USB, Hypertransport

*Learning objectives:*

1. Appreciate the need of open- and closed-loop communications and the use of buffers to control dataflow.
2. Explain how interrupts are used to implement I/O control and data transfers.
3. Identify various types of buses in a computer system and understand how devices compete for a bus and are granted access to the bus.
4. Be aware of the progress in bus technology and understand the features and performance of a range of modern buses (both serial and parallel).

## AR/MemoryArchitecture [core]

*Minimum core coverage time: 5 hours*

(was Memory system organization AR4)

*Topics:*

- Storage systems and their technology (semiconductor, magnetic)
- Storage standards (CD-ROM, DVD)
- Memory hierarchy, latency and throughput
- Cache memories - operating principles, replacement policies, multilevel cache, cache coherency

*Learning objectives:*

1. Identify the memory technologies found in a computer and be aware of the way in which memory technology is changing.
2. Appreciate the need for storage standards for complex data storage mechanisms such as DVD.
3. Understand why a memory hierarchy is necessary to reduce the effective memory latency.
4. Appreciate that most data on the memory bus is cache refill traffic
5. Describe the various ways of organizing cache memory and appreciate the cost-performance tradeoffs for each arrangement.
6. Appreciate the need for cache coherency in multiprocessor systems

## AR/FunctionalOrganization [core]

*Minimum core coverage time: 6 hours*
(was Functional organization AR6)

*Topics:*

- Review of register transfer language to describe internal operations in a computer
- Microarchitectures - hardwired and microprogrammed realizations
- Instruction pipelining and instruction-level parallelism (ILP)
- Overview of superscalar architectures
- Processor and system performance
- Performance – their meeasures and their limitations
- The significance of power dissipation and its effects on computing structures

*Learning objectives:*

1. Review of the use of register transfer language to describe internal operations in a computer
2. Understand how a CPU's control unit interprets a machine-level instruction – either directly or as a microprogram.
3. Appreciate how processor performance can be improved by overlapping the execution of instruction by pipelining.
4. Understand the difference between processor performance and system performance (i.e., the effects of memory systems, buses and software on overall performance).
5. Appreciate how superscalar architectures use multiple arithmetic units to execute more than one instruction per clock cycle.
6. Understand how computer performance is measured by measurements such as MIPS or SPECmarks and the limitations of such measurements.
7. Appreciate the relationship between power dissipation and computer performance and the need to minimize power consumption in mobile applications.

## AR/Multiprocessing [core]

*Minimum core coverage time:* 6 hours
(was Multiprocessing and alternative architectures AR7)

*Topics:*

- Amdahl's law
- Short vector processing (multimedia operations)
- Multicore and multithreaded processors
- Flynn's taxonomy: Multiprocessor structures and architectures
- Programming multiprocessor systems
- GPU and special-purpose graphics processors
- Introduction to reconfigurable logic and special-purpose processors

*Learning objectives:*

1. Discuss the concept of parallel processing and the relationship between parallelism and performance.
2. Appreciate that multimedia values (e.g., 8-/16-bit audio and visual data) can be operated on in parallel in 64-bit registers to enhance performance.
3. Understand how performance can be increased by incorporating multiple processors on a single chip.
4. Appreciate the need to express algorithms in a form suitable for execution on parallel processors.
5. Understand how special-purpose graphics processors, GPUs, can accelerate performance in graphics applications.
6. Understand the organization of computer structures that can be electronically configured and reconfigured

## AR/PerformanceEnhancements [elective]

(was performance enhancements AR8)

*Topics:*

- Branch prediction
- Speculative execution
- Superscalar architecture
- Out-of-order execution
- Multithreading
- Scalability
- Introduction to VLIW and EPIC architectures
- Memory access ordering

*Learning objectives:*

1. Explain the concept of branch prediction its use in enhancing the performance of pipelined machines.
2. Understand how speculative execution can improve performance.
3. Provide a detailed description of superscalar architectures and the need to ensure program correctness when executing instructions out-of-order.
4. Explain speculative execution and identify the conditions that justify it.
5. Discuss the performance advantages that multithreading can offer along with the factors that make it difficult to derive maximum benefits from this approach.
6. Appreciate the nature of VLIW and EPIC architectures and the difference between them (and between superscalar processors)
7. Understand how a processor re-orders memory loads and stores to increase performance

## AR/DistributedArchitectures [elective]

(was Architecture for networks and distributed systems AR9)

*Topics:*

- Introduction to LANs and WANs and the history of networking and the Internet
- Layered protocol design, network standards and standardization bodies
- Network computing and distributed multimedia
- Mobile and wireless computing
- Streams and datagrams
- Physical layer networking concepts
- Data link layer concepts (framing, error control, flow control, protocols)
- Internetworking and routing (routing algorithms, internetworking, congestion control)
- Transport layer services (connection establishment, performance issues)

*Learning objectives:*

1. Explain the basic components of network systems and distinguish between LANs and WANs.
2. Discuss the architectural issues involved in the design of a layered network protocol.
3. Explain how architectures differ in network and distributed systems.
4. Appreciate the special requirements of wireless computing.
5. Understand the difference between the roles of the physical layer and data link layer and appreciate how imperfections in the physical layer are handled by the data link layer.
6. Describe emerging technologies in the net-centric computing area and assess their current capabilities, limitations, and near-term potential.
7. Understand how the network layer can detect and correct errors.

## AR/Devices [elective]

(new material)

*Topics:*

- Representing analog values digitally – quantization and sampling
- Sound and audio, image and graphics, animation and video
- Multimedia standards (audio, music, graphics, image, telephony, video, TV)
- Input transducers (temperature, pressure, position, movement)
- Input devices: mice, keyboards (text and musical), scanners, touch-screen, voice

- Output devices: displays, printers
- Encoding and decoding of multimedia systems including compression and decompression
- Example of computer-based systems: GPS, MP3 players, digital cameras

*Learning objectives:*

1. Understand how analog quantities such as pressure can be represented in digital form and how the use of a finite representation leads to quantization errors.
2. Appreciate the need for multimedia standards and be able to explain in non-technical language what the standard calls for.
3. Understand how multimedia signals usually have to be compressed to conserve bandwidths using lossless or lossy encoding.
4. Discuss the design, construction, and operating principles of transducers such as Hall-effect devices and strain gauges
5. Appreciate how typical input devices operate.
6. Understand the principles of operation and performance of various display devices.
7. Study the operation of high-performance computer-based devices such as digital cameras

## AR/Directions in Computing [elective]

*7 hours* (New topic)

*Topics:*

- Semiconductor technology and Moore's law
- Limitations to semiconductor technology
- Quantum computing
- Optical computing
- Molecular (biological) computing
- New memory technologies

*Learning objectives:*

1. To appreciate the underlying physical basic of modern computing.
2. Understand how the physical properties of matter impose limitations on computer technology
3. Appreciate how the quantum nature of matter can be exploited to permit massive parallelism
4. Appreciate how light can be used to perform certain types of computation
5. Understand how the properties of complex molecules can be exploited by organic computers
6. To get an insight into trends in memory design such as ovonic memory and ferromagnetic memories

# Operating Systems

An operating system defines an abstraction of hardware behavior with which programmers can control the hardware. It also manages resource sharing among the computer's users. The topics in this area explain the issues that influence the design of contemporary operating systems. Courses that cover this area will typically include a laboratory component to enable students to experiment with operating systems.

Over the years, operating systems and their abstractions have become complex relative to typical application software. It is necessary to ensure that the student understands the extent of the use of an operating system prior to a detailed study of internal implementation algorithms and data structures. Therefore these topics address 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 science, such as concurrent programming. Studying internal design has relevance in such diverse areas as dependable programming, algorithm design and implementation, modern device development, building virtual environments, caching material across the web, building secure and safe systems, network management, and many others.

**OS. Operating Systems (18 core hours)**

**OS/OverviewOfOperatingSystems [core]**
**OS/OperatingSystemPrinciples [core]**
**OS/Concurrency [core]**
**OS/Scheduling and dispatch [core]**
**OS/MemoryManagement [core]**
**OS/DeviceManagement [elective]**
**OS/SecurityAndProtection [elective]**
**OS/FileSystems [elective]**
**OS/RealTimeAndEmbeddedSystems [elective]**
**OS/FaultTolerance [elective]**
**OS/SystemPerformanceEvaluation [elective]**
**OS/Scripting [elective]**
**OS/DigitalForensics [elective]**
**OS/SecurityModels [elective]**

## OS/OverviewOfOperatingSystems [core]

*Minimum core coverage time: 2 hours*
*Topics:*

- Role and purpose of the operating system
- History of operating system development
- 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:*

1. Explain the objectives and functions of modern operating systems.
2. Describe how operating systems have evolved over time from primitive batch systems to sophisticated multiuser systems.
3. Analyze the tradeoffs inherent in operating system design.
4. Describe the functions of a contemporary operating system with respect to convenience, efficiency, and the ability to evolve.
5. Discuss networked, client-server, distributed operating systems and how they differ from single user operating systems.
6. Identify potential threats to operating systems and the security features design to guard against them.
7. Describe how issues such as open source software and the increased use of the Internet are influencing operating system design.

## OS/OperatingSystemPrinciples [core]

*Minimum core coverage time: 2 hours*

*Topics:*
- Structuring methods (monolithic, layered, modular, micro-kernel models)
- Abstractions, processes, and resources
- Concepts of application program interfaces (APIs)
- Application 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. Explain the concept of a logical layer.
2. Explain the benefits of building abstract layers in hierarchical fashion.
3. Defend the need for APIs and middleware.
4. Describe how computing resources are used by application software and managed by system software.
5. Contrast kernel and user mode in an operating system.
6. Discuss the advantages and disadvantages of using interrupt processing.
7. Compare and contrast the various ways of structuring an operating system such as object-oriented, modular, micro-kernel, and layered.
8. Explain the use of a device list and driver I/O queue.

## OS/Concurrency [core]
*Minimum core coverage 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. Describe the need for concurrency within the framework of an operating system.
2. Demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks.
3. Summarize the range of mechanisms that can be employed at the operating system level to realize concurrent systems and 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.
5. Summarize the various approaches to solving the problem of mutual exclusion in an operating system.
6. Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system.
7. Create state and transition diagrams for simple problem domains.
8. Discuss the utility of data structures, such as stacks and queues, in managing concurrency.
9. Explain conditions that lead to deadlock.

## OS/SchedulingAndDispatch [core]
*Minimum core coverage time: 3 hours*

*Topics:*

Preemptive and nonpreemptive scheduling
Schedulers and policies
Processes and threads
Deadlines and real-time issues

*Learning objectives:*

Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes.
Describe relationships between scheduling algorithms and application domains.
Discuss the types of processor scheduling such as short-term, medium-term, long-term, and I/O.
Describe the difference between processes and threads.
Compare and contrast static and dynamic approaches to real-time scheduling.
Discuss the need for preemption and deadline scheduling.
Identify ways that the logic embodied in scheduling algorithms are applicable to other domains, such as disk I/O, network scheduling, project scheduling, and other problems unrelated to computing.

## OS/MemoryManagement [core]

Minimum core coverage time: 3 hours
*Topics:*

Review of physical memory and memory management hardware
Paging and virtual memory
Working sets and thrashing
Caching

*Learning objectives:*

Explain memory hierarchy and cost-performance trade-offs.
Explain the concept of virtual memory and how it is realized in hardware and software.
Summarize the principles of virtual memory as applied to caching and paging.
Evaluate the trade-offs in terms of memory size (main memory, cache memory, auxiliary memory) and processor speed.
Defend the different ways of allocating memory to tasks, citing the relative merits of each.
Describe the reason for and use of cache memory.
Discuss the concept of thrashing, both in terms of the reasons it occurs and the techniques used to recognize and manage the problem.

## OS/DeviceManagement [elective]

*Topics:*

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

*Learning objectives:*

1. Explain the key difference between serial and parallel devices and identify the conditions in which each is appropriate.
2. Identify the relationship between the physical hardware and the virtual devices maintained by the operating system.
3. Explain buffering and describe strategies for implementing it.
4. 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.
5. Describe the advantages and disadvantages of direct memory access and discuss the circumstances in which its use is warranted.
6. Identify the requirements for failure recovery.
7. Implement a simple device driver for a range of possible devices.

## OS/SecurityAndProtection [core]

*Minimum coverage time: 2 hours*

*Topics:*

- Overview of system security
- Policy/mechanism separation
- Security methods and devices
- Protection, access control, and authentication

- Backups

*Learning objectives:*

1. Defend the need for protection and security, and the role of ethical considerations in computer use.
2. Summarize the features and limitations of an operating system used to provide protection and security.
3. Explain the mechanisms available in an OS to control access to resources.
4. Carry out simple sysadmin tasks according to a security policy, for example creating accounts, setting permissions, applying patches, and arranging for regular backups.

## OS/FileSystems [elective]

*Topics:*

- Files: data, metadata, operations, organization, buffering, sequential, nonsequential
- Directories: contents and structure
- File systems: partitioning, mount/unmount, 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.
3. Summarize how hardware developments have lead to changes in our priorities for the design and the management of file systems.

## OS/RealTimeAndEmbeddedSystems [elective]

*Topics:*

- Process and task scheduling
- Memory/disk management requirements in a real-time environment
- Failures, risks, and recovery
- Special concerns in real-time systems

*Learning objectives:*

1. Describe what makes a system a real-time system.
2. Explain the presence of and describe the characteristics of latency in real-time systems.
3. Summarize special concerns that real-time systems present and how these concerns are addressed.

## OS/FaultTolerance [elective]

*Topics:*

- Fundamental concepts: reliable and available systems
- Spatial and temporal redundancy
- Methods used to implement fault tolerance
- Examples of reliable systems

*Learning objectives:*

1. Explain the relevance of the terms fault tolerance, reliability, and availability.
2. Outline the range of methods for implementing fault tolerance in an operating system.
3. Explain how an operating system can continue functioning after a fault occurs.

## OS/SystemPerformanceEvaluation [elective]

*Topics:*

- Why system performance needs to be evaluated
- What is to be evaluated
- Policies for caching, paging, scheduling, memory management, security, and so forth

- Evaluation models: deterministic, analytic, simulation, or implementation-specific
- How to collect evaluation data (profiling and tracing mechanisms)

*Learning objectives:*
1. Describe the performance measurements used to determine how a system performs.
2. Explain the main evaluation models used to evaluate a system.

## OS/Scripting [elective]

*Topics:*
- Scripting and the role of scripting languages
- Basic system commands
- Creating scripts, parameter passing
- Executing a script
- Influences of scripting on programming

*Learning objectives:*
1. Summarize a typical set of system commands provided by an operating system.
2. Demonstrate the typical functionality of a scripting language, and interpret the implications for programming.
3. Demonstrate the mechanisms for implementing scripts and the role of scripts on system implementation and integration.
4. Implement a simple script that exhibits parameter passing.

## OS/DigitalForensics [elective]

*Topics:*
- Digital forensics and its relationship to other forensic disciplines
- Incident response responsibilities
- Forensic procedures
- Digital evidence and tracking
- Rules/Standards of Evidence
- Evidence gathering and analysis
- Forensic mechanisms
- Profiling
- Tools to support investigative work

*Learning objectives:*
1. To explain the problems addressed by digital forensics and to outline the basic principles involved in its practice
2. To outline the basic processes of information gathering and analysis in line with best practices in digital forensics
3. To recognize the role that tools can play in digital forensics and to demonstrate their use in simple examples the use of tools and to demonstrate their use
4. To explain what questions must be asked and answered to determine if the interpretation of forensic evidence is valid or questionable.

## OS/SecurityModels [elective]

*Topics:*
- Models of protection
- Memory protection
- Encryption
- Recovery management
- Types of access control: mandatory, discretionary, originator-controlled, role-based
- Access control matrix model
- Harrison-Russo-Ullman model and undecidability of security
- Confidentiality models such as Bell-LaPadula
- Integrity models such as Biba and Clark-Wilson

- Conflict of interest models such as the Chinese Wall

*Learning objectives:*
1. Compare and contrast current methods for implementing security.
2. Compare and contrast the strengths and weaknesses of two or more currently popular operating systems with respect to security.
3. Compare and contrast the security strengths and weaknesses of two or more currently popular operating systems with respect to recovery management.
4. Describe the access control matrix and how it relates to ACLs and C-Lists
5. Apply Biba's model to the checking of inputs in a program (tainted v. untainted, for example)
6. Describe how the Bell-LaPadula model combines mandatory and discretionary access control mechanisms, and explain the lattice formulation of Bell-LaPadula and Biba
7. Compare and contrast two security models
8. Relate particular security models to the models of the software life cycle
9. Apply particular models to different environments and select the model that best captures the environment

## Net Centric Computing

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

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

**NC. Net-Centric Computing (18 core hours)**
**NC/Introduction [core]**
**NC/NetworkCommunication [core]**
**NC/NetworkSecurity [core]**
**NC/WebOrganization [elective]**
**NC/NetworkedApplications [elective]**
**NC/NetworkManagement [elective]**
**NC/Compression [elective]**
**NC/MultimediaTechnologies [elective]**
**NC/MobileComputing [elective]**

### NC/Introduction [core]
*Minimum core coverage time: 2 hours*

*Topics:*
- Background and history of networking and the Internet
- Network architectures
- The range of specializations within net-centric computing
- Networks and protocols
- Networked multimedia systems
- Distributed computing
- Client/server and Peer to Peer paradigms
- Mobile and wireless computing

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

### NC/NetworkCommunication [core]
*Minimum core coverage time: 7 hours*

*Topics:*
- Network standards and standardization bodies
- The ISO 7-layer reference model in general and its instantiation in TCP/IP
- Overview of Physical and Data Link layer concepts (framing, error control, flow control, protocols)
- Data Link layer access control concepts
- Internetworking and routing (routing algorithms, internetworking, congestion control)
- Transport layer services (connection establishment, performance issues, flow and error control)

*Learning objectives:*

1. Discuss important network standards in their historical context.
2. Describe the responsibilities of the first (lowest) four layers of the ISO reference model.
3. Explain how a network can detect and correct transmission errors.
4. Explain how a packet is routed over the Internet.
5. Install a simple network with two clients and a single server using standard host configuration software tools such as DHCP.

## NC/NetworkSecurity [core]

Minimum core *coverage time: 6 hours*

*Topics:*

- Fundamentals of cryptography
    - Secret-key algorithms
    - Public-key algorithms
- Authentication protocols
- Digital signatures
    - Examples
- Network attack types: Denial of service, flooding, sniffing and traffic redirection, message integrity attacks, identity hijacking, exploit attacks (buffer overruns, Trojans, backdoors), inside attacks, infrastructure (DNS hijacking, route blackholing, misbehaving routers that drop traffic), etc.)
- Use of passwords and access control mechanisms
- Basic network defense tools and strategies
    - Intrusion Detection
    - Firewalls
    - Detection of malware
    - Kerberos
    - IPSec
    - Virtual Private Networks
    - Network Address Translation
- Network Resource Management policies
- Auditing and logging

*Learning objectives:*

1. Describe the enhancements made to IPv4 by IPSec
2. Identify protocols used to enhance Internet communication, and choose the appropriate protocol for a particular case.
3. Understand Intrusions and intrusion detection
4. Discuss the fundamental ideas of public-key cryptography.
5. Describe how public-key cryptography works.
6. Distinguish between the use of private- and public-key algorithms.
7. Summarize common authentication protocols.
8. Generate and distribute a PGP key pair and use the PGP package to send an encrypted e-mail message.
9. Summarize the capabilities and limitations of the means of cryptography that are conveniently available to the general public.
10. Describe and discuss recent successful security attacks.
11. Summarize the strengths and weaknesses associated with different approaches to security.

## NC/WebOrganization [Elective]

Minimum core *coverage time: 3 hours*

*Topics:*

- Web technologies
    - Server-side programs
    - Client-side scripts
    - The applet concept
- Characteristics of web servers
    - Handling permissions
    - File management
    - Capabilities of common server architectures

- Role of client computers
- Nature of the client-server relationship
- Web protocols
- Support tools for web site creation and web management
- Developing Internet information servers
- Publishing information and applications
- Grid Computing, cluster, mesh
- Web Services, Web 2.0, ajax

*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, and a server process to respond to the form data and produce a result).

## NC/NetworkedApplications [elective]

*Topics:*
- Protocols at the application layer
- Web interfaces: Browsers and APIs
- Web Search Technologies
- Principles of web engineering
- Database-driven web sites
- Remote procedure calls (RPC)
- Lightweight distributed objects
- The role of middleware
- Support tools
- Security issues in distributed object systems
- Enterprise-wide web-based applications
    - Service-oriented Architectures

*Learning objectives:*
1. Illustrate how interactive client-server web applications of medium size can be built using different types of Web technologies.
2. Demonstrate how to implement a database-driven web site, explaining the relevant technologies involved in each tier of the architecture and the accompanying performance tradeoffs.
3. Illustrate the current status of Web search effectiveness.
4. Implement an application that invokes the API of a web-based application.
5. Implement a distributed system using any two distributed object frameworks and compare them with regard to performance and security issues.
6. Discuss security issues and strategies in an enterprise-wide web-based application.

## NC/NetworkManagement [elective]

*Topics:*
- Overview of the issues of network management
- Use of passwords and access control mechanisms
- Domain names and name services
- Issues for Internet service providers (ISPs)
- Security issues and firewalls
- Quality of service issues: performance, failure recovery

*Learning objectives:*
1. Explain the issues for network management arising from a range of security threats, including viruses, worms, Trojan horses, and denial-of-service attacks
2. Develop a strategy for ensuring appropriate levels of security in a system designed for a particular purpose.
3. Implement a network firewall.

## NC/MultimediaTechnologies] [elective]

*Topics:*

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

*Learning objectives:*

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

## NC/MobileComputing [elective]

*Topics:*

- Overview of the history, evolution, and compatibility of wireless standards
- The special problems of wireless and mobile computing
- Wireless local area networks and satellite-based networks
- Wireless local loops
- Mobile Internet protocol
- Mobile aware adaption
- Extending the client-server model to accommodate mobility
- Mobile data access: server data dissemination and client cache management
- Software package support for mobile and wireless computing
- The role of middleware and support tools
- Performance issues
- Emerging technologies

*Learning objectives:*

1. Describe the main characteristics of mobile IP and explain how differs from IP with regard to mobility management and location management as well as performance.
2. Illustrate (with home agents and foreign agents) how e-mail and other traffic is routed using mobile IP.
3. Implement a simple application that relies on mobile and wireless data communications.
4. Describe areas of current and emerging interest in wireless and mobile computing, and assess the current capabilities, limitations, and near-term potential of each.

# Programming Languages

A programming language is a programmer's principal interface with the computer. More than just knowing how to program in a single language, programmers need to understand the different styles of programming promoted by different languages. In their professional life, they will be working with many different languages and styles at once, and will encounter many different languages over the course of their careers. Understanding the variety of programming languages and the design tradeoffs between the different programming paradigms makes it much easier to master new languages quickly. Understanding the pragmatic aspects of programming languages also requires a basic knowledge of programming language translation and runtime features such as storage allocation.

**PL. Programming Languages (21 core hours)**

- **PL/Overview [core]**
- **PL/VirtualMachines [core]**
- **PL/BasicLanguageTranslation [core]**
- **PL/DeclarationsAndTypes [core]**
- **PL/AbstractionMechanisms [core]**
- **PL/ObjectOrientedProgramming [core])**
- **PL/FunctionalProgramming [elective]**
- **PL/LanguageTranslationSystems [elective]**
- **PL/TypeSystems [elective]**
- **PL/ProgrammingLanguageSemantics [elective]**
- **PL/ProgrammingLanguageDesign [elective]**

## PL/Overview [core]

*Minimum core coverage time: 2 hours*

*Topics:*

- History of programming languages
- Brief survey of programming paradigms
- Procedural languages
- Object-oriented languages
- Functional languages
- Declarative, non-algorithmic languages
- Scripting languages
- The effects of scale on programming methodology

*Learning objectives:*

1. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today.
2. Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit.
3. Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
4. Distinguish between programming-in-the-small and programming-in-the-large.

## PL/VirtualMachines [core]

*Minimum core coverage time: 1 hour*

*Topics:*

- The concept of a virtual machine
- Hierarchy of virtual machines
- Intermediate languages
- Security issues arising from running code on an alien machine

*Learning objectives:*

1. Describe the importance and power of abstraction in the context of virtual machines.
2. Explain the benefits of intermediate languages in the compilation process.
3. Evaluate the tradeoffs in performance vs. portability.

4. Explain how executable programs can breach computer system security by accessing disk files and memory.

## PL/BasicLanguageTranslation [core]
*Minimum core coverage time: 2 hours*

*Topics:*
- Comparison of interpreters and compilers
- Language translation phases (lexical analysis, parsing, code generation, optimization)
- Machine-dependent and machine-independent aspects of translation

*Learning objectives:*
1. Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
2. Describe the phases of program translation from source code to executable code and the files produced by these phases.
3. Explain the differences between machine-dependent and machine-independent translation and where these differences are evident in the translation process.

## PL/DeclarationsAndTypes [core]
*Minimum core coverage time: 3 hours*

*Topics:*
- The conception of types as a set of values with together with a set of operations
- Declaration models (binding, visibility, scope, and lifetime)
- Overview of type-checking
- Garbage collection

*Learning objectives:*
1. Explain the value of declaration models, especially with respect to programming-in-the-large.
2. Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size.
3. Discuss type incompatibility.
4. Demonstrate different forms of binding, visibility, scoping, and lifetime management.
5. Defend the importance of types and type-checking in providing abstraction and safety.
6. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection).

## PL/AbstractionMechanisms [core]
Minimum core *coverage time: 3 hours*

*Topics:*
- Procedures, functions, and iterators as abstraction mechanisms
- Parameterization mechanisms (reference vs. value)
- Activation records and storage management
- Type parameters and parameterized types
- Modules in programming languages

*Learning objectives:*
1. Explain how abstraction mechanisms support the creation of reusable software components.
2. Demonstrate the difference between call-by-value and call-by-reference parameter passing.
3. Defend the importance of abstractions, especially with respect to programming-in-the-large.
4. Describe how the computer system uses activation records to manage program modules and their data.

## PL/ObjectOrientedProgramming [core]
Minimum core *coverage time: 10 hours*

*Topics:*
- Object-oriented design
- Encapsulation and information-hiding
- Separation of behavior and implementation
- Classes and subclasses
- Inheritance (overriding, dynamic dispatch)

- Polymorphism (subtype polymorphism vs. inheritance)
- Class hierarchies
- Collection classes and iteration protocols
- Internal representations of objects and method tables

*Learning objectives:*
1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism.
2. Design, implement, test, and debug simple programs in an object-oriented programming language.
3. Describe how the class mechanism supports encapsulation and information hiding.
4. Design, implement, and test the implementation of "is-a" relationships among objects using a class hierarchy and inheritance.
5. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
6. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
7. Describe how iterators access the elements of a container.

## PL/ FunctionalProgramming [elective]

*Topics:*
- Overview and motivation of functional languages
- Recursion over lists, natural numbers, trees, and other recursively-defined data
- Pragmatics (debugging by divide and conquer; persistency of data structures)
- Amortized efficiency for functional data structures
- Closures and uses of functions as data (infinite sets, streams)

*Learning objectives:*
1. Outline the strengths and weaknesses of the functional programming paradigm.
2. Design, code, test, and debug programs using the functional paradigm.
3. Explain the use of functions as data, including the concept of closures.

## PL/LanguageTranslationSystems [elective]

*Topics:*
- Application of regular expressions in lexical scanners
- Parsing (concrete and abstract syntax, abstract syntax trees)
- Application of context-free grammars in table-driven and recursive-descent parsing
- Symbol table management
- Code generation by tree walking
- Architecture-specific operations: instruction selection and register allocation
- Optimization techniques
- The use of tools in support of the translation process and the advantages thereof
- Program libraries and separate compilation
- Building syntax-directed tools

*Learning objectives:*
1. Describe the steps and algorithms used by language translators.
2. Recognize the underlying formal models such as finite state automata, push-down automata and their connection to language definition through regular expressions and grammars.
3. Discuss the effectiveness of optimization.
4. Explain the impact of a separate compilation facility and the existence of program libraries on the compilation process.

## PL/TypeSystems [elective]

*Topics:*
- Data type as set of values with set of operations
- Data types
- Elementary types

- Product and coproduct types
- Algebraic types
- Recursive types
- Arrow (function) types
- Parameterized types
- Type-checking models
- Semantic models of user-defined types
- Type abbreviations
- Abstract data types
- Type equality
- Parametric polymorphism
- Subtype polymorphism
- Type-checking algorithms

*Learning objectives:*
1. Formalize the notion of typing.
2. Describe each of the elementary data types.
3. Explain the concept of an abstract data type.
4. Recognize the importance of typing for abstraction and safety.
5. Differentiate between static and dynamic typing.
6. Differentiate between type declarations and type inference.
7. Evaluate languages with regard to typing.

## PL/ProgrammingLanguageSemantics [elective]

*Topics:*
- Informal semantics
- Overview of formal semantics
- Denotational semantics
- Axiomatic semantics
- Operational semantics

*Learning objectives:*
1. Explain the importance of formal semantics.
2. Differentiate between formal and informal semantics.
3. Describe the different approaches to formal semantics.
4. Evaluate the different approaches to formal semantics.

## PL/ProgrammingLanguageDesign [elective]

*Topics:*
- General principles of language design
- Design goals
- Typing regimes
- Data structure models
- Control structure models
- Abstraction mechanisms

*Learning objectives:*
1. Evaluate the impact of different typing regimes on language design, language usage, and the translation process.
2. Explain the role of different abstraction mechanisms in the creation of user-defined facilities

# Human-Computer Interaction

Human-computer interaction is an important area of computing knowledge. As more people conduct more of their daily activities interacting with a computer, the construction of interfaces that ease that interaction is critical for increasing satisfaction and improving productivity. As more software requires a user interface, knowing how to create a usable interface and testing the usability of that interface become required skills for all computer science students.

The design of human-computer interfaces impacts the software life-cycle. Where interfaces used to be designed after the functionality was completed, we now know that the design of a usable interface should occur early in the cycle. We know that the design and implementation of the core functionality can influence the user interface. Human-computer interfaces are themselves software components, and the development and reuse of those components become an important part of the development of most software today.

**HC. Human-Computer Interaction (8 core hours)**

**HC/Foundations [core]**
**HC/BuildingGUIInterfaces [core]**
**HC/UserCcenteredSoftwareEvaluation [elective]**
**HC/UserCenteredSoftwareDevelopment [elective]**
**HC/GUIDesign [elective]**
**HC/GUIProgramming [elective]**
**HC/MultimediaAndMultimodalSystems [elective]**
**HC/CollaborationAndCommunication [elective]**
**HC/InteractionDesignForNewEnvironments [elective]**
**HC/HumanFactorsAndSecurity [elective]**

## HC/Foundations [core]

Minimum core *coverage time: 6 hours*

*Topics:*

- Motivation: Why the study of how people interact with technology is vital for the development of most usable and acceptable systems
- Contexts for HCI (mobile devices, consumer devices, business applications, web, business applications, collaboration systems, games, etc.)
- Process for user-centered development: early focus on users, empirical testing, iterative design.
- Different measures for evaluation: utility, efficiency, learnability, user satisfaction.
- Models that inform human-computer interaction (HCI) design: attention, perception and recognition, movement, and cognition.
- Social issues influencing HCI design and use: culture, communication, and organizations.
- Accommodating human diversity, including universal design and accessibility and designing for multiple cultural and linguistic contexts.
- The most common interface design mistakes.
- User interface standards

*Learning objectives:*

1. Discuss why user-centered product development is important.
2. Explain why both individual human models and social models are important to consider in design of human-computer interaction.
3. Define a user-centered design process that explicitly recognizes that the user is not like the developer or her acquaintances.
4. Describe ways in which a user-centered design process may fail, including examples.
5. Define different processes for defining interfaces for different contexts.
6. Differentiate between the role of hypotheses and experimental results vs. correlations.
7. Choose between qualitative and quantitative evaluation methods for a given evaluation question.
8. Use vocabulary for analyzing human interaction with software: perceived affordance, conceptual model, mental model, metaphor, interaction design, feedback, and so forth.

9. Provide examples of how different interpretations that a given icon, symbol, word, or color can have in (a) two different human cultures and (b) in a culture and one of its subcultures.
10. Be able to describe at least one national or international user interface design standard

## HC/BuildingGUIInterfaces [core]

Minimum core *coverage time: 2 hours*

*Topics:*

- Principles of graphical user interfaces (GUIs).
- Action-object versus object-action.
- User interface events.
- Constructing a user-interface for a native system vs. the web.

*Learning objectives:*

1. Explain principles for design of user interfaces, such as learnability, flexibility, and robustness.
2. Describe examples of bad navigation, bad screen layout, and incomprehensible interface design.
3. Create a simple application that supports a graphical user interface, for either the Web or a windowing system.
4. Observe a user attempting to use the application and have the user critique the application.
5. Explain how careful user evaluation goes beyond the simple observation of a single user.

## HC/UserCenteredSoftwareEvaluation [elective]

*Topics:*

- Evaluation without typical users: walkthroughs, KLM, expert-based analysis, heuristics, guidelines, and standards
- Evaluation with typical users: observation, think-aloud, interview, survey, experiment.
- Challenges to effective evaluation: sampling, generalization.
- Reporting the results of evaluations

*Learning objectives:*

1. Discuss evaluation criteria: task time/completion, time to learn, retention, errors, and user satisfaction.
2. Conduct a walkthrough, expert-based analysis, and a Keystroke Level Model (KLM) analysis.
3. Compare a given user interface to a set of guidelines or standards to identify inadequacies.
4. Conduct a usability test with more than one user, gathering results using at least two different methods.
5. Compare a laboratory test to a field test.
6. Explain a usability problem that is supported by results from a usability test. Recommend a solution to the usability problem.
7. Critique a user evaluation, to point out threats to validity.
8. Given an evaluation context (e.g. amount of time, availability of test users, place in the design process, evaluation goals), recommend and justify an evaluation method.

## HC/GUIDesign [elective]

*Topics:*

- Choosing interaction styles (command line, menu, voice, gestural, WIMP) and interaction techniques
- Choosing the right widget for users and tasks
- HCI aspects of screen design: layout, color, fonts, labeling
- Handling human/system failure.
- Beyond simple screen design: visualization, representation, metaphor
- Multi-modal interaction: graphics, sound, and haptics.
- 3D interaction and virtual reality
- Designing for small devices, e.g., cell phones.
- Multi-cultural interaction and communication

*Learning objectives:*

1. Summarize common interaction styles.
2. Explain good design principles of each of the following: common widgets; sequenced screen presentations; simple error-trap dialog; a user manual.

3. Design, prototype, and evaluate a simple 2D GUI illustrating knowledge of the concepts taught in HC3 and HC4.
4. Identify the challenges that exist in moving from 2D to 3D interaction.
5. Identify the challenges that exist in moving from desktop or laptop screen to a mobile device.

## HC/GUIProgramming [elective]

*Topics:*

- UIMS, dialogue independence and levels of analysis, Seeheim model
- Widget classes and libraries
- Event management and user interaction
- Web design vs. native application design
- Geometry management
- GUI builders and UI programming environments
- Cross-platform design
- Design for small, mobile devices

*Learning objectives:*

1. Differentiate between the responsibilities of the UIMS and the application.
2. Differentiate between kernel-based and client-server models for the UI.
3. Compare the event-driven paradigm with more traditional procedural control for the UI.
4. Describe aggregation of widgets and constraint-based geometry management.
5. Explain callbacks and their role in GUI builders.
6. Identify at least three differences common in cross-platform (e.g., desktop, Web, and cell phone) UI design.
7. Identify as many commonalities as you can that are found in UIs across different platforms.

## HC/MultimediaAndMultimodalSystems [elective]

*Topics:*

- Categorization and information architectures: hierarchies, grids, hypermedia , networks
- Information retrieval and human performance
- Web search
- Usability of database query languages
- Graphics
- Sound
- HCI design of multimedia information systems
- Speech recognition and natural language processing
- Information appliances and mobile computing
- Interactive visualizations
- Information design and navigation
- Touch interfaces

*Learning objectives:*

1. Discuss how information retrieval differs from transaction processing.
2. Explain how the organization of information supports retrieval.
3. Describe the major usability problems with database query languages.
4. Explain the current state of speech recognition technology in particular and natural language processing in general.
5. Design, prototype, and evaluate a simple Multimedia Information System illustrating knowledge of the concepts taught in HC4, HC5, and HC7.

## HC/CollaborationAndCommunication [elective]

*Topics:*

- Groupware to support specialized tasks: document preparation, multi-player games
- Asynchronous group communication: e-mail, bulletin boards, listservs, wikis, ...
- Synchronous group communication: chat rooms, conferencing
- Online communities: MUDs/MOOs,

- Software characters and intelligent agents, virtual worlds and avatars
- Social psychology
- Social networking
- Social Computing
- Collaborative usability techniques.

*Learning objectives:*
1. Compare the HCI issues in individual interaction with group interaction.
2. Discuss several issues of social concern raised by collaborative software.
3. Discuss the HCI issues in software that embodies human intention.
4. Describe the difference between synchronous and asynchronous communication.
5. Design, prototype, and evaluate a simple groupware or group communication application illustrating knowledge of the concepts taught in HC4, HC5, and HC8.
6. Participate in a team project for which some interaction is face-to-face and other interaction occurs via a mediating software environment.
7. Describe the similarities and differences between face-to-face and software-mediated collaboration.

## HC/InteractionDesignForNewEnvironments [elective]

*Topics:*
- Interaction design for engaging interactive experiences
- Presence, tele-presence and immersive environments
- Affective interaction and emotion
- Ambient intelligence
- Physical computing and embodied interaction

*Learning objectives:*
1. Compare the methodological and philosophical issues involved with designing for usability and designing for engagement.
2. Discuss several issues of social and ethical concern raised by immersive environments and high levels of emotion in HCI.
3. Discuss the HCI issues involved in interactive software that embodies a level of intelligence.
4. Describe the difference between interaction design and traditional HCI.
5. Design, prototype, and evaluate an engaging interactive system for entertainment or education.
6. Evaluate the experiences or people in immersive environments.
7. Describe the issues involved with tangible user interfaces, gesture and full body interaction.
8. Describe the issues involved with engaging all the senses in interactive experiences.

## HC/HumanFactorsAndSecurity [elective]

*Topics:*
- Applied psychology and security policies
- Usability design and security
- Social engineering
- Identity theft
- Phishing

*Learning objectives:*
1. To explain the concept of phishing, and how to recognize it
2. To explain the concept of identity theft is and how to hinder it
3. To design a user interface for a security mechanism
4. To discuss procedures that counter a social engineering attack
5. To analyze a security policy and/or procedures to show where they meet, or fail to meet, usability considerations

# Graphics and Visual Computing (GV)

The area encompassed by Graphics and Visual Computing (GV) is divided into four interrelated fields:

- *Computer graphics.* Computer graphics is the art and science of communicating information using images that are generated and presented through computation. This requires (a) the design and construction of models that represent information in ways that support the creation and viewing of images, (b) the design of devices and techniques through which the person may interact with the model or the view, (c) the creation of techniques for rendering the model, and (d) the design of ways the images may be preserved The goal of computer graphics is to engage the person's visual centers alongside other cognitive centers in understanding.
- *Visualization.* The field of visualization seeks to determine and present underlying correlated structures and relationships in both scientific (computational and medical sciences) and more abstract datasets. The prime objective of the presentation should be to communicate the information in a dataset so as to enhance understanding. Although current techniques of visualization exploit visual abilities of humans, other sensory modalities, including sound and haptics (touch), are also being considered to aid the discovery process of information.
- *Virtual reality.* Virtual reality (VR) enables users to experience a three-dimensional environment generated using computer graphics, and perhaps other sensory modalities, to provide an environment for enhanced interaction between a human user and a computer-created world.
- *Computer vision.* The goal of computer vision (CV) is to deduce the properties and structure of the three-dimensional world from one or more two-dimensional images. The understanding and practice of computer vision depends upon core concepts in computing, but also relates strongly to the disciplines of physics, mathematics, and psychology.

**GV. Graphics and Visual Computing (3 core hours)**

**GV/FundamentalTechniques [core]**
**GV/GraphicSystems [core]**
**GV/GraphicCommunication [elective]**
**GV/GeometricModeling [elective]**
**GV/BasicRendering [elective]**
**GV/AdvancedRendering [elective]**
**GV/AdvancedTechniques [elective]**
**GV/ComputerAnimation [elective]**
**GV/Visualization [elective]**
**GV/VirtualReality [elective]**
**GV/ComputerVision [elective]**
**GV/ComputationalGeometry [elective]**
**GV/GameEngineProgramming [elective]**

## GV/FundamentalTechniques [core]

Minimum core *coverage time: 2 hours*

*Topics:*

- Hierarchy of graphics software
- Using a graphics API
- Simple color models (RGB, HSB, CMYK)
- Homogeneous coordinates
- Affine transformations (scaling, rotation, translation)
- Viewing transformation
- Clipping

*Learning objectives:*

1. Distinguish the capabilities of different levels of graphics software and describe the appropriateness of each.
2. Create images using a standard graphics API.
3. Use the facilities provided by a standard API to express basic transformations such as scaling, rotation, and translation.

4. Implement simple procedures that perform transformation and clipping operations on a simple 2-dimensional image.
5. Discuss the 3-dimensional coordinate system and the changes required to extend 2D transformation operations to handle transformations in 3D

## GV/GraphicSystems [core]

Minimum core *coverage time: 1 hour*

*Topics:*

- Raster and vector graphics systems
- Video display devices
- Physical and logical input devices
- Issues facing the developer of graphical systems

*Learning objectives:*

1. Describe the appropriateness of graphics architectures for given applications.
2. Explain the function of various input devices.
3. Compare and contrast the techniques of raster graphics and vector graphics.
4. Use current hardware and software for creating and displaying graphics.
5. Discuss the expanded capabilities of emerging hardware and software for creating and displaying graphics.

## GV/GeometricModeling [elective]

*Topics:*

- Polygonal representation of 3D objects
- Parametric polynomial curves and surfaces
- Constructive Solid Geometry (CSG) representation
- Implicit representation of curves and surfaces
- Spatial subdivision techniques
- Procedural models
- Deformable models
- Subdivision surfaces
- Multiresolution modeling
- Reconstruction

*Learning objectives:*

1. Create simple polyhedral models by surface tessellation.
2. Construct CSG models from simple primitives, such as cubes and quadric surfaces.
3. Generate a mesh representation from an implicit surface.
4. Generate a fractal model or terrain using a procedural method.
5. Generate a mesh from data points acquired with a laser scanner.

## GV/BasicRendering [elective]

*Topics:*

- Line generation algorithms (Bresenham)
- Font generation: outline vs. bitmap
- Light-source and material properties
- Ambient, diffuse, and specular reflections
- Phong reflection model
- Rendering of a polygonal surface; flat, Gouraud, and Phong shading
- Texture mapping, bump texture, environment map
- Introduction to ray tracing
- Image synthesis, sampling techniques, and anti-aliasing

*Learning objectives:*

1. Explain the operation of the Bresenham algorithm for rendering a line on a pixel-based display.
2. Explain the concept and applications of each of these techniques.
3. Demonstrate each of these techniques by creating an image using a standard API.
4. Describe how a graphic image has been created.

## GV/AdvancedRendering [elective]

*Topics:*
- Transport equations
- Ray tracing algorithms
- Photon tracing
- Radiosity for global illumination computation, form factors
- Efficient approaches to global illumination
- Monte Carlo methods for global illumination
- Image-based rendering, panorama viewing, plenoptic function modeling
- Rendering of complex natural phenomenon
- Non-photorealistic rendering

*Learning objectives:*
1. Describe several transport equations in detail, noting all comprehensive effects.
2. Describe efficient algorithms to compute radiosity and explain the tradeoffs of accuracy and algorithmic performance.
3. Describe the impact of meshing schemes.
4. Explain image-based rendering techniques, light fields, and associated topics.

## GV/AdvancedTechniques [elective]

*Topics:*
- Color quantization
- Scan conversion of 2D primitive, forward differencing
- Tessellation of curved surfaces
- Hidden surface removal methods
- Z-buffer and frame buffer, color channels (a channel for opacity)
- Advanced geometric modeling techniques

*Learning objectives:*
1. Describe the techniques identified in this section.
2. Explain how to recognize the graphics techniques used to create a particular image.
3. Implement any of the specified graphics techniques using a primitive graphics system at the individual pixel level.
4. Use common animation software to construct simple organic forms using metaball and skeleton.

## GV/ComputerAnimation [elective]

*Topics:*
- Key-frame animation
- Camera animation
- Scripting system
- Animation of articulated structures: inverse kinematics
- Motion capture
- Procedural animation
- Deformation

*Learning objectives:*
1. Explain the spline interpolation method for producing in-between positions and orientations.
2. Compare and contrast several technologies for motion capture.
3. Use the particle function in common animation software to generate a simple animation, such as fireworks.
4. Use free-form deformation techniques to create various deformations.

## GV/Visualization [elective]

*Topics:*
- Basic viewing and interrogation functions for visualization

- Visualization of vector fields, tensors, and flow data
- Visualization of scalar field or height field: isosurface by the marching cube method
- Direct volume data rendering: ray-casting, transfer functions, segmentation, hardware
- Information visualization: projection and parallel-coordinates methods

*Learning objectives:*
1. Describe the basic algorithms behind scalar and vector visualization.
2. Describe the tradeoffs of the algorithms in terms of accuracy and performance.
3. Employ suitable theory from signal processing and numerical analysis to explain the effects of visualization operations.
4. Describe the impact of presentation and user interaction on exploration.

## GV/VirtualReality [elective]

*Topics:*
- Stereoscopic display
- Force feedback simulation, haptic devices
- Viewer tracking
- Collision detection
- Visibility computation
- Time-critical rendering, multiple levels of details (LOD)
- Image-base VR system
- Distributed VR, collaboration over computer network
- Interactive modeling
- User interface issues
- Applications in medicine, simulation, and training

*Learning objectives:*
1. Describe the optical model realized by a computer graphics system to synthesize stereoscopic view.
2. Describe the principles of different viewer tracking technologies.
3. Explain the principles of efficient collision detection algorithms for convex polyhedra.
4. Describe the differences between geometry- and image-based virtual reality.
5. Describe the issues of user action synchronization and data consistency in a networked environment.
6. Determine the basic requirements on interface, hardware, and software configurations of a VR system for a specified application.

## GV/ComputerVision [elective]

*Topics:*
- Image acquisition
- The digital image and its properties
- Image preprocessing
- Segmentation (thresholding, edge- and region-based segmentation)
- Shape representation and object recognition
- Motion analysis
- Case studies (object recognition, object tracking)

*Learning objectives:*
1. Explain the image formation process.
2. Explain the advantages of two and more cameras, stereo vision.
3. Explain various segmentation approaches, along with their characteristics, differences, strengths, and weaknesses.
4. Describe object recognition based on contour- and region-based shape representations.
5. Explain differential motion analysis methods.
6. Describe the differences in object tracking methods.

## GV/ComputationalGeometry [elective]

*Topics:*

- Purpose and nature of computational geometry
- Application areas – convex hull, line intersection issues, Delauney triangulation, polgygon triangulation, Voroni diagrams
- Combinatorial computational geometry: static problems such as developing efficient algorithms for certain geometric situations; dynamic problems including
- Numerical computational geometry: gmodeling, computer-aided geometric design; curve and surface modeling including representation of these: Bezier curves, spline curves and surfaces; level set method.

*Learning objectives:*
- Be aware of algorithms for certaoin geometric tasks
- Be able to select algorithms appropriate to particular situations

## GV/GameEngineProgramming [elective]

*Topics:*
- The nature of games engines (as an integrated development environment) and their purpose
- Hardware support including use of threading; performance issues; input devices
- Typical components including 3D rendering, and support for real-time graphics and interaction; also physics simulation, collision detaection, sound, artificial intelligence; terrain rendering

*Learning objectives:*
1. To be aware of the range of possibilities for games engines, including their potential and their limitations
2. To use a games engine to construct a simple game

# Intelligent Systems (IS)

The field of artificial intelligence (AI) is concerned with the design and analysis of autonomous agents. These are software systems and/or physical machines, with sensors and actuators, embodied for example within a robot or an autonomous spacecraft. An intelligent system has to perceive its environment, to act rationally towards its assigned tasks, to interact with other agents and with human beings.

These capabilities are covered by topics such as computer vision, planning and acting, robotics, multiagents systems, speech recognition, and natural language understanding. They rely on a broad set of general and specialized knowledge representations and reasoning mechanisms, on problem solving and search algorithms, and on machine learning techniques.

Furthermore, artificial intelligence provides a set of tools for solving problems that are difficult or impractical to solve with other methods. These include heuristic search and planning algorithms, formalisms for knowledge representation and reasoning, machine learning techniques, and methods applicable to sensing and action problems such as speech and language understanding, computer vision, and robotics, among others. The student needs to be able to determine when an AI approach is appropriate for a given problem, and to be able to select and implement a suitable AI method.

**IS. Intelligent Systems (10 core hours)**

- **IS/FundamentalIssues [core]**
- **IS/BasicSearchStrategies [core]**
- **IS/KnowledgeBasedReasoning [core]**
- **IS/AdvancedSearch [elective]**
- **IS/AdvancedReasoning [elective]**
- **IS/Agents [elective]**
- **IS/NaturaLanguageProcessing [elective]**
- **IS/MachineLearning [elective]**
- **IS/PlanningSystems [elective]**
- **IS/Robotics [elective]**
- **IS/Perception [elective]**

## IS/FundamentalIssues [core]

Minimum core *coverage time: 1 hour*

*Topics:*

- History of artificial intelligence
- Philosophical questions
- The Turing test
- Searle's "Chinese Room" thought experiment
- Ethical issues in AI
- Fundamental definitions
- Optimal vs. human-like reasoning
- Optimal vs. human-like behavior
- Philosophical questions
- Modeling the world
- The role of heuristics

*Learning objectives:*

1. Describe the Turing test and the "Chinese Room" thought experiment.
2. Differentiate the concepts of optimal reasoning and human-like reasoning.
3. Differentiate the concepts of optimal behavior and human-like behavior.
4. List examples of intelligent systems that depend on models of the world.
5. Describe the role of heuristics and the need for tradeoffs between optimality and efficiency.

## IS/BasicSearchStrategies [core]

Minimum core *coverage time: 5 hours*

*Topics:*
- Problem spaces; problem solving by search
- Brute-force search (breadth-first, depth-first, depth-first with iterative deepening)
- Best-first search (generic best-first, Dijkstra's algorithm, A*, admissibility of A*)
- Two-player games (minimax search, alpha-beta pruning
- Constraint satisfaction (backtracking and local search methods)

*Learning objectives:*
1. Formulate an efficient problem space for a problem expressed in English by expressing that problem space in terms of states, operators, an initial state, and a description of a goal state.
2. Describe the problem of combinatorial explosion and its consequences.
3. Select an appropriate brute-force search algorithm for a problem, implement it, and characterize its time and space complexities.
4. Select an appropriate heuristic search algorithm for a problem and implement it by designing the necessary heuristic evaluation function.
5. Describe under what conditions heuristic algorithms guarantee optimal solution.
6. Implement minimax search with alpha-beta pruning for some two-player game.
7. Formulate a problem specified in English as a constraint-satisfaction problem and implement it using a chronological backtracking algorithm.

## IS/KnowledgeBasedReasoning [core]
Minimum core *coverage time: 4 hours*

*Topics:*
- Review of propositional and predicate logic
- Resolution and theorem proving
- Nonmonotonic inference; unification and lifting, forward chaining, backward chaining, resolution
- Probabilistic reasoning
- Bayes theorem

*Learning objectives:*
1. Explain the operation of the resolution technique for theorem proving.
2. Explain the distinction between monotonic and non-monotonic inference.
3. Discuss the advantages and shortcomings of probabilistic reasoning.
4. Apply Bayes theorem to determine conditional probabilities.

## IS/AdvancedSearch [elective]

*Topics:*
- Heuristics
- Local search and optimization
- Hill climbing
- Genetic algorithms
- Simulated annealing
- Local beam search
- Adversarial search for games

*Learning objectives:*
1. Explain what genetic algorithms are and constrastcontrast their effectiveness with the classic problem-solving and search techniques.
2. Explain how simulated annealing can be used to reduce search complexity and contrast its operation with classic search techniques.
3. Apply local search techniques to a classic domain.

## IS/AdvancedReasoning [elective]

*Topics:*

- Structured representation
    - Frames and objects
    - Description logics
    - Inheritance systems
- Non-monotonic reasoning
    - Nonclassical logics
    - Default reasoning
    - Belief revision
    - Preference logics
    - Integration of knowledge sources
    - Aggregation of conflicting belief
- Reasoning on action and change
    - Situation calculus
    - Event calculus
    - Ramification problems
- Temporal and spatial reasoning
- Uncertainty
    - Probabilistic reasoning
    - Bayesian nets
    - Decision theory
- Knowledge representation for diagnosis, qualitative representation
- Ontology engineering
- Semantic networks

*Learning objectives:*
1. Compare and contrast the most common models used for structured knowledge representation, highlighting their strengths and weaknesses.
2. Characterize the components of nonmonotonic reasoning and its usefulness as a representational mechanisms for belief systems.
3. Apply situation and event calculus to problems of action and change.
4. Articulate the distinction between temporal and spatial reasoning, explaining how they interrelate.
5. Describe and contrast the basic techniques for representing uncertainty.
6. Describe and contrast the basic techniques for diagnosis and qualitative representation.

## IS/Agents [elective]

*Topics:*
- Definition of agents
- Successful applications and state-of-the-art agent-based systems
- Agent architectures
    - Simple reactive agents
    - Reactive planners
    - Layered architectures
    - Example architectures and applications
- Agent theory
    - Commitments
    - Intentions
    - Decision-theoretic agents
    - Markov decision processes (MDP)
- Software agents, personal assistants, and information access
    - Collaborative agents
    - Information-gathering agents
- Believable agents (synthetic characters, modeling emotions in agents)
    - Learning agents
    - Multi-agent systems
    - Economically inspired multi-agent systems
    - Collaborating agents
    - Agent teams

    - Agent modeling
    - Multi-agent learning
- Introduction to robotic agents
- Mobile agents

*Learning objectives:*
1. Explain how an agent differs from other categories of intelligent systems.
2. Characterize and contrast the standard agent architectures.
3. Describe the applications of agent theory, to domains such as software agents, personal assistants, and believable agents.
4. Describe the distinction between agents that learn and those that don't.
5. Demonstrate using appropriate examples how multi-agent systems support agent interaction.
6. Describe and contrast robotic and mobile agents.

## IS/NaturalLanguageProcessing [elective]
*Topics:*
- Deterministic and stochastic grammar
- Parsing algorithms
- Corpus-based methods
- Information retrieval and information extraction
- Language translation
- Speech recognition

*Learning objectives:*
1. Define and contrast deterministic and stochastic grammars, providing examples to show the adequacy of each.
2. Identify the classic parsing algorithms for parsing natural language.
3. Defend the need for an established corpus.
4. Give examples of catalog and look up procedures in a corpus-based approach.
5. Articulate the distinction between techniques for information retrieval, language translation, and speech recognition.

## IS/MachineLearning [elective]

*Topics:*
- Definition and examples of machine learning
- Inductive learning, statistical based learning, reinforcement learning
- Supervised learning
- Learning decision trees
- Learning neural networks
- Learning belief networks
- The nearest neighbor algorithm
- Learning theory
- The problem of overfitting
- Unsupervised learning
- Reinforcement learning

*Learning objectives:*
1. Explain the differences among the three main styles of learning: supervised, reinforcement, and unsupervised.
2. Implement simple algorithms for supervised learning, reinforcement learning, and unsupervised learning.
3. Determine which of the three learning styles is appropriate to a particular problem domain.
4. Compare and contrast each of the following techniques, providing examples of when each strategy is superior: decision trees, neural networks, and belief networks..
5. Implement a simple learning system using decision trees, neural networks and/or belief networks, as appropriate.
6. Characterize the state of the art in learning theory, including its achievements and its shortcomings.
7. Explain the nearest neighbor algorithm and its place within learning theory.
8. Explain the problem of overfitting, along with techniques for detecting and managing the problem.

## IS/PlanningSystems [elective]

*Topics:*

- Definition and examples of planning systems<
- Planning as search
- Operator-based planning
- Planning graphs
- Propositional planning
- Extending planning systems (case-based, learning, and probabilistic systems)
- Static world planning systems
- Planning and execution including conditional planning and continuous planning
- Mobile agent planning
- Planning and robotics

*Learning objectives:*

1. Define the concept of a planning system.
2. Explain how planning systems differ from classical search techniques.
3. Articulate the differences between planning as search, operator-based planning, and propositional planning, providing examples of domains where each is most applicable.
4. Define and provide examples for each of the following techniques: case-based, learning, and probabilistic planning.
5. Compare and contrast static world planning systems with those need dynamic execution.
6. Explain the impact of dynamic planning on robotics.

## IS/Robotics [elective]

*Topics:*

- Overview
- State-of-the-art robot systems
- Planning vs. reactive control
- Uncertainty in control
- Sensing
- World models
- Configuration space<
- Planning
- Sensing
- Robot programming
- Navigation and control
- Robotic software and its architecture

*Learning objectives:*

1. Outline the potential and limitations of today's state-of-the-art robot systems.
2. Implement configuration space algorithms for a 2D robot and complex polygons.
3. Implement simple motion planning algorithms.
4. Explain the uncertainties associated with sensors and how to deal with those uncertainties.
5. Design a simple control architecture.
6. Describe various strategies for navigation in unknown environments, including the strengths and shortcomings of each.
7. Describe various strategies for navigation with the aid of landmarks, including the strengths and shortcomings of each.

## IS/Perception [elective]

*Topics:*

- Perception: role and applications

- Image formation: light, colour, shades
- Image and object detection: feature recognition, object recognition
- Technologies
- Software characteristics

*Learning objectives:*

1. Describe the importance of image and object recognition in Ai and indicate significant applications of this technology.
2. Outline the main approaches to object recognition
3. Describe the distinguishing characteristics of the technologies used for perception.

# Information Management

**IM. Information Management (11 core hours)**

- **IM/InformationModels [core]**
- **IM/DatabaseSystems [core]**
- **IM/DataModeling [core]**
- **IM/Indexing [elective]**
- **IM/RelationalDatabases [elective]**
- **IM/QueryLanguages [elective]**
- **IM/RelationalDatabaseDesign [elective]**
- **IM/TransactionProcessing [elective]**
- **IM/DistributedDatabases [elective]**
- **IM/PhysicalDatabaseDesign [elective]**
- **IM/DataMining [elective]**
- **IM/InformationStorageAndRetrieval [elective]**
- **IM/Hypermedia [elective]**
- **IM/MultimediaSystems [elective]**
- **IM/DigitalLibraries [elective]**

## IM/InformationModels [core]

*Minimum core coverage time: 4 hours*

*Topics:*

- Information storage and retrieval (IS&R)
- Information management applications
- Information capture and representation
- Metadata/schema association with data
- Analysis and indexing
- Search, retrieval, linking, navigation
- Declarative and navigational queries
- Information privacy, integrity, security, and preservation
- Scalability, efficiency, and effectiveness
- Concepts of Information Assurance (data persistence, integrity)

*Learning objectives:*

1. Compare and contrast information with data and knowledge.
2. Critique/defend a small- to medium-size information application with regard to its satisfying real user information needs.
3. Show uses of explicitly stored metadata/schema associated with data
4. Explain uses of declarative queries
5. Give a declarative version for a navigational query
6. Describe several technical solutions to the problems related to information privacy, integrity, security, and preservation.
7. Explain measures of efficiency (throughput, response time) and effectiveness (recall, precision).
8. Describe approaches to ensure that information systems can scale from the individual to the global.
9. Identify issues of data persistence to an organization.
10. Describe vulnerabilities to data integrity in specific scenarios.

## IM/DatabaseSystems [core]

Minimum core *coverage time: 3 hours*

*Topics:*

- History and motivation for database systems
- Components of database systems
- DBMS functions

- Database architecture and data independence
- Use of a declarative query language

*Learning objectives:*
1. Explain the characteristics that distinguish the database approach from the traditional approach of programming with data files.
2. Cite the basic goals, functions, models, components, applications, and social impact of database systems.
3. Describe the components of a database system and give examples of their use.
4. Identify major DBMS functions and describe their role in a database system.
5. Explain the concept of data independence and its importance in a database system.
6. Use a declarative query language to elicit information from a database.

## IM/DataModeling [core]
Minimum core *coverage time: 4 hours*

*Topics:*
- Data modeling
- Conceptual models (such as entity-relationship or UML)
- Object-oriented model
- Relational data model
- Semistructured data model (expressed using DTD or XMLSchema, for example)

*Learning objectives:*
1. Categorize data models based on the types of concepts that they provide to describe the database structure—that is, conceptual data model, physical data model, and representational data model.
2. Describe the modeling concepts and notation of the entity-relationship model and UML, including their use in data modeling.
3. Describe the main concepts of the OO model such as object identity, type constructors, encapsulation, inheritance, polymorphism, and versioning.
4. Define the fundamental terminology used in the relational data model .
5. Describe the basic principles of the relational data model.
6. Illustrate the modeling concepts and notation of the relational data model.
7. Describe the differences between relational and semistructured data models
8. Give a semistructured equivalent (eg in DTD or XMLSchema) for a given relational schema

## IM/Indexing [Elective]

*Topics:*
- The massive impact of indexes on query performance
- The basic structure of an index;
- Keeping a buffer of data in memory;
- Creating indexes with SQL;
- Indexing text;
- Indexing the web (how search engines work)

*Learning Objectives*
1. Generate an index file for a collection of resources.
2. Explain the role of an inverted index in locating a document in a collection
3. Explain how stemming and stop words affect indexing
4. Identify appropriate indices for given relational schema and query set
5. Estimate time to retrieve information, when indices are used compared to when they are not used.

## IM/RelationalDatabases [elective]

*Topics:*
- Mapping conceptual schema to a relational schema
- Entity and referential integrity
- Relational algebra and relational calculus

*Learning objectives:*
1. Prepare a relational schema from a conceptual model developed using the entity- relationship model
2. Explain and demonstrate the concepts of entity integrity constraint and referential integrity constraint (including definition of the concept of a foreign key).
3. Demonstrate use of the relational algebra operations from mathematical set theory (union, intersection, difference, and cartesian product) and the relational algebra operations developed specifically for relational databases (select (restrict), project, join, and division).
4. Demonstrate queries in the relational algebra..
5. Demonstrate queries in the tuple relational calculus.

## IM/QueryLanguages [elective]

*Topics:*
- Overview of database languages
- SQL (data definition, query formulation, update sublanguage, constraints, integrity)
- QBE and 4th-generation environments
- Embedding non-procedural queries in a procedural language
- Introduction to Object Query Language
- Stored procedures

*Learning objectives:*
1. Create a relational database schema in SQL that incorporates key, entity integrity, and referential integrity constraints.
2. Demonstrate data definition in SQL and retrieving information from a database using the SQL SELECT statement.
3. Evaluate a set of query processing strategies and select the optimal strategy.
4. Create a non-procedural query by filling in templates of relations to construct an example of the desired query result.
5. Embed object-oriented queries into a stand-alone language such as C++ or Java (e.g., SELECT Col.Method() FROM Object).
6. Write a stored procedure that deals with parameters and has some control flow, to provide a given functionality

## IM/RelationalDatabaseDesign[elective]

*Topics:*
- Database design
- Functional dependency
- Decomposition of a schema; lossless-join and dependency-preservation properties of a decomposition
- Candidate keys, superkeys, and closure of a set of attributes
- Normal forms (1NF, 2NF, 3NF, BCNF)
- Multivalued dependency (4NF)
- Join dependency (PJNF, 5NF)
- Representation theory

*Learning objectives:*
1. Determine the functional dependency between two or more attributes that are a subset of a relation.
2. Connect constraints expressed as primary key and foreign key, with functional dependencies
3. Compute the closure of a set of attributes under given functional dependencies
4. Determine whether or not a set of attributes form a superkey and/or candidate key for a relation with given functional dependencies
5. Evaluate a proposed decomposition, to say whether or not it has lossless-join and dependency-preservation
6. Describe what is meant by 1NF, 2NF, 3NF, and BCNF.
7. Identify whether a relation is in 1NF, 2NF, 3NF, or BCNF.
8. Normalize a 1NF relation into a set of 3NF (or BCNF) relations and denormalize a relational schema.
9. Explain the impact of normalization on the efficiency of database operations, especially query optimization.
10. Describe what is a multivalued dependency and what type of constraints it specifies.
11. Explain why 4NF is useful in schema design.

## IM/TransactionProcessing [elective]

*Topics:*

- Transactions
- Failure and recovery
- Concurrency control

*Learning objectives:*

1. Create a transaction by embedding SQL into an application program.
2. Explain the concept of implicit commits.
3. Describe the issues specific to efficient transaction execution.
4. Explain when and why rollback is needed and how logging assures proper rollback.
5. Explain the effect of different isolation levels on the concurrency control mechanisms.
6. Choose the proper isolation level for implementing a specified transaction protocol.

## IM/DistributedDatabases [elective]

*Topics:*

- Distributed data storage
- Distributed query processing
- Distributed transaction model
- Concurrency control
- Homogeneous and heterogeneous solutions
- Client-server

*Learning objectives:*

1. Explain the techniques used for data fragmentation, replication, and allocation during the distributed database design process.
2. Evaluate simple strategies for executing a distributed query to select the strategy that minimizes the amount of data transfer.
3. Explain how the two-phase commit protocol is used to deal with committing a transaction that accesses databases stored on multiple nodes.
4. Describe distributed concurrency control based on the distinguished copy techniques and the voting method.
5. Describe the three levels of software in the client-server model.

## IM/PhysicalDatabaseDesign [elective]

*Topics:*

- Storage and file structure
- Indexed files
- Hashed files
- Signature files
- B-trees
- Files with dense index
- Files with variable length records
- Database efficiency and tuning

*Learning objectives:*

1. Explain the concepts of records, record types, and files, as well as the different techniques for placing file records on disk.
2. Give examples of the application of primary, secondary, and clustering indexes.
3. Distinguish between a nondense index and a dense index.
4. Implement dynamic multilevel indexes using B-trees.
5. Explain the theory and application of internal and external hashing techniques.
6. Use hashing to facilitate dynamic file expansion.
7. Describe the relationships among hashing, compression, and efficient database searches.
8. Evaluate costs and benefits of various hashing schemes.
9. Explain how physical database design affects database transaction efficiency.

## IM/DataMining [elective]

*Topics:*
- The usefulness of data mining
- Associative and sequential patterns
- Data clustering
- Market basket analysis
- Data cleaning
- Data visualization

*Learning objectives:*
1. Compare and contrast different conceptions of data mining as evidenced in both research and application.
2. Explain the role of finding associations in commercial market basket data.
3. Characterize the kinds of patterns that can be discovered by association rule mining.
4. Describe how to extend a relational system to find patterns using association rules.
5. Evaluate methodological issues underlying the effective application of data mining.
6. Identify and characterize sources of noise, redundancy, and outliers in presented data.
7. Identify mechanisms (on-line aggregation, anytime behavior, interactive visualization) to close the loop in the data mining process.
8. Describe why the various close-the-loop processes improve the effectiveness of data mining.

## IM/InformationStorageAndRetrieval [elective]

*Topics:*
- Characters, strings, coding, text
- Documents, electronic publishing, markup, and markup languages
- Tries, inverted files, PAT trees, signature files, indexing
- Morphological analysis, stemming, phrases, stop lists
- Term frequency distributions, uncertainty, fuzziness, weighting
- Vector space, probabilistic, logical, and advanced models
- Information needs, relevance, evaluation, effectiveness
- Thesauri, ontologies, classification and categorization, metadata
- Bibliographic information, bibliometrics, citations
- Routing and (community) filtering
- Search and search strategy, information seeking behavior, user modeling, feedback
- Information summarization and visualization
- Integration of citation, keyword, classification scheme, and other terms
- Protocols and systems (including Z39.50, OPACs, WWW engines, research systems)

*Learning objectives:*
1. Explain basic information storage and retrieval concepts.
2. Describe what issues are specific to efficient information retrieval.
3. Give applications of alternative search strategies and explain why the particular search strategy is appropriate for the application.
4. Perform Internet-based research.
5. Design and implement a small to medium size information storage and retrieval system.

## IM/Hypermedia [elective]

*Topics:*
- Hypertext models (early history, web, Dexter, Amsterdam, HyTime)
- Link services, engines, and (distributed) hypertext architectures
- Nodes, composites, and anchors
- Dimensions, units, locations, spans
- Browsing, navigation, views, zooming

- Automatic link generation
- Presentation, transformations, synchronization
- Authoring, reading, and annotation
- Protocols and systems (including web, HTTP)

*Learning objectives:*
1. Summarize the evolution of hypertext and hypermedia models from early versions up through current offerings, distinguishing their respective capabilities and limitations.
2. Explain basic hypertext and hypermedia concepts.
3. Demonstrate a fundamental understanding of information presentation, transformation, and synchronization.
4. Compare and contrast hypermedia delivery based on protocols and systems used.
5. Design and implement web-enabled information retrieval applications using appropriate authoring tools.

## IM/MultimediaSystems [elective]

*Topics:*
- Devices, device drivers, control signals and protocols, DSPs
- Applications, media editors, authoring systems, and authoring
- Streams/structures, capture/represent/transform, spaces/domains, compression/coding
- Content-based analysis, indexing, and retrieval of audio, images, and video
- Presentation, rendering, synchronization, multi-modal integration/interfaces
- Real-time delivery, quality of service, audio/video conferencing, video-on-demand

*Learning objectives:*
1. Describe the media and supporting devices commonly associated with multimedia information and systems.
2. Explain basic multimedia presentation concepts.
3. Demonstrate the use of content-based information analysis in a multimedia information system.
4. Critique multimedia presentations in terms of their appropriate use of audio, video, graphics, color, and other information presentation concepts.
5. Implement a multimedia application using a commercial authoring system.

## IM/DigitalLibraries [elective]

*Topics:*
- Digitization, storage, and interchange
- Digital objects, composites, and packages
- Metadata, cataloging, author submission
- Naming, repositories, archives
- Spaces (conceptual, geographical, 2/3D, VR)
- Architectures (agents, buses, wrappers/mediators), interoperability
- Services (searching, linking, browsing, and so forth)
- Intellectual property rights management, privacy, protection (watermarking)
- Archiving and preservation, integrity

*Learning objectives:*
- Explain the underlying technical concepts in building a digital library.
- Describe the basic service requirements for searching, linking, and browsing.
- Critique scenarios involving appropriate and inappropriate use of a digital library, and determine the social, legal, and economic consequences for each scenario.
- Describe some of the technical solutions to the problems related to archiving and preserving information in a digital library.
- Design and implement a small digital library.

## Social and Professional Issues (SP)

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

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

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

*Students also need to develop the ability to ask serious questions about the social impact of computing and to evaluate proposed answers to those questions. Future practitioners must be able to anticipate the impact of introducing a given product into a given environment. Will that product enhance or degrade the quality of life? What will the impact be upon individuals, groups, and institutions?*

*Finally, students need to be aware of the basic legal rights of software and hardware vendors and users, and they also need to appreciate the ethical values that are the basis for those rights. Future practitioners must understand the responsibility that they will bear, and the possible consequences of failure. They must understand their own limitations as well as the limitations of their tools. All practitioners must make a long-term commitment to remaining current in their chosen specialties and in the discipline of computing as a whole.*

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

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

There is an unresolved pedagogical conflict between having the core course at the lower (freshman-sophomore) level versus the upper (junior-senior) level. Having the course at the lower level

1. Allows for coverage of methods and tools of analysis (SP3) prior to analyzing ethical issues in the context of different technical areas
2. Assures that students who drop out early to enter the workforce will still be introduced to some professional and ethical issues.

On the other hand, placing the course too early may lead to the following problems:

1. Lower-level students may not have the technical knowledge and intellectual maturity to support in-depth ethical analysis. Without basic understanding of technical alternatives, it is difficult to consider their ethical implications.
2. Students need a certain level of maturity and sophistication to appreciate the background and issues involved. For that reason, students should have completed at least the discrete mathematics course and the second computer science course. Also, if students take a technical writing course, it should be a prerequisite or corequisite for the required course in the SP area.
3. Some programs may wish to use the course as a “capstone” experience for seniors.

Although items SP2 and SP3 are listed with a number of hours associated, they are fundamental to all the other topics. Thus, when covering the other areas, instructors should continually be aware of the social context issues and the ethical analysis skills. In practice, this means that the topics in SP2 and SP3 will be continually reinforced as the material in the other areas is covered.

**SP. Social and Professional Issues (16 core hours)**

**SP/HistoryOfComputing [core]**
**SP/SocialContext [core]**
**SP/AnalyticalTools [core]**
**SP/ProfessionalEthics [core]**
**SP/Risks [core]**
**SP/SecurityOperations [elective]**
**SP/IntellectualProperty [core]**
**SP/PrivacyAndCivilLiberties [core]**
**SP/ComputerCrime [elective]**
**SP/EconomicsOfComputing [elective]**
**SP/PhilosophicalFrameworks [elective]**

## SP/HistoryOfComputing [core]

Minimum core *coverage time: 1 hour*

*Topics:*

- Prehistory—the world before 1946
- History of computer hardware, software, networking
- Pioneers of computing

*Learning objectives:*

1. List the contributions of several pioneers in the computing field.
2. Compare daily life before and after the advent of personal computers and the Internet.
3. Identify significant continuing trends in the history of the computing field.

## SP/SocialContext [core]

Minimum core *coverage time: 3 hours*

*Topics:*

- Introduction to the social implications of computing
- Social implications of networked communication
- Growth of, control of, and access to the Internet
- Gender-related issues
- Cultural issues
- International issues
- Accessibility issues (e.g. underrepresentation of minorities, women and the disabled in the computing profession)
- Public policy issues (e.g. electronic voting)

*Learning objectives:*

1. Interpret the social context of a particular implementation.
2. Identify assumptions and values embedded in a particular design including those of a cultural nature.
3. Evaluate a particular implementation through the use of empirical data.
4. Describe positive and negative ways in which computing alters the modes of interaction between people.
5. Explain why computing/network access is restricted in some countries.
6. Indicate the role of cultural issues in considering team-work.
7. Analyze the role and risks of computing in the implementation of public policy and government (e.g. electronic voting).
8. Articulate the impact of the input deficit from diverse populations in the computing profession.

## SP/AnalyticalTools [core]
Minimum core *coverage time: 2 hours*

*Topics:*
- Making and evaluating ethical arguments
- Identifying and evaluating ethical choices
- Understanding the social context of design
- Identifying assumptions and values

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

## SP/ProfessionalEthics [core]
Minimum core *coverage time: 3 hours*

*Topics:*
- Community values and the laws by which we live
- The nature of professionalism (including care, attention and discipline, fiduciary responsibility, and mentoring)
- Keeping up-to-date as a professional (in terms of knowledge, tools, skills, legal and professional framework as well as the ability to self-assess and computer fluency)
- Various forms of professional credentialing and the advantages and disadvantages
- The role of the professional in public policy
- Maintaining awareness of consequences
- Ethical dissent and whistle-blowing
- Codes of ethics, conduct, and practice (IEEE, ACM, SE, AITP, and so forth)
- Dealing with harassment and discrimination
- "Acceptable use" policies for computing in the workplace
- Healthy computing environment (ergonomics)

*Learning objectives:*
1. Identify progressive stages in a whistle-blowing incident.
2. Specify the strengths and weaknesses of relevant professional codes as expressions of professionalism and guides to decision-making.
3. Identify ethical issues that arise in software development and determine how to address them technically and ethically.
4. Develop a computer use policy with enforcement measures.
5. Analyze a global computing issue, observing the role of professionals and government officials in managing the problem.
6. Evaluate the professional codes of ethics from the ACM, the IEEE Computer Society, and other organizations.
7. Describe the mechanisms that typically exist for a professional to keep up-to-date.
8. Identify the social implications of ergonomic devices and the workplace environment to people's health.

## SP/Risks [core]
Minimum core *coverage time: 2 hours*

*Topics:*
- Historical examples of software risks (such as the Therac-25 case) Implications of software complexity
- Risk assessment and risk management; risk removal, risk reduction and risk control

*Learning objectives:*
1. Explain the limitations of testing as a means to ensure correctness.
2. Describe the differences between correctness, reliability, and safety.
3. Discuss the potential for hidden problems in reuse of existing components.
4. Describe current approaches to managing risk, and characterize the strengths and shortcomings of each.

5. Outline the role of risk management in systems design and construction.

## SP/SecurityOperations [elective]

*Topics:*
- Physical security
- Physical access controls
- Personnel access controls
- Operational security
- Security policies for systems/networks
- Recovery and response
- Dealing with problems (both technical and human)

*Learning objectives:*
1. Develop an incident-recovery plan for handling system compromises for an organization
2. Analyze stated security procedures for "weak points" that an attacker could exploit, and explain how they could (or will) fail
3. Propose appropriate security measures for different situations
4. Explain to a non-security community of users what measures they must follow and why, in a situation where their jobs are not security-related

## SP/IntellectualProperty [core]
Minimum core *coverage time: 3 hours*

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

*Learning objectives:*
1. Distinguish among patent, copyright, and trade secret protection.
2. Discuss the legal background of copyright in national and international law.
3. Explain how patent and copyright laws may vary internationally.
4. Outline the historical development of software patents.
5. Discuss the consequences of software piracy on software developers and the role of relevant enforcement organizations.

## SP/PrivacyAndCivilLiberties [core]
Minimum core *coverage time: 2 hours*

*Topics:*
- Ethical and legal basis for privacy protection
- Ethical and legal framework for freedom of information
- Privacy implications of database systems (e.g. data gathering, storage, and sharing, massive data collecting, computer surveillance systems)
- Technological strategies for privacy protection
- Freedom of expression in cyberspace
- International and intercultural implications

*Learning objectives:*
1. Summarize the legal bases for the right to privacy and freedom of expression in one’s own nation and how those concepts vary from country to country.
2. Describe current computer-based threats to privacy.
3. Explain how the Internet may change the historical balance in protecting freedom of expression.
4. Describe trends in privacy protection as exemplified in technology.

5. Clarify the apparent conflict between the requirements of freedom of information and the protection of the rights of the individual.

## SP/ComputerCrime [elective]

*Topics:*

- History and examples of computer crime
- "Cracking" ("hacking") and its effects
- Viruses, worms, and Trojan horses
- Identity theft
- Crime prevention strategies

*Learning objectives:*

1. Describe trends in privacy protection as exemplified in technologyOutline the technical basis of viruses and denial-of-service attacks.
2. Enumerate techniques to combat "cracker" attacks.
3. Discuss several different "cracker" approaches and motivations.
4. Identify the professional's role in security and the tradeoffs involved.
5. Indicate measure to be taken both by individuals themselves and by organizations (including government) to prevent identity theft.

## SP/EconomicsOfComputing [elective]

*Topics:*

- Monopolies and their economic implications
- Effect of skilled labor supply and demand on the quality of computing products
- Pricing strategies in the computing domain
- The phenomenon of outsourcing and offshoring; impacts on employment and on economics
- Differences in access to computing resources and the possible effects thereof
- Environmental sustainability

*Learning objectives:*

1. Summarize the rationale for antimonopoly efforts.
2. Describe several ways in which the information technology industry is affected by shortages in the labor supply.
3. Suggest and defend ways to address limitations on access to computing.
4. Outline the evolution of pricing strategies for computing goods and services.
5. Discuss the benefits, the drawbacks and the implications of offshoring and outsourcing.
6. Identify ways to support environmental computing (e.g. green operations, recyclable products, reduced green house emissions).

## SP/PhilosophicalFrameworks [elective]

*Topics:*

- Philosophical frameworks, particularly utilitarianism and deontological theories
- Problems of ethical relativism
- Scientific ethics in historical perspective
- Differences in scientific and philosophical approaches

*Learning objectives:*

1. Summarize the basic concepts of relativism, utilitarianism, and deontological theories.
2. Recognize the distinction between ethical theory and professional ethics.
3. Identify the weaknesses of the "hired agent" approach, strict legalism, naïve egoism, and naïve relativism as ethical frameworks.

# Software Engineering (SE)

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

Software engineering employs engineering methods, processes, techniques, and measurement. It benefits from the use of tools for managing software development; analyzing and modeling software artifacts; assessing and controlling quality; and for ensuring a disciplined, controlled approach to software evolution and reuse. Software development, which can involve an individual developer or a team of developers, requires choosing the tools, methods, and approaches that are most applicable for a given development environment. The SE toolbox has evolved over the years; for instance, the use of contracts (such as a ‘requires’ clause, an ‘ensures’ clause, class invariants, etc.) is now regarded as good practice.

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

**SE. Software Engineering (31 core hours)**

- **SE/SoftwareDesign [core]**
- **SE/UsingAPIs [core]**
- **SE/ToolsAndEnvironments [core]**
- **SE/SoftwareProcesses [core]**
- **SE/RequirementsSpecifications [core]**
- **SE/SoftwareValidation [core]**
- **SE/SoftwareEvolution [core]**
- **SE/SoftwareProjectManagement [core]**
- **SE/ComponentBasedComputing [elective]**
- **SE/FormalMethods [elective]**
- **SE/SoftwareReliability [elective]**
- **SE/SpecializedSystems [elective]**
- **SE/RiskAssessment [elective]**

## SE/SoftwareDesign [core]

Minimum core *coverage time: 8 hours*

*Topics:*

- Fundamental design concepts and principles
- The role and the use of contracts
- Design patterns
- Software architecture
- Structured design
- Object-oriented analysis and design
- Component-level design
- Design qualities
- Internal including low coupling, high cohesion, information hiding, efficiency
- External including reliability, maintainability, usability, performance
- Other approaches: data-structured centered, aspect oriented, function oriented, service oriented, agile
- Design for reuse
- Use of open-source materials

*Learning objectives:*

1. Discuss the properties of good software design including the nature and the role of associated documentation.
2. Evaluate the quality of multiple software designs based on key design principles and concepts.
3. Select and apply appropriate design patterns in the construction of a software application.

4. Create and specify the software design for a medium-size software product using a software requirement specification, an accepted program design methodology (e.g., structured or object-oriented), and appropriate design notation.
5. Conduct a software design review of open-source materials using appropriate guidelines.
6. Evaluate a software design at the component level.
7. Evaluate a software design from the perspective of reuse.

## SE/UsingAPIs [core]

Minimum core *coverage time: 3 hours*

*Topics:*

- Programming using APIs
- Design of APIs
- Class browsers and related tools
- Debugging in the API environment Introduction to component-based computing

Note: see also SE/RequirementsSpecifications and SE/ComponentBasedComputing

*Learning objectives:*

1. Explain the value of application programming interfaces (APIs) in software development.
2. Use class browsers and related tools during the development of applications using APIs.
3. Design, implement, test, and debug programs that use large-scale API packages.

## SE/ToolsAndEnvironments [core]

Minimum core *coverage time: 3 hours*

*Topics:*

- Programming environments
- Requirements analysis and design modeling tools
- Testing tools including static and dynamic analysis tools
- Tools for source control, and their use in particular in team-work
- Configuration management and version control tools
- Tool integration mechanisms

*Learning objectives:*

1. Select, with justification, an appropriate set of tools to support the development of a range of software products.
2. Analyze and evaluate a set of tools in a given area of software development (e.g., management, modeling, or testing).
3. Demonstrate the capability to use a range of software tools in support of the development of a software product of medium size.

## SE/SoftwareProcesses [core]

Minimum core *coverage time: 2 hours*

*Topics:*

- Software life-cycle and process models
- Software process capability maturity models
- Approaches to process improvement
- Process assessment models
- Software process measurements

*Learning objectives:*

1. Explain the concept of a software life cycle and provide an example, illustrating its phases including the deliverables that are produced.
2. Select, with justification the software development models and process elements most appropriate for the development and maintenance of a diverse range of software products.
3. Explain the role of process maturity models.

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

## SE/RequirementsSpecifications [core]

Minimum core *coverage time: 4 hours*

*Topics:*

- Systems level considerations
- Software requirements elicitation
- Requirements analysis modeling techniques
- Functional and non-functional requirements
- Acceptability of certainty / uncertainty considerations regarding software / system behaviour
- Prototyping
- Basic concepts of formal specification techniques

*Learning objectives:*

1. Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system.
2. Discuss the challenges of maintaining legacy software.
3. Use a common, non-formal method to model and specify (in the form of a requirements specification document) the requirements for a medium-size software system.
4. Conduct a review of a software requirements document using best practices to determine the quality of the document.
5. Translate into natural language a software requirements specification (e.g., a software component contract) written in a formal specification language.

## SE/SoftwareVerificationValidation [core]

Minimum core *coverage time: 3 hours*

*Topics:*

- Distinguishing between verification and validation
- Static approaches and dynamic approaches
- Validation planning; documentation for validation
- Different kinds of testing – human computer interface, usability, reliability, security, conforman to specification
- Testing fundamentals, including test plan creation and test case generation black-box and white-box testing techniques
- Defect seeding
- Unit, integration, validation, and system testing
- Object-oriented testing; systems testing
- Measurements: process, design, program
- Verification and validation of non-code (documentation, help files, training materials)
- Fault logging, fault tracking and technical support for such activities
- Regression testing
- Inspections, reviews, audits

*Learning objectives:*

1. Distinguish between program validation and verification.
2. Describe the role that tools can play in the validation of software.
3. Distinguish between the different types and levels of testing (unit, integration, systems, and acceptance) for medium-size software products and related materials.
4. Create, evaluate, and implement a test plan for a medium-size code segment.
5. Undertake, as part of a team activity, an inspection of a medium-size code segment.
6. Discuss the issues involving the testing of object-oriented software.

## SE/SoftwareEvolution [core]
Minimum core *coverage time: 3 hours*

*Topics:*
- Software maintenance
- Characteristics of maintainable software
- Reengineering Legacy systems
- Refactoring
- Software reuse

*Learning objectives:*
1. Identify the principal issues associated with software evolution and explain their impact on the software life cycle.
2. Discuss the challenges of maintaining legacy systems and the need for reverse engineering.
3. Outline the process of regression testing and its role in release management.
4. Estimate the impact of a change request to an existing product of medium size.
5. Develop a plan for re-engineering a medium-sized product in response to a change request.
6. Discuss the advantages and disadvantages of software reuse.
7. Exploit opportunities for software reuse in a given context.
8. Identify weaknesses in a given simple design, and highlight how they can be removed through refactoring.

## SE/SoftwareProjectManagement [core]
Minimum core *coverage time: 3 hours*

*Topics:*
- Team management
    - Team processes
    - Team organization and decision-making
    - Roles and responsibilities in a software team
    - Role identification and assignment
    - Project tracking
    - Team problem resolution
- Project scheduling
- Software measurement and estimation techniques
- Risk analysis
    - The issue of security
    - High integrity systems, safety critical systems
    - The role of risk in the life cycle
- Software quality assurance
    - The role of measurements
- Software configuration management and version control; release management
- Project management tools
- Software process models and process measurements

*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. Indicate an approach to risk that will help to secure the on-time delivery of software.
4. Compare and contrast the different methods and techniques used to assure the quality of a software product.

## SE/ComponentBasedComputing [elective]

*Topics:*
- Fundamentals
    - The definition and nature of components
    - Components and interfaces

    - Interfaces as contracts
    - The benefits of components
    - Basic techniques
    - Component design and assembly
    - Relationship with the client-server model and with patterns
    - Use of objects and object lifecycle services
    - Use of object brokers
    - Marshalling
- Applications (including the use of mobile components)
- Patterns as used in analysis and design; context of use including enterprise architectures
- Architecture of component-based systems
- Component-oriented design
- Application frameworks
- Event handling: detection, notification, and response
- Middleware
    - The object-oriented paradigm within middleware
    - Object request brokers
    - Transaction processing monitors
    - Workflow systems
    - State-of-the-art tools

*Learning objectives:*
1. Explain and apply recognized principles to the building of high-quality software components.
2. Discuss and select an architecture for a component-based system suitable for a given scenario.
3. Identify the kind of event handling implemented in one or more given APIs.
4. Explain the role of objects in middleware systems and the relationship with components.
5. Apply component-oriented approaches to the design of a range of software including those required for concurrency and transactions, reliable communication services, database interaction including services for remote query and database management, secure communication and access.

## SE/FormalMethods [elective]

*Topics:*
- Formal methods concepts
- Formal specification languages
- Model checking
- Executable and non-executable specifications
- Pre and post assertions
- Formal verification
- Tools in support of formal methods

*Learning objectives:*
1. Apply formal verification techniques to software segments with low complexity.
2. Discuss the role of formal verification techniques in the context of software validation and testing, and compare the benefits with those of model checking.
3. Explain the potential benefits and drawbacks of using formal specification languages.
4. Create and evaluate pre- and post-assertions for a variety of situations ranging from simple through complex.
5. Using a common formal specification language, formulate the specification of a simple software system and demonstrate the benefits from a quality perspective.

## SE/SoftwareReliability [elective]

*Topics:*
- Software reliability models
- Redundancy and fault tolerance
- Defect classification
- Probabilistic methods of analysis

*Learning objectives:*
1. Demonstrate the ability to apply multiple methods to develop reliability estimates for a software system.
2. Identify and apply redundancy and fault tolerance for a medium-sized application.
3. Explain the problems that exist in achieving very high levels of reliability.
4. Identify methods that will lead to the realization of a software architecture that achieves a specified reliability level.

## SE/SpecializedSystems [elective]

*Topics:*
- Real-time systems
- Client-server systems
- Distributed systems
- Parallel systems
- Web-based systems
- High-integrity systems

*Learning objectives:*
1. Identify and discuss different specialized systems.
2. Discuss life cycle and software process issues in the context of software systems designed for a specialized context, including systems that may have to operate in a degraded mode of operation.
3. Select, with appropriate justification, approaches that will result in the efficient and effective development and maintenance of specialized software systems.
4. Given a specific context and a set of related professional issues, discuss how a software engineer involved in the development of specialized systems should respond to those issues.
5. Outline the central technical issues associated with the implementation of specialized systems development.

## SE/RiskAssessment [Elective]

*Topics:*
- Definition of terms: in security, vulnerability, threat, security breach; in safety, hazard.
- The concept of risk; hazard and risk identification
- Risk analysis including evaluation
- Need for a system-wide approach including hazards associated with tools
- Risk and immature technologies
- Cost/benefit analysis
- Principles of risk management

*Learning objectives:*
1. To define the concepts of hazard and risk, hazard
2. To recognize common security risks in at least two operating systems
3. To describe the categories of threats to networked computing systems
4. To display a systematic approach to the task of identifying hazards and risks in a particular situation
5. To apply the basic principles of risk management in a variety of simple scenarios including a security situation

## SE/RobustAndSecurity-EnhancedProgramming [elective]
*Topics:*
- Defensive programming
    - Principles of secure design and coding:
    - Principle of least privilege
    - Principle of fail-safe defaults
- Principle of psychological acceptability
    - How to detect potential security problems in programs
    - Buffer and other types of overflows
    - Race conditions
    - Improper initialization, including choice of privileges

    - Checking input
    - Assuming success and correctness
    - Validating assumptions
- How to document security considerations in using a program

*Learning objectives:*

1. Rewrite a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows, and race conditions
2. State and apply the principles of least privilege and fail-safe defaults.
3. Write a simple library that performs some non-trivial task and will not terminate the calling program regardless of how it is called

# Computational Science (CN)

From the earliest days of the discipline, the techniques of computational science have constituted a major area of computer science research. As computers increase in their problem-solving power, this area—like much of the discipline—has grown in both breadth and importance. At the present time, scientific computational science stands as an intellectual discipline in its own right, closely related to but nonetheless distinct from computer science.

Although courses in computational science are extremely valuable components of an undergraduate program in computer science, the CS2001 Task Forces believe that none of the topics in this area represent core knowledge. From our surveys of curricula and interaction with the computer science education community, we are convinced no consensus exists that this material is essential for all CS undergraduates. It remains a vital part of the discipline, but need not be a part of every program.

For those who choose to pursue it, this area offers exposure to many valuable ideas and techniques, including precision of numerical representation, error analysis, numerical techniques, parallel architectures and algorithms, modeling and simulation, and scientific visualization. At the same time, students who take courses in this area have an opportunity to apply these techniques in a wide range of application areas, such as the following:

- Molecular dynamics
- Fluid dynamics
- Celestial mechanics
- Economic forecasting
- Optimization problems
- Structural analysis of materials
- Bioinformatics
- Computational biology
- Geologic modeling
- Computerized tomography

Each of the units in this area corresponds to a full-semester course at most institutions. The level of specification of the topic descriptions and the learning objectives is therefore different from that used in other areas in which the individual units typically require smaller blocks of time.

**CN. Computational Science (no core hours)**
**CN/ModelingAndSimulation [elective]**
**CN/OperationsResearch [elective]**
**CN/ParallelComputation [elective]**

## CN/ModelingAndSimulation [elective]

*Topics:*

- Definition of simulation and modeling; relationship between simulation and modeling
- Purpose including benefits and limitations: role – addressing performance, optimization; supporting decision making, forecasting, safety considerations; for training and education
- Important application areas: healthcare (including assisting with diagnostics); economics and finance; classroom of the future; training and education; city and urban simulations; simulation in science and in engineering; games; military simulation
- Different kinds of simulations – physical, human in the loop, interaction, computer, virtual reality
- The simulation process – sound basis, identification of key characteristics or behaviors, simplifying assumptions; validation of outcomes. Model building: use of mathematical formula or equation, graphs, constraints. Methodologies and techniques. Use of time stepping for dynamic systems
- Theoretical considerations; Monte Carlo methods, stochastic processes, queuing theory
- Technologies in support of simulation and modeling: graphics processors; Haptic feedback devices. Human computer interaction considerations.
- Assessing and evaluating simulations in a variety of contexts.
- Software in support of simulation and modeling; packages, languages

*Learning objectives:*
1. Explain the benefits of simulation and modeling in a range of important application areas.
2. Demonstrate the ability to apply the techniques of modeling and simulation to a range of problem areas.
3. Evaluate a simulation, highlighting the benefits and the drawbacks.

## CN/Operations Research [elective]

*Topics:*
- Linear programming
    - Integer programming
    - The Simplex method
- Probabilistic modeling
- Queuing theory
    - Petri nets
    - Markov models and chains
- Optimization
- Network analysis and routing algorithms
- Prediction and estimation
    - Decision analysis
    - Forecasting
    - Risk management
    - Econometrics, microeconomics
    - Sensitivity analysis
- Dynamic programming
- Sample applications
- Software tools

*Learning objectives:*
1. Apply the fundamental techniques of operations research.
2. Describe several established techniques for prediction and estimation.
3. Design, code, test, and debug application programs to solve problems in the domain of operations research.

## CN/Parallel Computation [elective]

*Topics:*
- Overview of topics
- Models of computation
- Kinds of computation
- Task parallelism
- Data parallelism
- Event parallelism
- Properties of computation
- Bandwidth
- Latency
- Scalability
- Granularity
- Parallel architectures
- Processor architectures including multi-core
- Memory systems for high performance
- Caching and coherence
- Clusters
- Parallel programming paradigms
- Threading
- Message passing
- Event driven techniques

- Parallel software architectures
    - MapReduce
- Grid computing
- Open community distributed computing (BOINC, SETI, …)

*Learning objectives:*

1. Compare and contrast architectures for parallel computing, recognizing the strengths and weaknesses of each
2. Compare and contrast parallel programming paradigms recognizing the strengths and weaknesses of each
3. Identify the basic properties of bandwidth, latency, scalability and granularity
4. Design, code, test and debug programs for parallel computation

# Appendix C - Course Guidance

By way of guidance, the Review Task Force is providing guidance in two different areas:

- An introduction to security
- An introduction to computer security

In some sense these are intended to reflect areas of concern in the curriculum and changes in emphasis since 2001.

## C.1 Introduction to Security

This course is a survey of the fundamentals of information assurance and computer security. Topics include: security standards, policies and best practices; principles of ethical and professional behavior; regulatory compliance and legal investigations; information assurance; risk management and threat assessment; business continuity and disaster recovery planning; security architecture and design; elements of cryptography; digital forensics; physical (environmental) security; networking fundamentals; access control and authentication; network and application security; exploiting network, web, software and insider vulnerabilities.

## CS3xx Introduction to Computer Security

Prerequisite: a course that provides an introduction to programming building on a broad introduction to computer science; in the context of CS2001 such a course would be CS 102
Co-requisite: a programming course that provides an introduction to data structures and algorithms; in the context of CS2001 such a course would be CS 103

Syllabus:

- Security Goals, Fundamentals (confidentiality, integrity, availability, etc.)
- Introduction to risk assessment and management
- Security standards in government and industry (Common Criteria/Orange Book, sample corporate and institutional security policies)
- Computer system protection principles (UNIX and Windows)
- Access controls, including MAC, DAC, and role-based
- Networking fundamentals (Internet, TCP/IP network services)
- Cryptography fundamentals
- Authentication, passwords, introduction to protocols, Kerberos
- Security operations
- Attacks: software attacks, malicious code, buffer overflows, social engineering, injection attacks, and related defense tools
- Network attacks: Denial of service, flooding, sniffing and traffic redirection, defense tools and strategies
- Attacking web sites: cross-site scripting
- IPSec, Virtual Private networks and Network Address Translation
- Ethics, SP issues that are related
- Drop on Forensics

## C.2 Course on Parallelism

The class is a suitable prerequisite to a capstone project class focusing on creating a complex parallel system. It cuts across several advance courses of CS2001, repackaging the material to eliminate more specialized information and emphasizing content that is most applicable to current students: CS314 Parallel Algorithms, Parallel Architectures, CS326 Concurrent and Distributed Systems, CS333 Cluster Computing, CS343 Programming Paradigms, CS344 Functional Programming, CS372 Transaction Processing, CS390 Advanced Software Development.

## CS3xx Parallel Computation

A survey of parallel computation across hardware, software, programming languages, algorithms and applications areas. The emphasis is on learning enough context in each topic area to support effective parallel programming.
Prerequisites: CS112A Programming Methodology, CS210 Algorithm Design and Analysis, CS221 Architecture and Operating Systems

Syllabus:

- Context for present day parallel computing, including everyday parallelism, parallelism in processor architecture, parallel computer structures from multicore to huge clusters, etc.
- Basic concepts: models of parallel computers, differences (and similarities) between parallel and distributed computation, data and task parallelism, threads and processes, latency and bandwidth, locality, concurrent and exclusive reads and writes, dependences, MIMD/SIMD, Amdahl's Law, performance measurements, true speed-up, relative speed-up, efficiency, super-linear speed-up
- Shared memory parallel machine architecture concepts, multi-threading, multi-core, SMP, snooping and directory-based coherency protocols, sequential consistency, private and shared memory
- Distributed memory parallel machine architecture concepts, disjoint address spaces, globalization of address spaces, message passing, synchronous and asynchronous communication
- Interconnection networks, common topologies, bisection bandwidth, bus and cross-bar connectivity
- Basics of threaded parallel computation, including problem partitioning, hazards of concurrency, locking, lock contention, thrashing, privatization, false sharing, serialization
- Algorithmic issues including problem decomposition, memory reference costs, techniques for improving locality, parallel prefix, controlling granularity and dependences, block and cyclic storage organization, tree decompositions, static and dynamic task assignment
- Languages and libraries for threaded parallel programming, POSIX threads, Java threads and memory model, automatic threading systems (OpenMP)
- Languages and libraries for distributed memory parallel programming, PVM and MPI, partitioned global address space languages (PGAS)
- Higher-level approaches including array-based, functional, domain-specific
- Transaction approach to memory consistency, hardware techniques, software techniques, rollback
- Emerging languages including high-productivity parallel languages
- Co-processor techniques including GPU, Cell, FPGA, characteristics of co-processor programming methodologies
- Experimental techniques, measuring performance, computing speed-up, experimental precautions, controlling variation, experimental dangers, reporting performance