The United Nations University

UNU/IIST

International Institute for Software Technology

Towards an Improved Computing Curriculum for San Pablo Catholic University in Peru

Elizabeth Vidal Duarte and Bernhard K. Aichernig

June 2004

UNU-IIST Report No. 304

UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macau, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the Governor of Macau and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endownment Fund. As well as providing two-thirds of the endownment fund, the Macau authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research R, Technical T, Compendia C or Administrative A. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macau or visit UNU-IIST's home page: http://www.iist.unu.edu, if you would like to know more about UNU-IIST and its report series.

Chris George, Acting Director

The United Nations University

UNU/IIST

International Institute for Software Technology

P.O. Box 3058
Macau

Towards an Improved Computing Curriculum for San Pablo Catholic University in Peru

Elizabeth Vidal Duarte and
Bernhard K. Aichernig

Abstract

This report shows the analysis of the current curriculum at San Pablo Catholic University in order to identify possibles areas that could be improved. This analysis is based on the CC2001 and IS2002 curricula recommendations, and on the study of the characteristics of undergraduate CS curricula in two leading universities. As a result of this analysis we present 15 concrete recommendations in order to improve the current curriculum.

Elizabeth Vidal Duarte is a Fellow at UNU/IIST on leave from San Pablo Catholic University, Arequipa - Peru, where she is a lecturer in Informatic Engineering. Her current research interests include requirements engineering using object-oriented development technology and UML. Her email address is elizabeth@iist.unu.edu.

Bernhard K. Aichernig is a Research Fellow at UNU/IIST. He is also an assistant professor at the Institute for Software Technology at the Graz University of Technology in Austria. His research interests include the synergies of testing and formal development methods, techniques of refinement, and requirements engineering supported by formal specification languages. His email address is bka@iist.un.edu.

Contents

1 Introduction

San Pablo Catholic University (UCSP) was established in January 1997 in the city of Arequipa, Peru. Along with it was created the Bachelor study of Informatics Engineering. The Bachelor study of Informatics Engineering consists of ten semesters. In the ten semesters students receive instruction in computing science, social and human formation, studies in math, networks and communications, information systems, research formation and capstone projects. These courses are categorized in basic courses, speciality courses and general formation courses. The details and organization of this current UCSP curriculum is shown in Appendix A.

The objective of this report is to analyze the current UCSP curriculum in order to identify possible areas that could be improved. The analysis is based on the revision of the recommendations for Computer Science programs (CC2001) [13], and, the latest report on the Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems (IS2002) [10]. Both of these reports are relevant for our analysis because the current UCSP curriculum presents characteristics and topics that correspond to both fields.

CC2001 represents the third public draft of the Computing Curricula 2001 project (CC2001) [26], a joint undertaking of the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) to develop curricular guidelines for undergraduate programs in computing. The CC2001 report describes the recommendations specifically for Computer Science (CS) programs.

The first part of this report presents an analysis of CC2001. CC2001 has defined a minimal core consisting of those units for which there is a broad consensus that the corresponding material is essential to anyone obtaining an undergraduate degree in this field. The second part shows the analysis of the IS2002. Here the ACM has been a major organizer for IS2002. IS2002 is a model curriculum for undergraduate degree programs in Information Systems. The model is grounded in a fundamental body of computing and information systems knowledge.

The third part of this report, of this report includes the undergraduate computer science curriculum study in two recognized universities in Computer Science, one in the United Kingdom (UK) an the other in Germany. The study points out its alignment to the CC2001 recommendations, how their advanced courses are driven by research and some relevant features in their curriculum.

The guidelines to analyze the UCSP curriculum have been taken from the study of CC2001 and IS2002 reports, from the study of the curriculums in CS at Leicester and München Universities, discussions with academic staff of UNU-IIST and discussions with academic staff at Leicester University. As a result, a series of recommendations has been obtained, and these should be implemented in the UCSP curriculum in a medium and long term strategy.

This report is structured as follows: we present the review of computer science curriculum recommendations CC2001 in Section 2. We study the model curriculum and guidelines for undergraduate degree programs in Information Systems in Section 3. Section 4 shows the detailed analysis of the UCSP curriculum based on CC2201 and IS2002 reports. Section 5 shows the review of computer science curriculum in some leading universities at a world-class level, analyzing some relevant features. Section 6 concludes the report with general recommendations to San Pablo Catholic University.

2 Review of Computer Science Curriculum Recommendations CC2001

2.1 Overview CC2001

In the fall of 1998, the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) established the Joint Task Force on Computing Curricula 2001 (CC2001) to undertake a major review of curriculum guidelines for undergraduate programs in computing. CC2001 [13] represents the third public draft of the Computing Curricula 2001 project.

The main body of the computer science report consists of 13 chapters. Chapter 2 begins with a survey and analysis of past reports, focusing most closely on Computing Curricula 1991 (CC1991) [12]. Chapter 3 outlines the changes that have occurred in computer science since the publication of the CC1991 report and the implications that those changes have for curriculum design and pedagogy. Chapter 4 articulates a set of principles that have guided the development of CC2001 as it attempts to build on the strengths of its predecessors while avoiding some of the problems observed in the earlier reports. Chapter 5 and 6 present overviews of the computer science **Body of Knowledge** and the curriculum recommendations that are examined in detail in the appendices. Chapter 7 and 8 describe the courses and approaches it recommends at the introductory and intermediate levels of the curriculum, respectively. Because these courses alone do not constitute a complete undergraduate curriculum, Chapter 9 summarizes additional courses and topics that must be included as part of the academic program. Chapter 10 and 11 describe the study of professional practice. Chapter 12 looks at the problem of teaching computer science and computing-related skills to students in other disciplines. Finally, Chapter 13 offers a variety of strategic and tactical suggestions for implementing the recommendations in this report.

For our analysis only certain chapters have been considered as relevant. Chapter 3 points out technological and cultural changes in CS, these changes will be analyzed according to the reality of Arequipa and Peru. Chapter 5 presents the essential material to anyone obtaining an undergraduate degree in this field. Chapters 6, 7,8 and 9 give us strategies and hints for the organization of the topics in the introductory, intermediate and advanced levels of the curriculum, as well as the knowledge and abilities necessary to complete the curriculum in CS.

2.2 Changes in the Computer Science Discipline

The rapid evolution of Computer Science has a profound effect on computer science education, affecting both content and pedagogy. The CC2001 Task Force reviewed the CC1991 and developed a revised and enhanced version for the year 2001 that matched the latest developments of computing technologies. These developments fell into two categories–technological and cultural–each of which have a significant effect on computer science education.

Technological changes: Much of the change that affects computer science comes from advances in technology. Both evolutionary and revolutionary changes affect the body of knowledge required for computer science and the educational process. Technological advancement over the past decade has increased the importance of many curricular topics, such as the following: the World Wide Web (WWW) and its applications, Networking Technologies, particularly those based on TCP/IP, Graphics and Multimedia, Embedded Systems, Relational Databases, Interoperability, Object-Oriented Programming, the use of so-

phisticated Application Programmer Interfaces (APIs), Human-Computer Interaction, Software Safety, Security and Cryptography, Application Domains.

***Cultural changes*:** Computing education is also affected by changes in the cultural and sociological context in which it occurs. The following changes, have all had an influence on the nature of the educational process:

- Changes in pedagogy enabled by new technologies. The technological changes that have driven the recent expansion of computing have direct implications on the culture of education. Computer networks make distance education much more feasible, leading to enormous growth in this area.
- The dramatic growth of computing throughout the world. Computing has expanded enormously over the last decade.
- The growing economic influence of computing technology. The dramatic excitement surrounding high-tech industry, as evidenced by the Internet startup fever of the past five years, has significant effects on education and its available resources. The enormous demand for computing expertise and the vision of large fortunes to be made has attracted many more students to the field, including some who have little intrinsic interest in the material. At the same time, the demand from industry has made it harder for most institutions to attract and retain faculty, imposing significant limits on the capacity of those institutions to meet the demand.

Section 2.5 of this report analyzes in detail these cultural changes in CS discipline according to the reality of Arequipa and Peru.

2.3 CC2001-CS Body of Knowledge and Knowledge Units

The CC2001 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 Table 1. The CS Body of Knowledge is organized hierarchically into three levels. The highest level of the hierarchy is the **Area**, which represents a particular disciplinary subfield. Each area is identified by a two-letter abbreviation, such as OS for operating systems or PL for programming languages (as shown in Table 1). The areas are broken down into smaller divisions called **Units**, which represent individual thematic modules within an area (it can be seen in Appendix B) . Each unit is identified by adding a numeric suffix to the area name; as example, OS3: Concurrency. Each **unit** is further subdivided into a set of **Topics**, which are the lowest level of the hierarchy. The details of the **topics** appear in [25].

The CC2001 Task Force has defined a minimal **core** consisting of those **units** for which there is a broad consensus that the corresponding material is essential to anyone obtaining an undergraduate degree in this field. Units that are taught as part of an undergraduate program but which fall outside the core are considered to be **elective** (as shown in Appendix B).

The CC2001 recommendations have found that it helps to emphasize the following points:

- The core refers to those units required of all students in all computer science degree programs. Several topics that are important in the education of many students are not included in the core.
- The core is not a complete curriculum. Because the core is defined as minimal, it does not, by itself, constitute a complete undergraduate curriculum.

Table 1: Body of knowledge [25].

1.	(DS)	Discrete Structures
2.	(PF)	Programming Fundamentals
3.	(AL)	Algorithms and Complexity
4.	(PL)	Programming Languages
5.	(AR)	Architecture and Organization
6.	(OS)	Operating Systems
7.	(NC)	Net-Centric Computing
8.	(HC)	Human-Computer Interaction
9.	(GV)	Graphics and Visual Computing
10.	(IS)	Intelligent Systems
11.	(IM)	Information Management
12.	(SE)	Software Engineering
13.	(SP)	Social and Professional Issues
14.	(CN)	Computational Science and Numerical Methods

- The core must be supplemented by additional course-work. Every undergraduate program must include additional elective topics from the body of knowledge.

For time required to cover a particular unit, the CC2001 defines a metric that establishes a standard of measurement. For consistency with the earlier curriculum reports, the Task Force has chosen to express time in **hours**, corresponding to the in-class time required to present the material in a traditional lecture-oriented format (as shown in Appendix B).

2.4 Overview of the Curricular Models

The body of knowledge presented above does not by itself constitute a curriculum. To be useful, CC2001 report also defines detailed course implementations and strategies for assembling the individual courses into a complete undergraduate curriculum. CC2001 presents a brief description of the overall philosophy behind the curricular models.

The courses described in CC2001 are divided into three categories according to the level at which they occur in the curriculum. Courses designated as introductory are typically entry-level courses offered in the first or second year of a university curriculum. Courses listed as intermediate are usually second or third-year courses and build a foundation for further study in the field. Courses designated as advanced are taken in later years and focus on those topics that require significant preparation in terms of earlier course-work.

While these distinctions are easy to understand in their own right, it is important to recognize that there is no necessary relationship between the level of the course and the notions of core and elective, which apply only to units in the body of knowledge. Although introductory and intermediate courses will certainly concentrate on core material, it is perfectly reasonable to include some elective material even in the earliest courses. The idea behind this structure is to offer greater flexibility by making it possible to start with any of the introductory approaches an then follow up that introduction with any of the intermediate approaches.

A complete undergraduate curriculum consists of an introductory phase to establish basic foundations for further study, an intermediate phase to cover most of the **core-units** in the body of knowledge, and additional advanced courses to round out the curriculum. In doing so, it is important to ensure that the curriculum that results includes at least the minimum coverage specified in the core of the body of knowledge.

CC2001 has established a set of expectations for the introductory approaches in the form of a set of units and topics that each of those approaches must cover. The details of this coverage are outlined below.

2.4.1 Introductory Courses

Most introductory computer science courses have focused primarily on the development of programming skills. The adoption of a programming-first introduction arises from a number of practical and historical factors, including: programming is an essential skill that must be mastered by anyone studying computer science, placing it early in the curriculum ensures that students have the necessary facility with programming when they enroll in intermediate and advanced courses.

The programming-first approach, however, has several shortcomings: focusing on programming to the exclusion of other topics gives students a limited sense of the discipline; introductory programming courses often oversimplify the programming process to make it accessible to beginning students, giving too little weight to design, analysis, and testing relative to the conceptually simpler process of coding; among others. The CC2001 Task Force offers three implementations of a programming-first model and three that adopt an alternative paradigm. The programming-first implementations are:

Imperative-first approach that introduces the fundamental concepts of procedural programming topics including data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging. In addition, adopting the imperative-first strategy means that students will get less exposure to the techniques of object-oriented programming than they would if the curriculum followed the objects-first model. The programming languages are: C, Modula2, Fortran, Pascal among others.

Objects-first approach also focuses on programming, but emphasizes the principles of object-oriented programming and design from the very beginning. This approach begins immediately with the notions of objects and inheritance, giving students early exposure to these ideas. After experimenting with these ideas in the context of simple interactive programs, it goes on to introduce more traditional control structures, but always in the context of an overarching focus on object-oriented design. The programming languages are: C++, Java, Smalltalk among others.

Functional-first approach that introduces algorithmic concepts in a language with a simple functional syntax. The minimalist syntax of functional languages means that courses can focus on more fundamental issues. Several important ideas–most notably recursion, linked data structures, and functions as first-class data objects–occur naturally in this domain and can be covered much earlier in the curriculum. The programming languages are: Scheme, Haskell, Ocaml among others.

The three alternative models are a **breadth-first** approach that begins with a general overview of the discipline, an **algorithms-first** strategy that focuses on algorithms over syntax, and a **hardware-first** model that begins with circuits and then builds up through increasingly sophisticated layers in the abstract machine hierarchy. The CC2001 Task Force has chosen not to recommend any single approach. The truth is that no ideal strategy has yet been found, an that every approach has strengths and weaknesses.

Table 2 presents a set of concepts, knowledge, and skills that the CC2001 Task Force believe should be a part of each introductory curriculum. Table 3 and Table 4 express the same guidelines in terms of **units** and **topics** from the Body of Knowledge introduced before (See Appendix B).

Table 2: Concepts covered in the introductory curriculum.

Algorithmic thinking
Algorithmic computation
Algorithmic efficiency and resource usage
Programming fundamentals
Data models
Control structures
Order of execution
Encapsulation
Relationships among encapsulated components
Testing and debugging
Computing environments
Layers of abstraction
Programming languages and paradigms
Basic hardware and data representation
Tools

Table 3: Units for which all topics must be covered.

(DS1)	Functions, relations and sets
(DS2)	Basic logic
(DS4)	Basics of counting
(DS6)	Discrete probability
(PF1)	Fundamental programming constructs
(PF4)	Recursion
(PL1)	Overview of programming languages
(PL2)	Virtual machines
(PL4)	Declarations and types
(PL5)	Abstraction mechanisms
(SP1)	History of computing

Table 4: Units for which only a subset of the topics must be covered.

(DS3) Proof techniques
(PF2) Algorithms and problem solving
(PF3) Fundamental data structures
(AL1) Basic algorithmic analysis
(AL3) Fundamental computing algorithms
(PL6) Object-oriented programming
(AR1) Digital logic and digital systems
(SE1) Software design
(SE2) Using APIs
(SE3) Software tools and environments
(SE5) Software requirements and specifications
(SE6) Software validation

2.4.2 Intermediate Courses

The intermediate courses in the curriculum are designed to provide a solid foundation that serves as a base for more advanced study of particular topics. At the same time, it is important to keep in mind that the introductory courses and the intermediate courses together do not constitute a complete curriculum. All undergraduate programs will include a significant amount of additional elective material described in the next section. CC2001 proposes four implementations for the intermediate level of the curriculum: topic-based, compressed, systems-based, and web-based. These implementations are presented as representative models rather than as prescriptive standards. As with the introductory courses, individual faculty and institutions have crafted many different approaches to the intermediate level courses.

A **traditional topic-based** approach is 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. It is not necessary, however, to require separate courses in every area covered by the body of knowledge. Some areas with relatively few core-units can be integrated into the introductory curriculum; others, can be incorporated into advanced courses that explore the nature of professional practice in the discipline.

A **compressed** approach, combines the core-units from more than one area into a single course that adopts a broader, cross-cutting perspective. As a result, this approach may prove useful in environments where it is important to keep the number of courses down. It is important to note that even this implementation devotes additional time to the courses beyond the theoretical minimum established by the size of the core.

A **systems-based** approach 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.

A **web-based** approach 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.

In each of these models, the intent of the intermediate level of the CC2001 curriculum remains unchanged: to present the fundamental ideas and enduring concepts of computer science that every student must learn to work successfully in the field.

2.4.3 Completing the Curriculum

The main purpose of Sections 2.4.1 and 2.4.2 is to outline a variety of approaches for covering the core-units in the body of knowledge. As CC2001 emphasizes on several occasions, the computer science core does not in itself constitute a complete curriculum. To complete the curriculum, universities must also ensure that students have the background knowledge and skills they need to succeed as well as the chance to do advanced work that goes beyond the boundaries of the core. CC2001 describes a set of general requirements and offer some suggestions on advanced courses.

General Requirements: CC2001 describes a set of general requirements that support the broad education of computer science students. General requirements for a successful computer science graduate needs many skills beyond the technical ones found in the CS body of knowledge. Computer science students must have a certain level of mathematical sophistication, familiarity with the methods of science, a sense of how computing is applied in practice, effective communication skills, and the ability to work productively in teams.

- *Mathematical sophistication:* Mathematics techniques and formal mathematical reasoning are integral to most areas of computer science. Mathematics provides a language for working with ideas relevant to computer science, specific tools for analysis and verification, and a theoretical framework for understanding important computing ideas. Given the pervasive role of mathematics within computer science, the CS curriculum must include mathematical concepts early and often. Basic mathematical concepts should be introduced early within a student's course work, and later courses should use these concepts regularly. Computer science programs must take responsibility for ensuring that students get the mathematics they need, especially in terms of discrete mathematics. The material on discrete structures can be presented in separate courses or integrated more directly into the curriculum by presenting the mathematical material together with the computer science topics that depend on it. It is essential to make sure that the curriculum emphasizes the use of discrete mathematical techniques throughout the undergraduate curriculum.
- *Communication skills:* Computer science curriculum should require course work that emphasizes the mechanics and process of writing, formal oral presentation to a group, the opportunity to critique at least one oral presentation. Communication skills should not be seen as separate but should instead be fully incorporated into the computer science curriculum and its requirements.
- *Work in teams:* To ensure that students have the opportunity to acquire the working in teams skill they should work in a significant project that involves a complex implementation task in which both the design and implementation are undertaken by a small student team. This project is often scheduled for the last year of undergraduate study, where it can serve as a capstone for the undergraduate experience.

Advanced Courses: CC2001 uses the term advanced course to mean courses whose content is substantially beyond the material of the core. The units in the body of knowledge give testimony to the rich set of possibilities that exist for such courses, but few if any institutions will be able to offer courses

covering every unit in detail. Institutions will wish to orient such courses to their own areas of expertise, guided by the needs of students, the expertise of faculty members, and the needs of the wider community. The titles for these courses appear in Table 5 organized by knowledge area.

Table 5: Advanced courses by area.

(AL) Algorithms and Complexity
Automata Theory
Advanced Algorithmic Analysis
Coding and Information Theory
Security and Cryptography
Parallel Algorithms
(PL) Programming Languages
Programming Language Translation
Programming Language Design
Programming Language Semantics
Programming Paradigms
Programming Languages and Syntax-Directed Tools
(AR) Architecture and Organization
Advanced Computer Architecture
Parallel Architectures
(OS) Operating Systems
Advanced Operating Systems
Concurrent Systems
Real-time Systems
Concurrency and Distributed Systems
(NC) Net-Centric Computing
Distributed Systems
Mobile Computing
Cluster Computing
(HC) Human-Computer Interaction
Human-Centered Design and Evaluation
Computer-Supported Cooperative Work
Graphical User Interfaces
(GV) Graphics and Visual Computing
Advanced Computer Graphics
Image Processing
Computer Vision
(IS) Intelligent Systems
Intelligent Systems
Machine Learning
Agents
Robotics

(IM) Information Management
Database Design
Distributed Databases
Data Compression
(SE) Software Engineering
Professional Practice
Software Engineering
Advanced Software Design
Event-driven Programming
Component-based Computing
Formal Specification
Software Engineering and Formal Specification
(CN) Computational Science and Numerical Methods
Scientific Computing
Numerical Analysis
Operations Research
Modeling and Simulation

2.5 Discussion

Currently, CC2001 can be considered to be a set of recommendations from an international perspective reflecting the current state of the art in Computer Science and requirements for CS education in the rest of the current decade. It can readily serve as a reference model for curriculum improvement at San Pablo Catholic University.

2.5.1 Cultural Changes in CS and its impact in Peru

To get a better understanding of the current situation in Peru and Arequipa it is necessary to look into some relevant features:

- Peru is largely considered to be a developing country. Probably the most difficult subject is that Peru lacks hight-tech industry. That is why there is not a large computing expertise demand. Currently, the occupation field is more related to the Information Systems academic field.
- It does not exist a notable development in the industry of software development either. The type of software development is oriented to Small Medium Enterprises (SME) and this type of software development has not achieved importants levels of the Capability Maturity Model for Software (CMM). Currently the culture that demands and produces software of highest quality does not exist in Peru.
- Peru networking is applied as LAN and WAN in enterprises and institutions. The WWW is widely used as a means of communication, search and publishing of information. Peru has not developed electronic business yet. We can not say that networking and WWW in Peru have become a support for the Peruvian economy.
- For the purposes of this discussion we want to point out that Arequipa got access to Home Internet Service in 2001 and its cost right now is US$ 40.00 per month. It is a high price for the income level

in Arequipa. For example, the avarage monthly income of a university lecturer is about US$ 300 in a private university, like UCSP, and US$ 200 in public universities. Only approximately 10% of the population of Peru has a computer at home. This ratio can be assume to be apply to Arequipa since statistics only exists on a national level [16]. For this low ratio we can see that is very difficult these homes can get access to the Internet Services.

With the background outlined above, it should be obvious this makes it difficult for Peru to take advantage of the cultural changes in Computer Science discipline we pointed out in Section 2.2.

2.5.2 The Body of Knowledge and Implementation Strategies at UCSP

The Body of Knowledge Section 2.3 showed that the body of knowledge and the units present invaluable tools for curriculum analysis. CC2001 has defined a minimal core consisting of those units for which there is a broad consensus that the corresponding material is essential to anyone obtaining an undergraduate degree in Computer Science. We should analyze in what percentage the UCSP curriculum meets this minimum core suggested.

Introductory Level Section 2.4.1 described six implementations of the introductory curriculum that the CC2001 Task Force feels have achieved a level of success that allows them to serve as models of best practices. From this analysis we have identified that the UCSP curriculum follows the **imperative-first**[1] approach. The first years courses introduce UCSP students to the fundamental concepts of procedural programming, topics of algorithms and problem-solving, fundamental data structures, overview of operating systems, introduction to human-computer interaction and social context of computing are developed. The programming language used so far in this first stage is C, now this is changing to Java and C++. These courses have been designed according to CC2001.

In addition, Section 2.4.1 pointed out some units and topics that CC2001 believe should be a part of each introductory curriculum. We think it is necessary to make a comparative analysis between these units and topics recommended by CC2001 and the topics taught at UCSP during the first years.

Intermediate Level Section 2.4.2 presented four implementations for the intermediate level of the curriculum proposed by CC2001. These implementations were presented as representative models rather than as prescriptive standards. As with introductory level, we have identified that the UCSP curriculum follows the **traditional topic-based** approach.

UCSP currently teaches courses related to almost all of the areas presented in Section 2.3 (see Table 1). We can mention: *Discrete Structures*, *Basic Computer* (Programming Fundamentals in Table 1), *Programming Languages*, *Analysis and Design Algorithms* (Algorithms and Complexity in Table 1), *Software Engineering*, among others. Although we can note the coincidence with the set of areas presented by CC2001, it is necessary to analyze to what extend the courses taught at UCSP meet the minimum core suggested in each area.

[1] Introductory courses at UCSP curriculum are undergoing change switch to object-oriented approach. This is following some initial recommendation suggested in a early version of this report

Advanced Level and General Requirements To complete the curriculum, UCSP must also ensure that students go beyond the boundaries of the core. Section 2.4.3 showed the advanced courses. This point will enable us to analyze which is the orientation of the advanced courses taught at UCSP. Furthermore, Section 2.4.3 presented a set of general requirements that support the broad education of CS students. CC2001 has recommended certain degree of mathematical sophistication. The current UCSP curriculum meets this requirement with the following courses: *Discrete Mathematics I*, *Discrete Mathematics II*, *Single calculus*, *Multi-variable calculus*, *Basic Mathematics* (Basic Logic), among others. The mathematical concepts are studied in the first semesters. CC2001 also recommends effective communication skills. UCSP meets this requirement with the *Spanish* course, where students develop writing and oral skills. This course is taught in the first semester. These abilities are reinforced in the following advanced courses: *Project I*, *Project II*, *Project III* and *Enterprise Systems*.

In conclusion, taking CC2001 as reference, we will be able to have a global vision of the UCSP curriculum according to the international standards. The details of the analysis are outlined in Section 4.

3 Review of Model Curriculum and Guidelines for Information Systems IS2002

3.1 Overview IS2002

The IS2002 report [10] is the latest output from model curriculum work for Information Systems (IS) that began in the early 1970s and has matured over a thirty year period. The IS2002 model curriculum is the first update of the curriculum effort of the Association for Computing Machinery (ACM), Association for Information System (AIS) and Association of Information Technology Professionals (AITP) societies since the Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems 1997 (IS97) [9].

IS2002 report is part of the Computing Curricula 2001 project [26]. IS2002 provides the background material and a detailed set of course descriptions and advice to the intended users of the report who have a stake in the achievement of quality IS degree programs. Information Systems, as an academic field, encompasses two broad areas: (1) acquisition, deployment, and management of information technology resources and services (the information systems function); and (2) development and evolution of technology infrastructures and systems for use in organizational processes (systems development).

IS2002 is a model curriculum for undergraduate degree programs in Information Systems. It includes a mapping from the course structure IS97 to the new set of courses IS2002. The model curriculum is based on common structures and degree programs in the United States and Canada. The curriculum represents a reasonable consensus of the IS community. The curriculum reflects input from both industry and universities. The exit characteristics of information systems graduates are defined in the report. The curriculum has formal information systems courses but also assumes use of prerequisite or co-requisite courses in communications, mathematics, and statistics, and business functions.

The main body of IS2002 report consists of 13 sections. Section 1, presents the use of the IS2002 Curriculum Report. Section 2, summarizes the principles guiding the curriculum design. Section 3 outlines the motivation for the curriculum update of IS97. Section 4 presents the guiding assumptions about the IS profession. Section 5 focusing on scope of the curriculum update. Section 6 describes the IS as a field of academic study. Section 7 summarizes the relationship between the information systems core courses, the minor , and the major. Section 8 looks at pre and co-requisites to an Information Systems degree program. Section 9 outlines the exit characteristic of IS graduates. Section 10, 11 and 12 describes the architecture of the information system curriculum, the resources for IS degree programs and the shared courses with other computing disciplines. Finally section 13 offers the IS2002 course specifications.

For our analysis only certain sections have been considered as relevant. Section 4, presents us the conceptualization, characteristics and the role of IS profession. Section 8, presents the architecture of IS curriculum through 5 areas of study and the courses related to them. Section 10 presents the additional knowledge and abilities to complete the curriculum in this field, and finally, Section 13 details the objectives and topics of each of the courses presented in Section 8.

3.2 IS2002 Guiding Assumptions About the Information Systems Profession

In conceptualizing the role of information systems in the requirements for IS curriculum, several elements remain important and characteristic of the discipline. These characteristics evolve around four major areas of the IS profession and therefore must be integrated into any IS curriculum:

- IS professionals must have a broad business and real world perspective: IS are enablers of successful performance in organizations, IS span and integrate all organizational levels and business functions, IS are increasingly of strategic significance because of the scope of the organizational systems involved and the role systems play in enabling organizational strategy.

- IS professionals must have strong analytical and critical thinking skills: be problem solvers and critical thinkers, use systems concepts for understanding and framing problems, be capable of applying both traditional and new concepts and skills, understand that a system consists of people, procedures, hardware, software, and data.

- IS professionals must exhibit strong ethical principles and have good interpersonal communication and team skills: IS require the application of professional codes of conduct, IS require collaboration as well as successful individual effort, IS design and management demand excellent communication skills.
- IS professionals must design and implement information technology solutions that enhance organizational performance.

IS professionals work with information technology and must have sound technical knowledge of computers, communications, and software. Since they operate within organizations and with organizational systems, they must also understand organizations and the functions within organizations (accounting, finance, marketing, operations, human resources, and so forth). They must understand concepts and processes for achieving organizational goals with informations technology. The academic content of an information system degree program therefore includes information technology, information systems management, information systems development and implementation, organizational functions, and concepts and processes of organizational management.

3.3 Architecture of the Information Systems Curriculum

The IS2002 curriculum is organized at the highest level as a set of curriculum presentation **Areas**. Each of these areas has one or more **Courses**. Each course should be built from **Learning Units**. A course is a group of learning units. Each learning unit is stated in terms of a goal, a set of objectives,and elements of the **Body of Knowledge**.

3.3.1 Curriculum Presentation Areas

The IS2002 identified a set of 5 **Areas** that together represented the highest level of curriculum. A description of the content for the areas is presented in Table 6.

Table 6: Content for five IS curriculum presentation areas.

1. Information Systems Fundamentals
It includes a broad introduction to the field of Information Systems and information technology plus instruction designed to improve personal productivity in an organization through effective and efficient use of information technology.
2. Information Systems Theory and Practice
Students will be introduced to concepts and theories that explain or motivate methods and practices in the development and use of information systems in organizations. The concepts and theories will include systems, management, and organization, information, quality, and decision making. The relationship of information systems to corporate planning and strategy and concepts relating information technology to comparative advantage and productivity are explained. The concepts and practices underlying the use of information technology and systems in improving organizational performance are presented.
3. Information Technology
Students will gain breadth and depth in the technical aspects of the discipline. Computing system architectures, operating systems software, and interconnection of information resources through networking are major components of presentation and discussion. Students will be expected to develop significant skills by participating in installation, configuration, and operation of the technologies.
4. Information Systems Development
Students will work in teams to analyze problems and design and implement information systems. Systems analysis provides experience determining system requirements and developing a logical design. Instruction in physical design of information systems will ensure that the students can use a logical design to implement information systems in both a Data Base Management Systems (DBMS) and in emerging development environments. Students should be exposed to a variety of development approaches.
5. Information Systems Deployment and Management
Students engage in numerous projects. Management of the information systems function, systems integration, and project management to ensure project quality are integral components of this curriculum area.

3.3.2 Curriculum Presentation Courses

Courses in IS2002 are the building blocks that implement the broad curriculum presentation areas shown in Table 6. The courses are labeled IS2002_P0 through IS2002_10. IS2002_P0 is considered to be a prerequisite to the program. The IS2002 curriculum assumes that students have a prerequisite knowledge of desktop computing with an elementary exposure to a suite of software applications useful for knowledge workers such as word processing, spreadsheets, e-mail, and Internet browsing.

Table 7 shows the course architecture. The structure is a suggested architecture with the appreciation

that each university's situation is somewhat unique. Table 8 shows the courses and their related areas. The set of courses represents a complete model that includes all of the learning units. Courses are described in Appendix E.

Table 7: IS curriculum courses.

IS2002_P0	Personal Productivity with IS Technology
IS2002_1	Fundamentals of Information Systems
IS2002_2	Electronic Business Strategy, Architecture and Design
IS2002_3	Information Systems Theory and Practice
IS2002_4	Information Technology Hardware and Software
IS2002_5	Programming, Data, File and Object Structures
IS2002_6	Networks and Telecommunication
IS2002_7	Analysis and Logical Design
IS2002_8	Physical Design and Implementation with a DBMS
IS2002_9	Physical Design and Implementation in Emerging Environments
IS2002_10	Project Management and Practice

Table 8: IS curriculum courses and related areas.

Prerequisite
IS2002_P0 Personal Productivity with IS Technology
1. Information System Fundamentals
IS2002_1 Fundamentals of Information Systems IS2002_2Electronic Business Strategy, Architecture and Design
2. Information System Theory and Practice
IS2002_3 Information Systems Theory and Practice
3. Information Technology
IS2002_4 Information Technology Hardware and Software IS2002_5 Programming, Data, File and Object Structures IS2002_6 Networks and Telecommunication
4. Information System Development
IS2002_7Analysis and Logical Design IS2002_8Physical Design and Implementation with a DBMS IS2002_9 Physical Design and Implementation in Emerging Environments
5. Information Systems Deployment and Management
IS2002_10 Project Management and Practice

3.3.3 Learning Units

A learning unit describes a set of material to be learned by students. A course is a group of learning units. Each learning unit is stated in terms of a goal, a set of objectives, and elements of the IS body of knowledge along with competency or depth of knowledge levels. The material to be covered by a learning unit is expressed in a presentation goal. The learning unit is designed to combine elements from the IS body of knowledge. Each learning unit is specified by a goal statement that explains the purpose of the learning unit. Each learning unit has a set of topics that define the coverage for the unit. These topics consist of elements from the IS body of knowledge. The learning units provide the basis for detailed

course design. A topic may be covered at a low depth of knowledge level as part of an introductory course and in more depth (higher competency) in a subsequent course.

3.3.4 Body of Knowledge

The IS body of knowledge consists of the topics to be taught in an IS curriculum. The IS2002 body of knowledge is a reorganization and extension of the IS97 body of knowledge. The body of knowledge was derived from surveys of practitioners and academics and mapping of relevant topics from curricula for Computer Science and other computer related disciplines. The topics in the IS body of knowledge form the lowest level building blocks for the curriculum. The elements are grouped under learning units and learning units are grouped into courses. The body of knowledge units are described at a detailed level at [11].

3.4 Pre and Co-requisites to an Information Systems Degree Program

Section 3.3 had presented formal IS courses. In addition, it is necessary to include courses in communications, mathematics, statistics, and business functions since students are expected to go beyond the basic knowledge presented above.

Students are expected, as a prerequisite, to have a basic proficiency in the fundamental tools of personal computing such as e-mail, web browsing, spreadsheets, word processing, desktop database management systems, presentations graphics, and externals database retrieval tools.

Co-requisites courses include communication (general and technical writing, oral communications, presentations, and listening skills), quantitative an qualitative analysis (topics as discrete mathematics, introduction to calculus, introductory statistics, and archival document analysis), functional areas of organizations (accounting, finance, human resources, marketing, logistics, and operations), they should also be introduced to special issues in international business.

3.5 Discussion

The IS2002 report provides the rationale for adopting the curriculum recommendations for an undergraduate program in Information Systems. IS encompasses two broad areas: information system functions and system development. IS2002 provides the background material, and a detailed set of course descriptions and advice for the universities who have a stake in the achievement of quality IS degree programs.

The model curriculum is based on common structures and degree programs in the USA and Canada. The model, however, is grounded in a fundamental body of computing an information system knowledge. It can therefore, employed as a reference model for international use.

3.5.1 Guideline Assumptions

Section 3.2 showed the requirements and characteristics that have to be present in the Information System curriculum. We pointed out that IS professionals must have specific characteristics: a broad business and

a real world perspective, strong analytical and critical thinking skills, interpersonal communications and team skills, strong ethical principles, and, to implement information technology solutions that enhance organizational performance. We consider it necessary to analyze if these characteristics and requirements are integrated in the current UCSP curriculum.

The first one, related to the performance within the organizations is include in the UCSP curriculum in the following courses: *Processes Analysis* course (regarding to principal functional areas and business processes that operate in the organizations), *Information Systems* course(it gives students a conceptual frame to understand the role of the information systems in the organizations and how IS can improve the enterprise performance), *Strategic Planning of Information Systems* course (regarding to the strategic direction of the enterprise, as well as the role of the information technology in the planning of such process)

The second one, related to analytical and critical thinking skills is include in the following courses: *Information Systems* course (it gives students a vision about how hardware, software, people, processes and organization are linked), and the *System Analysis* course (it presents the concepts to understand and specify the requirement to solve problems related to the enterprise).

The third one, related to interpersonal communication and team skills, is present in the following courses: *Spanish* course, *Project I, Project II, Project III)*, and *Enterprise Systems* course. *Enterprise Systems* course gives students their first real work experience. UCSP students execute a project from its conception to the implementation with a real client.

The fourth requirement, related to ethical principles, is presented in the course of *Ethic* at UCSP curriculum. It is important to UCSP to ensure that this course includes social, legal, and ethical issues inherent in the discipline of computing.

We have observed that in regards to the last requirement, to implement information technology solutions the *Information Systems* course and the *Strategic Planning of Information Systems* course give students an integrating vision of the impact of technology in the organization.

3.5.2 Body of Knowledge and UCSP

The IS2002 curriculum model is organized at the highest level as a set of curriculum presentation areas as we showed in Section 3.3 (Table 6). Each of these areas has one or more courses presented in Table 8. Each course is described with course title, catalog, scope, topics and discussion. We will use as an analysis tool the total description of the courses presented in Appendix E. We consider important to analyse in what percentage the UCSP curriculum meets the contents of these courses. The analysis is shown in Section 4.

3.5.3 UCSP and Pre and Co-Requisites

Finally Section 3.4 showed the additional requirements to an information systems degree program. Right now, UCSP students who begin studies in Informatics Engineering have previous knowledge in the use of e-mail, web-browsing, and the word processing. The management of desktop databases managements systems and external database retrieval tools is learned in courses related to Databases in the sixth and seventh semesters.

The requirements considered as co-requisites are present in UCSP curriculum in the following courses: *Spanish*, which develops writing (no technical writing) and oral communications abilities. *Discrete Mathematics I*, *Discrete Mathematics II*, *Single calculus*, *Multi-variable calculus*, *Statics*, among others where students develop quantitative and qualitative analysis skills.*Processes Analysis*, *Administrative Theory*, and, *Economics* where students receive instruction in topics related to organization and economics (the courses currently taught at UCSP are shown in Appendix A).

In conclusion, taking IS2002 as reference, we will be able to have a global vision of the UCSP curriculum according to Information Systems as an academic field and the achievement of quality IS degree programs. The details of this analysis are outlined in Section 4.

4 Detailed Analysis of the UCSP Curriculum

The main purpose of Sections 2 and 3 was to review CC2001 and IS2002 curricular recommendations in computer science and information systems. In this section taking CC2001 and IS2002 as reference models, we analyze the UCSP curriculum according to these international standards. In addition to the standards, we use specific information about UCSP courses obtained from discussions with UCSP lecturers. We consider it relevant to concentrate on: the Body of Knowledge CC2001, Introductory Courses CC2001, Advanced Courses CC20001, and IS2002 Courses. At the end of this section we will get a clear picture where UCSP curriculum stands. The details of these analysis are shown in Appendix G (Figures 1, 2, 3, 4 and 5).

4.1 Body of Knowledge CC2001

Section 2.3 pointed out a minimal core consisting of those **units** for which there is a broad consensus that the corresponding material is essential to anyone obtaining an undergraduate degree in Computer Science. Figure 1 shows the analysis of the UCSP curriculum. As we can observe it shows in the main row the courses taught in the UCSP curriculum and the semester in which they are taught, and, the main column shows the units recommended by CC2001.

The analysis was made considering the set of topics recommended for each unit in CC2001 (the details of the topics appear in [25]) and the topics taught currently in each of the courses of the UCSP curriculum. In this analysis we have considered only the **core-hours** that make a total of 280 hours. Each combination row-column shows the number of core-hours dedicated to each unit in the courses taught at UCSP.

Figure 1 shows that there are units that are not covered completely in number of core-hours, such as **DS6: Discrete probability** or **SE1: Software design**. This is because, according to the analysis made, only a subset of topics from those recommended by CC2001 is taught now.

In other cases, there are units in which the topics are not covered at all, such as **HC1: Foundations of human-computer interaction** and **HC2: Building a simple graphical user interface**.

Summarizing, we can observe that UCSP curriculum only covers 220 out of the 280 minimum core-hours suggested by CC2001, which makes only a 78.57% of the total suggested. There are certain **units** that need to be included at UCSP curriculum to complete the minimal core recommended by CC2001. They are listed below: (The details of these units are shown in Appendix H).

- Area Discrete Structures (DS)
  - DS6: Discrete probability
- Area Programming Fundamentals (PF)
  - PF5: Event-driven programming
- Area Programming Languages (PL)
  - PL2: Virtual Machines
- Area Algorithms and Complexity (AL)

  - AL4: Distributed Algorithms
- Area Net-Centring Computing (NC)
  - NC4: The web as an example of client-server computing
  - NC3: Network security
- Area Human-Computer Interaction (HC)
  - HC1: Foundations of human-computer interaction
  - HC2: Building a simple graphical user interface
- Area Software Engineering (SE)
  - SE2: Using APIs
  - SE3: Software tools and environments
- Area Social and Professional Issues Units (SP)
  - SP1: History of computing
  - SP2: Social context of computing
  - SP3: Methods and tools of analysis
  - SP4: Professional and ethical responsibilities
  - SP5: Risk and liabilities of computer-based systems
  - SP6: Intellectual property
  - SP7: Privacy and civil liberties

4.2 Introductory Courses CC2001

Section 2.4.1 showed the introductory phase of the undergraduate curriculum in computer science. CC2001 presented a set of concepts, knowledge, and skills in terms of units and topics from the body of knowledge.

Figure 5 shows the analysis of the introductory courses taught in the UCSP curriculum. The analysis was made evaluating the courses taught at UCSP during the first two years (first to fourth semesters) and the topics of the body of knowledge (Tables 3 and 4) that CC2001 Task Force believe should be a part of each introductory course of the curriculum.

As we can see in Figure 5, we show in the main row the courses taught in the Curriculum at UCSP and the semester in which they are taught. In the main column we show the units for which **all topics** must be covered.

Each combination row-column shows the number of core-hours dedicated to each unit in the introductory courses taught at UCSP. We can observe in Figure 5 that:

- **PL2: Virtual Machines** is not covered at all.
- **DS6: Discrete probability** is not covered completely in number of core-hours.

- **PF4: Recursion**, has not been considered as an introductory unit. This is because, according to the analysis made, this unit is taught in the fifth semester at UCSP curriculum.

Figure 5 points out that UCSP curriculum only covers 42 out of the 50 minimum core-hours of introductory units for which **all topics** must be covered suggested by CC2001, this makes only an 84% of the total suggested.

Figure 6 shows in the main row the courses taught in UCSP curriculum and the semester in which they are taught, and, in the main column, it shows the units for which **a subset** of the topics must be covered and the subset core-hours. We can observe in Figure 6 that there is one unit that is not covered completely in number of subset core-hours, **PF3: Fundamental Data Structures**. We can also see that many of the units have not been considered as introductory. This is because, according to the analysis made, they are taught in the later semesters (fifth to eight). That is the case of:

- AL1: Basic algorithmic analysis
- AL3: Fundamental computing algorithms
- AR1: Digital logic and digital systems
- SE1: Software Design
- SE5: Software requirements and specifications
- SE6: Software validation.

In other cases, there are units in which the topics are not covered at all:

- SE2: Using API's
- SE3: Software tools and environments.

Figure 6 points out that UCSP curriculum only covers 23 out of the 47 minimum subset core-hours units for which a subset of the topics must be covered, which makes only a 48.94% of the total subset core-hours suggested.

As a result of this analysis, we can see that the UCSP curriculum is giving too little core-hours to design, analysis, and testing relative to the conceptually simpler process of coding in the introductory courses. It is the case of: *SE1: Software Design*, *SE5: Software requirements and specifications* and *SE6: Software validation*, which are units taught in the later semesters at UCSP curriculum. The same concern is present at *AL1: Basic algorithmic analysis*, and, *AL3: Fundamental computing algorithms*, units which correspond to **Algorithms and Complexity** area and are not taught but in the fifth semester. (The details of all the units listed in this section are shown in Appendix H).

4.3 Advanced Courses CC2001

Section 2.4.3 introduced the advanced courses to complete the curriculum. The units (elective-units) and topics in the body of knowledge gave testimony to the rich set of possibilities that exist for such

courses. The CC2001 has produced a set of advanced courses using the framework provided by the body of knowledge. The titles for these courses organized by area were shown in Table 5.

Figure 8 shows the analysis of the advanced courses at UCSP curriculum. We analyze the full descriptions of the advanced courses to see which ones are currently taught at UCSP. As we can see in Figure 8, each combination row-column shows the number of hours dedicated to each topic considered as advanced. As result of the analysis we can appreciate in Table 9 the advanced courses currently taught at UCSP:

Table 9: Advanced courses at UCSP.

CC2001 Advanced Course	**UCSP Advanced Courses**
(AL) Automata Theory	Computer Theory
(PL) Programming Language Translation	Computer Theory
(GV) Advanced Computer Graphics	Computer Graphics I
(GV) Image Processing	Computer Graphics II
(IS) Intelligent Systems	Artificial Intelligence, Expert Systems
(IS) Machine Learning	Neural Networks
(IS) Agents	Artificial Intelligence, Expert Systems
(IM) Database Design	Database I
(IM) Distributed Databases	Database II
(CN) Numerical Analysis	Numerical Methods
(CN) Operations Research	Mathematic Programming, Stochastic Processes
(CN) Modeling and Simulation	System Simulation

We can observe that advanced courses at UCSP curriculum are related to the following areas:

- (AL)Algorithms and Complexity
- (PL)Programming Languages
- (GV)Graphics and Visual Computing
- (IS)Intelligent Systems
- (IM)Information Management
- (CN)Computer Science and Numerical Methods

It is important to UCSP ask itself if the advanced courses taught in the UCSP curriculum shown in Table 9, are designed according to the specialization degree of the academic staff and, in addition, to analyze if these advanced courses are related to the topics taught in the introductory and intermediate courses at UCSP curriculum.

Since the UCSP is interested in developing a Software Developing Industry according to international standards, we think it is necessary to consider the advanced courses related to **Software Engineering** area, such as : *Advanced Software Engineering* course, *Formal Specification and Software Engineering* course or *Formal Specification* course among others presented in Section 2.4.3.

Furthermore, UCSP curriculum teaches the advanced courses of the area of **Computational Science and Numerical Methods** (see Table 5 in Section 2.4.3). These courses are present in the UCSP curriculum under the names of: *Numerical Methods*, *Mathematics Programming*, *Stochastic Process* and *System Simulation*). Note that even though these courses are considered as advanced in CC2001, the current UCSP curriculum teaches theses courses as part of the core in the introductory and intermediate levels, this is between the fourth and eighth semesters.

4.4 IS2002 Courses

Section 3.3 presented the courses which are the building blocks that implement the curriculum for Information Systems. Appendix E details the set of topics to be covered in each course.

Unlike the analysis made in CC2001, the IS2002 does not suggest a minimum core-hours to be covered in each topic, in order to make our analysis, we have assigned weights to determine the percentage of teaching in each one of the topics suggested by IS2002. Table 10 shows that relation.

Table 10: Weights and percentage taught IS2002.

Weight	Percentage taught
3	100% - 70%
2	69% - 40%
1	39% - 1%
0	is not studied

The analysis was made considering the set of topics included in each course recommended in IS2002 and the topics taught currently in each of the courses of the curriculum at UCSP.

Figure 13 shows the analysis of the Curriculum at UCSP and the IS2002. As we can see in Figure 13, we present in the main row the courses taught in UCSP curriculum and the semester in which they are taught. In the main column we present the courses and their related topics recommended by IS2002. We have assigned the highest weight (3) to each topic recommended by IS2002. It makes a total of 342 points. Each combination row-column shows the weight assigned to each topic in the courses taught at UCSP.

As we can see, the UCSP curriculum currently covers only 216 out of the 342 (69.16 %). There are topics that are not covered completely such as *IS1.14: Crime and ethics* or *IS2.1: Electronic commerce economics*. In other cases, there are topics which are not covered at all, such as *IS1.12: Characteristics of IS professionals and IS career path* or, *IS2.18: Payment systems*.

We can observe in Figure 13 that there are three courses that make the percentage be low. The first one is **IS2002_2: Electronic Business Strategy, Architecture and Design**, in which UCSP curriculum only cover 16 out of the 54 points. The second one is **IS2002_6: Networks and Telecommunications**, 6 out of 30 points, and the third one is **IS2002_7: Analysis and Logical Design**, 19 out of 36 points. For our analysis it is important to deepen the explanations about these three courses:

Electronic Business Strategy, Architecture and Design: When IS97 was analyzed, the missing element in the curriculum was found to be a course on Internet-based commerce: *Electronic Business*

Strategy, Architecture and Design. Being this course recently added in the IS2002, we can see in Figure 13 that most of the topics of this course are not taught in the current UCSP curriculum. Right now, only some topics are taught which give a global vision and the importance of electronic business to the enterprise. This topics are taught in the course of *Information Systems* in the UCSP curriculum.

Networks and Telecommunications: Even though we can see that there is little coverage of the suggested topics, the current orientation of UCSP curriculum is more related to CC2001 (Net-Centric Computing area (NC)), this is computer and telecommunications networking, particularly those based on laboratory experiments involving data collection and synthesis, empirical modeling, protocol analysis at the source code level, network packet monitoring among others. The IS2002 orientation is related to the design and implementation work, specifically: installation, configuration and operation of bridges, routers, switches, and gateways; installations and configuration of networks, monitoring of networks among others. Differences between approaches of CC2001 and IS2002 explain why the UCSP curriculum only covers 6 out of the 30 points.

Analysis and Logical Design: Most of the topics not taught are referred to the orientation of the structured analysis, this is: structured methodologies, joint application development ,and structured walkthroughs. The orientation of the *Analysis* course and the *Design* course in the current UCSP curriculum are closer to the CC2001 recommendations (object-oriented analysis and design). Differences between CC2001 and IS2002 approaches explain the low percentage in this course.

Summarizing, UCSP curriculum only cover 69.16 % of the total suggested by IS2002. It is necessary to include and complete certain topics in order to cover the minimal core suggested by IS2002. This topics are listed below:

- IS 2002_1 Fundamentals of Information Systems
  - IS1.3 Cost/Value and quality of information
  - IS1.12 Characteristics of IS professionals and IS career paths
  - IS1.13 Information security
  - IS1.14 Crime, and ethics
  - IS1.15 Practical exercises developing macros.
- IS2002_2 Electronic Business Strategy, Architecture and Design
  - IS2.1 Electronic commerce economics
  - IS2.2 Business model
  - IS2.3 Technology architectures for electronic business
  - IS2.5 Supply chain management
  - IS2.6 Consumer Behavior within electronic environments
  - IS2.7 Legal and ethical issues
  - IS2.8 Information privacy and security
  - IS2.9 Transborder data flow
  - IS2.10 Information accuracy and error handling

  - IS2.11 Disaster planning and recovery
  - IS2.12 Solution planning
  - IS2.13 Implementation and Roll-out
  - IS2.15 Internet standard and methods
  - IS2.17 EDI
  - IS2.18 Payment systems
  - IS2.19 Support for inbound and outbound logistics

- IS2002_3 Information Systems Theory and Practice
  - IS3.11 Human-computer interface

- IS2002_6 Networks and Telecommunications
  - IS6.3 Distributed systems
  - IS6.4 Wired and wireless architectures, topologies, and protocols
  - IS6.5 Installation, configuration, and operation of bridges, routers, switches, and gateways.
  - IS6.7 Privacy, security, firewall, reliability
  - IS6.8 Installation and configuration of networks
  - IS6.9 Monitoring and management of networks
  - IS6.10 Communications standard

- IS2002_7 Analysis and Logical Design
  - IS7.2 Interpersonal skills, interviewing, presentations skills
  - IS7.3 Groups dynamics
  - IS7.6 Join applications development (JAD) and structured walkthroughs
  - IS7.7 Structured versus object oriented methodologies
  - IS7.8 RAD and prototyping
  - IS7.10 Software package evaluation, acquisition and integration
  - IS7.11 Global an inter-organizational issues and system integration
  - IS7.12 Professional code of ethics

- IS2002_8 Physical Design and Implementation with DBMS
  - IS8.3 Design tools
  - IS8.5 Database implementation including user interface and reports
  - Multi-tier planning and implementation
  - Data conversion and post implementation review

- IS2002_9 Physical and Implementation in Emerging Environments
  - IS9.2 Structured, even driven, an object oriented application design
  - IS9.5System implementation
  - IS9.6 User training
  - IS9.7 System delivery

  - IS9.8 Post implementation review
  - IS9.9 Configuration management
  - IS9.10 Maintenance
  - IS9.11 Multi-tier architectures and client independent design
- IS2002_10 Project Management and Practice
  - IS10.3 Network management
  - IS10.8 Reporting and presentation techniques
  - IS10.10 Change management
  - IS10.11 Software tools for project tracking and monitoring
  - IS10.12 Team collaboration techniques and tools

5 Review of Other Computer Science Curricula

This section briefly presents some relevant features of undergraduate CS curriculum in one university in the UK, and one university in Germany. This study is done in order to have an idea about current curriculum in some leading universities at a world-class level. Information about the curriculum of the two universities was obtained from the university web-sites [27, 19] and discussions with academic staff.

5.1 Summary of B.Sc. Computer Science at Leicester University

The current 3-years B.Sc. Computer Science at the Leicester University is designed to meet current technological demands and enable graduates to adapt to future change. Particular attention is paid to Software Engineering and Formal Methods in Software Development. The range of expertise in the department includes Semantics of Programming Languages, Algorithm Design and Engineering, Software System Development and Evolution, Real-Time and Fault-Tolerant Systems, and Theoretical Computer Science.

Leicester is concerned with both theory and practice. Much of the teaching is driven by research, so, many of their modules, especially third year options, consist of state-of-the-art topics. It is important in a rapidly changing subject that they are taught by staff at the forefront of both technology and theory. As part of the degree scheme at Leicester, students learn about the specification, design, development and programming of large computer software systems. The curriculum structure is given in Table 11. The details of the courses can be seen in Appendix C.

Table 11: Curriculum structure university of Leicester.

1st	Program Design
1st	Information Systems
1st	Computer Systems
2nd	Operating Systems and Networks
2nd	Algorithms and Data Structures
2nd	Software Engineering and Professional Practice
2nd	Logic and Discrete Structures
3rd	Software Engineering and System Development
3rd	Functional Programming
3rd	Automata, Languages and Computation
4th	Design and Analysis of Algorithms
4th	Object-Oriented Programming Using C++
4th	Logic Programming
4th	Software Engineering Project
5th	Software Measurement and Quality Assurance
5th	Compression Methods for Multimedia
5th	Formal Software Specifications
5th	Computer Science Project 1
6th	Communication and Concurrency
6th	Semantics of Programming Languages
6th	Computer Science Project 2
6th	Programming Secure and Distributed Systems

The first two years of the degree course are geared to providing a basic understanding of the overall construction of digital computers, knowledge of various programming languages and algorithms, and a general introduction to software engineering. In more detail, in the first year students learn: to write small, well-structured programs; how to develop algorithms and data structures to solve problems; the fundamentals of elementary logic and discrete mathematics (these foundational topics are used throughout the degree); how large software systems are designed and built; and students gain a basic knowledge of computer architectures, operating systems and networks.

In the second year students learn: both the theory and practice of new programming languages; how to design more complex algorithms; the principles of compiler design and the theory of automata and formal languages; and students learn that software engineering is also about programming in the large- by carrying out a substantial project during the course of which they are taught how to break large programming problems into smaller ones which can be easily solved.

The third year consists of a number of specialised courses which take students to the frontiers of current computer science. They also write a substantial dissertation in an area of computer science of their choice: this involves design and implementation work, presentations and orals.

5.2 Summary of B.Sc. Informatik at the Technische Universität München

The Bachelor study of Informatics at the Technische Universität München (TUM) consists of six semesters. In the last year students have to choose a main focus which at the moment can be either databases, software engineering or distributed systems. The curriculum structure is given in Table 12. The detail of the subjects can be seen in Appendix D.

The first year of the degree courses introduces the fundamental concepts of programming, starting from the very beginning with the object-oriented paradigm, imperative and functional programming style. Then they learn fundamental computing algorithms, application of recursion and semantics of functional programs. The first year courses are also geared to providing a basic understanding of the overall knowledge of various programming styles such as: functional programming, imperative programming, object-oriented modelling and component-based programming, event-based programming and, machine-oriented programming. In the sixth semester students have to write a thesis in the area of their main focus.

5.3 Conclusions of Reviews

5.3.1 Leicester Discussion

From the analysis made to Leicester University and discussions with academic staff at Leicester, we can see that the courses taught in the first two years have been created based on the core-hours suggested by CC2001. Leicester curriculum follows the objects-first approach at the introductory phase . Leicester students use Java and C++ as programming languages. The first two years at Leicester give students the fundamentals, knowledge and scientific base so that they can keep on learning on their own once they finished their career. Leicester provides students with presentation and communication abilities, oral as well as writing abilities along the career. The development of these abilities is present in each one of the courses. Leicester curriculum gives students practical abilities required by UK industry, and also the fundamentals to study an academic career.

Table 12: Curriculum structure university of TUM.

1st	Introduction to Informatics I
1st	Mathematics for Engineers I
1st	Technical Foundations of Informatics
1st	Interdisciplinary Courses
2nd	Introduction to Informatics II
2nd	Mathematics for Engineers II
2nd	Lab course on Technical Foundations of Informatics
2nd	Interdisciplinary Courses
3rd	Introduction to Informatics III
3rd	Discrete Structures I
3rd	Basic Algorithms
3rd	Database Systems
4th	Introduction to Informatics IV
4th	Discrete Structures II
4th	Operating Systems
4th	Lab course on Programming
4th	Pro-seminar
5th	Required elective courses
5th	Software Engineering
5th	Main Focus courses
5th	Lab course
5th	Seminar
6th	Required Elective Courses
6th	Main Focus courses
6th	Bachelor Thesis
6th	Seminar

The analysis of Leicester curriculum has given us important lessons that will be taken as a guideline for UCSP curriculum improvement:

Advanced Courses Design Lessons It is interesting to point out that Leicester has designed its advanced courses by answering the question: "What is the best we can teach?". The answer is given by its current research groups and interest. The advanced courses at Leicester are mainly oriented to Semantics of Programming Languages, Algorithm Design and Engineering, Software Systems Development and Evolution, Formal Software Specification, and Algorithms and Data Structures.

One of the important conclusions in this point is that CC2001 says that UCSP must not cover all the advanced courses presented in CC2001, but UCSP should ask itself: "What is the best UCSP can teach?". That means which areas the academic staff has expertise in.

Software Engineering Lessons We want to point out the software engineering area at Leicester University and make a comparative analysis between Leicester best practices and UCSP curriculum.

In the course of *Software Engineering and Systems Development* at Leicester the main aim is to teach the understanding and use of object-oriented methods to analyse, specify, design and implement large

computer systems. This includes topics of software engineering principles, software development process, introduction to object-oriented development, object-oriented requirement capture and analysis, system behavior, object-oriented design, implementing a design and advanced modelling techniques among others. UML is used for consistent specification of the models at different levels of abstraction.

Then, in the *Software Engineering Project* course students put the techniques and skills of the software engineering that have been studied so far into practical use. Students implement a software system in response to a set of customer requirements. (Appendix C shows the details of these courses.)

It is interesting for our analysis to observe how Leicester makes a direct link from the theory (*Software Engineering and Systems Development* course) to the practice (*Software Engineering Project* course).

In the current UCSP curriculum the topics taught in the *Software Engineering and Systems Development* course at Leicester are present in the courses of: *Systems Analysis*, *Systems Design*, and, *Software Engineering*. These courses are taught in the sixth, seventh and eighth semester respectively. The topics that UCSP still has to include in its curriculum are **Testing based on Use Cases**, and, **Mapping a Design to Code**. UCSP courses are also based on object-oriented methods using UML.

If it is true that almost all the topics considered as core are taught at UCSP curriculum in the courses previously mentioned, we believe it is necessary to consider the application of a real project that integrates all the acquired knowledge along the study of these three courses at UCSP . This is to implement a software system in response to a set of customer requirements, as Leicester does in the course of *Software Engineering Project*.

Project Courses Lessons Related with how Leicester makes a direct link from theory to practice, Leicester has two types of project courses: team-project and individual-project.

The team-project is presented in the *Software Engineering Project* course (fourth semester). In this course students put in practice the knowledge acquired in *Software Engineering and Systems Development* course (third semester), as it was mentioned before.

The *Software Engineering Projects* course tries to develop skills in addressing an audience during a talk, skills involved in producing a written report, produce written work in a number of different formats, analyze problems, formulate strategies to solve them, design a plan, carry out the required research, implement and evaluate the solution, recognise the need for information, work effectively in a group, communicate ideas effectively, among others. This kind of projects is carry out in the UK industry. As a result of the project , not only the necessary documentation is presented (analysis diagrams, design diagrams,etc), but also the implementation. There is freedom to choose the developing process to use as well as programming language. Generally, the development is done in Java.

The individual project is organized in two courses in the last year: *Computer Science Project 1* (fifth semester) and *Computer Science Project 2* (sixth semester). The projects are suggested by the academic staff according to their speciality. The first one focuses on the specification and requirements. The second one leads to the implementation of the requirements. As with the team-project, there is freedom to choose the process and the programming language. The project also includes all the documentation. The purpose of these projects is for the students to combine skills acquired in the other computer science modules in the production of a suitable project.

We emphasized these courses at Leicester University because in the current UCSP curriculum there is one course that makes a direct link from theory to practice: *Enterprise Systems* course (tenth semester

- elective course). This course gives students their first real software development experience. UCSP students execute a software project from the specification and requirements to the implementation. The project is developed on-campus. The suggested development process is Metrica 2.0. The project also includes all the documentation (analysis diagrams, design diagrams,etc). This course is focused on allow UCSP students to develop personal and team-work skills.

In addition UCSP has three courses refered to individual-project: *Project I* (sixth semester), *Project II*(seventh semester), and *Project III* (eighth semester). The project courses differs one of each others in the complexity. In general the projects are suggested by the students according to their interests. During these courses students should be able to combine knowledge acquired before in the production of a suitable project. In doing this, students will produce a small software application and a brief technical report.

We believe that the description and discussions about Leicester presented above will be a guideline to improve the projects courses at UCSP.

It is important to point out that Leicester capstone projects are carry out in UK industry. As Arequipa has not developed a software industry, UCSP has in it a restriction in the field of capstone projects.

Formal Specification and Functional Programming courses Leicester students develop skills in functional programming using the language Haskell. This enables them focus on problem solving rather than in implementation details. This is a means of teaching them to think in a abstract way without been imprecise. The current UCSP curriculum includes only the imperative and object-oriented programming. Finally, we want to point out the advanced course on *Formal Software Specifications* at Leicester where students learn the use of mathematics to formally specify the requirements of a computer systems. Both of these courses are not include in the current UCSP curriculum.

As we can observe, the topics analyzed in detail have been those strongly related to the area of Software Engineering. The reason for this is that the UCSP is interested in achieving the development of the software industry in the city of Arequipa according to international standards.

5.3.2 TUM Discussion

The analysis made to TUM curriculum was focused on its introductory phase and on its mathematical foundation. In addition, we consider as relevant to deepen the explanation about the specialized courses (advanced courses) at the TUM curriculum.

The Introductory Curriculum We have identified that TUM follows the functional-first approach where students focus on problem solving rather than in implementation details. That means that the chief concern is to work productively and efficiently, and to minimize the chance of making serious errors. TUM students are encouraged to write algorithms and data structures at a high level without worrying about the details of their machine-level implementation. In addition to the functional programming style, TUM gives instruction in imperative programming, object-oriented programming, component-based programming, event-based programming, and machine-oriented programming. In this ways TUM students are able to produce a variety of work in different programming approaches. TUM represents an example of how to implement the functional-first approach.

Mathematical Foundation Section 2.4.3 pointed out the pervasive role of mathematics whitin CS. TUM includes three courses in the first year that ensure that TUM students receive the foundational mathematics for computer science: *Mathematics for Engineers I* course, *Technical Foundations of Informatics* course, and *Mathematics for Engineers II* course. We can observe that the TUM curriculum has a strong focus on foundational knowledge in the introductory phase. This knowledge is reinforced in the intermediate phase with the *Discrete Structures I* course and the *Discrete Structures II* course. TUM students receive a strong background in discrete structures according to CC2001.

Advanced Courses In the last year TUM students have to focus on speciality areas (Databases, Software Engineering or Distributed Systems). As we can observe in Table 12, TUM students have to choose *Required Elective* courses and *Main Focus* courses.

The current structure of UCSP curriculum (see Appendix A.1) devotes only 8 lecture-hours to elective courses (4 lecture-hours in the nineth semester and 4 lecture-hours in the tenth). These characteristics makes it difficult for UCSP curriculum to include speciality areas since 8 lecture-hours are insufficient to ensure a real focus on an specific area.

We can observe in Appendix A.4 UCSP elective courses are refered to a wide range of areas:

- (PL) Programming Languages: *Concurrent Programming* and *Distributed Systems* courses
- (GV) Graphics and Visual Computing: *Graphics Computing II* course
- (IS) Intelligent Systems: *Expert Systems, Systems Simulation*, and *Neural Networks* courses
- (IM) Information Management: *Strategic Planning of IS, Enterprise Systems*, and *Systems Audit* courses.

There are some elective courses at UCSP curriculum that has not been taught yet: *Systems Audit* course, *Distributed Systems* course and *Graphics Computing II* course.

6 General Recommendations

6.1 Covering the Gap

In Section 4 we made an analysis of UCSP curriculum based on CC2001 and IS2002. Both of these reports were relevant because the current UCSP curriculum presents characteristics and topics that corresponds to both fields.

Section 4.1 showed that the UCSP curriculum covers only 220 out of 280 minimum core-hours (78.57 %) suggested by CC2001. The analysis showed that there exists **units** that are not completely covered in number of core-hours in UCSP curriculum, and, in some cases there are units in which topics are not covered at all.

Recommendation 1 *UCSP should cover the gap completing the units identified and listed in Section 4.1.*

Another gap, is related to the units that must be present in the introductory phase. Section 4.2 showed that UCSP curriculum covers only 42 out of the 50 minimum all-topics core-hour (84 %) and, 23 out of 47 minimum subset core-hours (48.94 %). This percentages are low, because of UCSP teaches some units in the intermediate phase (fifth to eight semester).

Recommendation 2 *UCSP should reassign the identified units listed in Section 4.2 to the introductory phase.*

Section 4.3 showed that UCSP advanced courses are related to some specific areas: Algorithms and Complexity, Programming Languages, Graphics and Visual Computing, Intelligent Systems, Information Management, and Computer Science and Numerical Methods. Section 5.3 pointed out that the design of the advanced courses at the world-class universities is driven by research.

Recommendation 3 *Based on the best practices of the world-class universities, UCSP should asks itself if the advanced courses currently taught answer the question: "It is the best UCSP can teach?". That means to analyze if the courses listed in Section 4.3 (see Table 9) are closer to the UCSP academic staff has as expertise.*

In addition, Section 4.3 pointed out that UCSP is interested in achieving the development of the software industry in Arequipa according to international standards.

Recommendation 4 *According to that goal, UCSP should consider advanced courses related to the Software Engineering area presented in Section 2.4.3 (see Table 5).*

Section 2.4.3 pointed out that mathematics techniques and formal mathematical reasoning are integral to most areas of computer science since mathematics provides a language for working with ideas relevant

to computer science and a theoretical framework for understanding important computing ideas. Section 5.3.2 showed as example the strong mathematical foundation at TUM.

Recommendation 5 *UCSP lecturers should ensure to integrate the mathematics techniques and formal mathematical reasoning in their courses.*

For example to relate the *Algorithm and Design of Algorithms* course with the mathematical topics of counting, permutations and combinations, and probability; to discuss the concurrency and deadlock in the *Operating Systems* course related with graph theory; to point out both program verification and computability build upon formal logic and deduction; to relate the *Functional programming* course with the mathematical concepts and notations for functions.

Regarding to the IS academic field, Section 4.4 showed that the UCSP curriculum covers only 216 out of the 342 points suggested (69.16 %) by IS2002. The missing topics are related mainly to Electronic Business course, Analysis and Logical Design course, and, Network and Telecommunications course, among other specific topics in some others courses.

Recommendation 6 *UCSP should cover the gap completing the topics identified and listed in Section 5.4.*

6.2 Adding new Courses

6.2.1 Formal Methods Course

Section 2.4.3 presented a set of general requirements that support the broad education of computer science students. Mathematical techniques and formal mathematical reasoning are integral to most areas of computer science. Mathematical techniques for specification, development and verification of software systems, often termed formal methods, are now coming into use for the construction of software-systems. The purpose of a formal specification is to set down concisely and unambiguously what is required. In addition Section 2.4.3 presented the *Formal Specification* course as an advanced course in the area of Software Engineering according to CC2001. Section 5.3.1 pointed out that Leicester curriculum has the *Formal Software Specification* course as advanced course.

Recommendation 7 *UCSP should include a Formal Methods course as elective. The suggested course must include the topics recommended by CC2001 in:* ***SE10: Formal methods [25].*** *For this course we recommend 2 lecture-hours and 2 problem-classes hour per week. As tool for this course we suggest RAISE [21] to reinforce analysis and design, or, B [1] to reinforce programming and program refinement. An example of the implementation of a* Formal Methods *course using RAISE is suggested in [18], and an example of a* Formal Software Specifications *course is detailed in Appendix F.*

This suggested course will provide students with the basis of the use of mathematics to formally specify the requirements of a computer system, and then to develop correct code meeting those requirements. It is advisable that this course be taught in the sixth semester in the UCSP curriculum, when students

already have a base in topics of discrete structures(DS).

The *Formal Methods* course should be taught along with the *System Analysis* course at UCSP curriculum, where students could apply key elements and common methods for elicitation and analysis to produce a set of software requirements and use mathematics to formally specify the requirements of a software system (formal methods).

Formal Methods Europe (FME) [20], is a worldwide association bringing together researchers and practitioners in formal methods developing computing systems and software. FME's website provides information about formal method tools, applications, literature, journals, and some other relevant information to be used for UCSP.

6.2.2 Functional Programming Course

Section 5 pointed out some relevant features of undergraduate CS curriculum in some leading universities: Leicester and TUM. As part of Leicester degree, students learn about fundamental concepts of programming, starting from the very beginning with the functional programming style. Leicester gives the students a thorough basis for programming in the functional style using the Haskell language. Students develop skilled use of basic functions and techniques to solve problems in practical applications and do not have to spend much time exploring the language.

TUM students learn about definition, termination, correctness, central functional algorithms and data-structures, type theory, lambda calculus, fixpoint theory, machine-oriented implementation of functional languages. TUM students use OCaml programming language (functional language with OO support).

Right now, the UCSP curriculum only presents imperative and object-oriented programming using Java and C++. This important paradigm is missing in the current UCSP curriculum.

Recommendation 8 *UCSP should include a Functional Programming course. We suggest for this course 2 lecture-hours and 2 problem-classes hour per week. The Functional Programming course must be considered as a pre-requisite course for the Formal Methods course suggested before in this section. An example of the Functional Programming course is shown in Appendix F.*

The suggested course will introduce the basic concepts of programming in the context of a functional language that emphasizes algorithmic strategies over syntactic detail. The minimalistic syntax of functional languages means that courses can focus on more fundamental issues and do not have to spend too much time explaining the language. While forcing students to think in this way is certainly valuable and needs to be part of the UCSP curriculum. Placing this course so early can discourage some students with less experience in that style of thought. It should be expressive enough for a UCSP students to work productively and efficiently.

6.3 Capstone Projects

Section 5.3.1 pointed out lessons about Leicester team-project and individual-project courses. We observed how Leicester makes a direct link from theory to practice. In addition we explained the characteristics of the projects courses at UCSP curriculum (*Project I*, *Project II*, *Project III* and *Enterprise*

Systems).

Recommendation 9 *In order to reinforce and ensure that everything that students learn is also practiced, we suggest some changes and a new organization in the projects courses at UCSP curriculum:*

- *To consider having only two project courses: Project I (seventh semester) and Project II (eighth semester). (Lecture-hours of Project III have been distributed among Project I and Project II)*
- *The first one should be focused on the specification and requirements. The second one should lead to the implementation of the requirements.*

  (We move Project I from sixth to seventh semester, and Project II from the seventh to the eighth semester in order to ensure that students already have the basic knowledge that they need.)
- *The project should be suggested by the lecturers according to the areas they have as expertise.*
- *The team-project course described in Section 5.3.1 (*Systems Enterprise*) should be mandatory. It should be teach in the nineth semester.*

6.4 New Curriculum Structure

In order to take the first step towards an improved curriculum we propose a new curriculum structure (See Table 13). It contains slight changes in the intermediate level (fiveth to nineth semesters). These changes visage to facilitate the implementation of the recommendations pointed out above in this section.

Table 13: UCSP new curriculum proposal.

Course Name	*Lecture*	*Practice*	*Lab*	*Credits*
I Semester				
Basic Mathematics I	4	2		5
Algebra and Geometry	4	2		5
Basic Computing	3		2	4
Introduction to University Life	2			2
Spanish	3			3
Study Methodology	3			3
	19	4	2	22
II Semester				
Calculus single-variable	4	2		5
Linear Algebra	3			3
Discrete Mathematics I	4			4
Programming Language I	2		4	4
Introduction to Philosophy	3			3
Administrative Theory	2	2		3
	18	4	4	22
III Semester				
Calculus multi-variable	4	2		5
Discrete Mathematics II	4			4
Programming Language II	3		2	4
Processes Analysis	2	2		3

Culture History I	3			3
Philosophy Anthropology	3			3
	19	4	2	22
IV Semester				
Numerical Methods	4		2	5
Data Structures I	4		2	5
General Physics	4		2	5
Information Systems	3	2		4
Christian Formation I	3			3
	18	2	6	22
V Semester				
Statistics mathematic	4			4
Analysis and Design algorithms	4		2	5
Logic and Knowledge	4			4
Operating System	3		2	4
Peruvian Reality Analysis	3			3
General Formation Elective	2			2
	18		4	22
VI Semester				
Computing Theory	3			3
Data Structures II	2		2	3
Database I	3		2	4
System Analysis	2		2	3
Formal Methods	2		2	3
Architecture and machine organization	3			3
Culture History II	3			3
	20		8	22
VII Semester				
Software Engineering	4			4
Database II	2		2	3
System Design	4		2	5
Project I	4			4
Social Doctrine of the Churc	3			3
Programming Mathematics	3			3
	20		4	22
VIII Semester				
Compilers	3			3
Stochastic Processes	3			3
Graphics Computing I	3		2	4
Artificial Intelligence	3			3
Project II	4			4
Networking and data communication	2		2	3
Christian Formation II	3			3
	21		4	23
IX Semester				
New Technologies	3		2	4
Managements Projects	3			3
Merchandising	3			3
Internet Services	3		2	4

Group SW-Project	4			4
General Formation Elective	3			3
	19		4	21
X Semester				
Thesis Project	2		8	6
General Economics	3			3
Christian View	3			3
Ethics	2			2
Speciality Elective	4			4
General Formation Elective	3			3
	17		8	21

6.5 Lectures improvement program

One of the features of world-class universities, is that they use less lecturer-hours per-week to cover the body of knowledge. The average assigned is 2 lecture-hours and 1 laboratory-hour per course - per week. For the purpose of this recommendation, we will focus on why some times UCSP lecturers do not cover all the topics considered in the courses they currently taught at UCSP, considering that UCSP curriculum has the double of assigned hours per course - per week. UCSP should ask itself about the best practices of lecturing in the universities on world-class level.

Recommendation 10 *We suggest the UNU-IIST Lecture Improvement Programme to UCSP lecturers.*

According to this programme UCSP lecturers are selected as UNU-IIST fellows, and they study at partner universities in developed countries. These fellowships cover one semester at one of these partner universities, during which time the fellow will be expected to study at least four courses offered by the partner university. These courses may be at either undergraduate or postgraduate level, depending on the specific needs of UCSP (which are normally determined by discussion between the university and a member of UNU-IIST's staff).

The program also provides supporting course materials for all the courses studied by the fellows (lecture material, student's notes, course exercises, recommended text books, etc.), and these become the property of the fellows home departments at the end of the fellowships. The fellows thus have all the knowledge and materials they need to introduce the courses they have studied into the curricula of UCSP after they complete their fellowships.

Regarding to lecture improvement programme it is important to point out that the newest computing research results, relevant information and textbooks are published in English. In addition English is considered as the computing's language. Under this circunstances, UCSP lecturers should be able to communicate effectively in English in order to insert themselves in a cooperative international research environment. A fluent English is hended not only to be able to read, but also to talk, exchange ideas and/or express points of view.

Recommendation 11 *We consider it essential that lecturers at UCSP should develop better communication skills in English. The same holds for the students. To encourage students to improve their English, original English text books should be used. In addition, some courses should be taught in English.*

6.6 Research at UCSP

Computer Science is an enormously vibrant field that continues to evolve at an astonishing pace. New technologies are introduced continuously, and existing ones become obsolete within a few years. In the leading universities in computers science much of the teaching is driven by research, so many of their advanced courses consist of state-of-the-art topics.

Recommendation 12 *In order to maintain high quality education in a rapidly changing area lecturers at UCSP should do research. We recommend that UCSP lecturers should work in research projects at least 6 hours peer week. The research activities could include:*

- *Research in the lecturer's special area of interest.*
- *Developing new projects.*
- *Exploring external funding possibilities.*
- *Establishing and maintaining international scientific contacts.*
- *Incorporate newer results into the lectures.*

6.7 Open Source Software

Section 2.5.1 pointed out that Peru software developed is oriented mainly to SME. Peru has not developed a software development industry. This was presented as a restriction in the field of capstone projects for the UCSP in Section 3.3.3. For the purpose of overcoming this drawbacks an important resource to involve students in real software development projects is the alternative of Open Source Software (OSS).

6.7.1 Open Source Software and UCSP

OSS is usually collaboratively developed through the Internet. A core international group of experts who share a common interest in an application develop about 80 percent of the code. Subsequently, a larger community of users interact with the developers by reporting bugs, suggestions, and code improvements. Experience shows that this concurrent inspection process leads to a quality that can easily compete with that of commercial products. It should be pointed out that in the case of OSS "cheap" does not imply "low-quality". The opposite is, in fact, the case: the offered technology is of high quality, often outperforming commercial products. The high quality is the outcome of team work among idealistic and self-motivated people who freely share their knowledge via the Internet. It is now recognised that it is

the very publication of the source code that makes the validation and verification phase more efficient than that of conventional software [14].

Recommendation 13 *This open feature of the OSS development process creates opportunities for UCSP to participate in the technological developments in this area. At UCSP , this rich source of technological knowledge could be used in the training of UCSP students. That means to carry out large software projects with little resources. UCSP students could studying the features of OSS, work in real OSS projects and in addition they could be encouraged to contribute pieces to OSS.*

6.7.2 Examples of OSS Projects and Technologies

The following list describes some of the most known, useful and successful examples of OSS projects [30]. These projects have a huge potential for improving and maintain quality of computing science education at UCSP. These projects provide a viable and affordable alternative to involve students in real software development projects:

GNU/Linux: Very stable multi-user and multitasking operating system which is being used on a very wide range of computers and devices.

Apache: Is the world's most popular web server. Apache powers more than 60% of the world's web servers. Apache is available for more than 20 platforms, including Linux and Microsoft Windows.

Mozilla: Web Browser, available on Linux, Windows and Mac Operating Systems. Gecko, the engine behind Mozilla is also used in the current versions of Netscape.

OpenOffice: OpenOffice components include word processing, spreadsheets, presentations, drawings, data charting, formula editing, and file conversion facilities (including those for Microsoft Office formats). OpenOffice.org uses EXtensible Markup Language (XML) as a standard for its data formats.

PHP: Recursive acronym for "PHP: Hypertext Preprocessor" is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

Perl: Is a high-level programming language with an eclectic heritage. It is the favourite tool of any system administrator, and runs on most of the UNIX platforms, Windows and Mac.

MySQL: The most popular open source database server in the world, with more than 4 million installations powering websites, datawarehouses, business applications, logging systems and more. Yahoo, Cisco, NASA, Lucent Technologies, Motorola, Google, Silicon Graphics, HP, Xerox and Sony Pictures use MySQL for mission-critical applications.

Some others examples of OSS projects are: Zope, Sendmail, Postfix and BIND.

6.7.3 Adapting OSS to Special Needs of Peru

Section 2.5.1 pointed out that the type of software development in Peru is oriented towards SMEs. Also, it is important to point out that actually SMEs that use tools of OSS are very few because of the lack

of professionals in this area. The growing trend towards broader sharing of knowledge via the Internet provides an opportunity for SMEs in Arequipa and Peru.

Recommendation 14 *UCSP should fill this gap by training students in software development using OSS tools and components in order to provide the needed expertise that could help SMEs, since OSS provides a cheap and high quality source of knowledge. UCSP could start OSS projects to develop software according to the special needs of SMEs and the local community (Center of Excellence on OSS at the university).*

Local OSS projects would generate two main forms of activity that match UCSP education goals and the necessities of SME in Arequipa: production of software as was mentioned before, and, servicing: maintenance, adaptation, training, and other forms of assistance and support.

For example, since SME in Chile presents similar characteristics than SME in Peru, we would mention the case of one SME of the VII Region of the Maule in Chile that has developed its own open source software for an Account System. They have used Python [29] as programming language, GLADE-2[22] as user interface design environment and PostgreSQL [28] as relational database.

Another interesting application is the possibility to adapt free OSS packages to the special needs of a society. Such special needs may range from local language translations of application software to the development of low cost solutions. A South African initiative is currently translating Linux into all 11 official languages [17]. A group in Argentina has developed an optimised Linux version that is tailored to Argentinian users [15]. This operating system can be executed from a CD and does not require installation.

6.8 Library

As we pointed out before, computing is a dynamic and rapidly changing discipline and the only way to achieve and maintain high quality education is having teaching driven by research. In order to support research activities and to keep up to date with the newest developments IEEE-CS and ACM provide technical information through journals and magazines for the the world's computing professionals and researchers.

Recommendation 15 *UCSP should includes in its library a selection of internationally recognized magazines and journals in the computing field.*

Some of the most important magazines and journals provided by IEEE and ACM are listed below:

Journal of the ACM (JACM) [7]: It serves as a venue for careful presentation of theorical research in the core areas of computing: complexity of algorithms, computer architecture, system modeling, artificial intelligence, data structures, database theory and graph theory, to name a few. The authors are world class scientist, writing to other scientists about advances, methods and finding behind the fundamentals.

Communications of the ACM (CACM) [6]: It is the internationally acknowledged premier magazine of the computing field. It presents industry news covering both established and emerging areas

of computer technology. Whether it is embedded systems, graphical user interfaces, security technology, or a host of other topic. Communications of the ACM also offers regular feature columns that comment on such topics as legal issues, practical programming, market trends, business enterprises and cyber citizenry.

ACM Transactions on Information Systems (TOIS) [8]: It appeals to industry practitioners for its wealth of creative ideas, and to academic researchers for its descriptions of their colleagues' work. Though its scope encompasses all aspects of computerized information systems, TOIS most frequently addresses issues in information retrieval and filtering, information interfaces, and information systems design.

ACM Transactions on Software Engineering and Methodology (TOSEM) [5]: It publishes papers on all aspects of that challenge: specification, design, development and maintenance. It covers tools and methodologies, languages, data structures, and algorithms. TOSEM also reports on successful efforts, noting practical lessons that can be scaled and transferred to other projects, and often looks at applications of innovative technologies.

ACM Transactions on Programming Languages and Systems(TOPLAS) [2]: It presents research results on all aspects of the design, definition, implementation, and use of programming languages and programming systems.

ACM Transactions on Database Systems (TODS) [4]: It publishes original archival papers in the area of databases and closely related disciplines. The majority of the papers that have appeared in TODS address the logical and technical foundation of data management. TODS encourages papers that explore the above subjects in the context of large distributed networks of computers, parallel or multiprocessing computers, or new data devices (including data storage devices, data capture devices, and data presentation devices). TODS also encourages papers that describe emerging data-intensive applications that cannot be satisfied by the current database technology.

ACM Transactions on Computer Systems (TOCS) [3]: It publishes the newest findings of the computing research field. Papers published in TOCS are theoretical and conceptual explorations of operating systems, distributed systems and networks. We will find design principles, case studies and experimental results in specification, processor management, memory and communication management, implementation techniques and protocols. TOCS also discusses security and reliability, and offers experience-based papers on all these topics.

IEEE Transactions on Software Engineering (TSE) [23]: The IEEE Transactions on Software Engineering is an archival journal published monthly. It is interested in well-defined theoretical results and empirical studies that have potential impact on the construction, analysis, or management of software. The scope of this Transactions ranges from the mechanisms through the development of principles to the application of those principles to specific environments. Specific topic areas include: development and maintenance methods and models, assessment methods, software project management, tools and environments, system issues, and, state-of-the-art surveys that provide a synthesis and comprehensive review of the historical development of one particular area of interest.

IEEE Transactions on Computers (TC) [24]: The IEEE Transactions on Computers is a monthly publication with a wide distribution to researchers, developers, technical managers, and educators in the computer field. It publishes papers, brief contributions, and comments on research in areas of current interest to the readers. These areas include, but are not limited to, the following: a) computer organizations and architectures; b) operating systems, software systems, and communication protocols; c) real-time systems and embedded systems; d) digital devices, computer components, and interconnection networks; e) specification, design, prototyping, and testing methods and tools; f) performance, fault tolerance, reliability, security, and testability; g) case studies and experimental and theoretical evaluations; and h) new and important applications and trends.

IEEE-CS and ACM provide these magazines and journals in digital format available via the Internet.

7 Conclusions

This report analyzed the current UCSP curriculum based on the revision of the recommendations of CC2001 and IS2002. Both of these reports were relevant for our analysis because the UCSP curriculum has characteristics that correspond to both fields.

According to CC2001 we described six implementations of the introductory curriculum and four implementations of the intermediate curriculum. We have identified that the UCSP curriculum follows the imperative-first approach in the introductory phase and the traditional topic-based approach in the intermediate phase. Our analysis showed that advanced courses at UCSP curriculum are related to the areas of: Algorithms and Complexity, Programming Languages, Graphics and Visual Computing Intelligent Systems, Information Management and Computer Science and Numerical Methods. Since the UCSP is interested in developing a software developing industry we consider it necessary to include advanced courses related to the software engineering area.

Taking CC2001 and IS2002 as reference models this report identified some areas that could be improved in the UCSP curriculum. From the comparative analysis made between the body of knowledge of CC2001 and IS2002, and, the UCSP curriculum we pointed out that UCSP curriculum should cover the gap in those units and topics considered as core. It is important to point out that differences between CC2001 and IS2002 approaches explain some missing core-units showed in this report. Also the analyzis pointed out that it is necessary to reassign some units suggested by CC2001 from the UCSP curriculum intermediate phase to the introductory phase in order to introduce UCSP students to a set of fundamental computer science concepts, and to facilitate the development of cognitive models for these concepts to develop the skills necessary to apply the conceptual knowledge in their further study.

From the study of Leicester and TUM curricula we got important lesson. The Leicester curriculum study focused on the design of advanced courses, software engineering area, and the capstone projects courses. Since this report pointed out that lack of industry in Arequipa is a restriction in the field of capstone projects for UCSP students, OSS projects became an important resource to involve students in real software development. The study of TUM focused on the introductory phase of its curriculum, on its strong mathematical foundation, and on the study of TUM specialised courses. This lessons could be used as guideline to UCSP curriculum improvement.

In order to maintain the hight quality education in computing it is necessary that UCSP lecturers make research. IEEE-CS and ACM provide technical information through journals and magazines that support research activities.

Acknowledgement

Sincere thanks to the UCSP lecturers and academic staff for their support along the development of this work. In a special way to Mr. Luis Diaz Basurco, Mr. Juan Carlos Mendoza, and Mr. Alberto Garcia Garcia for their useful information, feedback and comments about UCSP curriculum. Special thanks to Mr. Percy Pari Salas for all the discussions and for the revision of the early draft of this technical report.

Sincere thanks to Dr. Zhiming Liu for his discussions and suggestions about curriculum development and

curriculum best practices at Leicester University.

The first author is deeply grateful to her supervisor at UNU-IIST, Dr. Bernhard K. Aichernig, for his extensive support, patience and guidance throughout her work, and, for giving her the opportunity to be a fellow at UNU-IIST. The first author is grateful to Mr. Chris George, and, all the academic and administrative staff at UNU-IIST for their support and for given her the opportunity to learn and improve herself during her stay.

References

[1] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press (USA), 1996.

[2] ACM. ACM Transaction on Programming Languages and Systems. http://www.cs.wustl.edu/ toplas/. June 2004 (Last visited).

[3] ACM. ACM Transactions on Computer Systems. http://www.acm.org/tods/. June 2004 (Last visited).

[4] ACM. ACM Transactions on Database Systems. http://www.acm.org/tods/. June 2004 (Last visited).

[5] ACM. ACM Transactions on Software Engineering and Methodology. http://www.acm.org/pubs/tosem/. June 2004 (Last visited).

[6] ACM. Communications of the ACM. http://www.acm.org/pubs/cacm/. June 2004 (Last visited).

[7] ACM. Journal of the ACM. http://www.acm.org/jacm/. June 2004 (Last visited).

[8] ACM. Transactions on Information Systems . http://www.acm.org/pubs/tois/. June 2004 (Last visited).

[9] ACM/AIS/AIT. Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems (IS97). http://192.245.222.212:8009/is97/rev/review1.html. March 2004 (Last visited).

[10] ACM/AIS/AITP. Information Systems 2002 (IS2002). http://www.is2002.org. March 2004 (Last visited).

[11] ACM/AIS/AITP. IS Curriculum Development. http://www.is2002.org. April 2004 (Last visited).

[12] ACM/IEEE. Computing Curricula 1991 Report of the ACM/IEEE-CS Joint Curriculum Task Force (CC1991). http://www.computer.org/education/cc1991/. March 2004 (Last visited).

[13] ACM/IEEE. Computing Curricula 2001 (CC2001) – Steelman Report –. http://www.computer.org/education/cc2001/steelman/cc2001/index.htm. March 2004 (Last visited).

[14] Bernhard K. Aichernig. Open Source Software: Challenges and Prospects for Developing Countries. *UNU-INTECH Technology Policy Briefs*, 2003.

[15] Cesar Brod. Free Software in Latin America. http://www.maailma/kaapeli.fi/america.html. May 2004 (Last visited).

[16] Alonso Mesones Chiape. The Internet Coffe Shop in Arequipa: A Proposal to Generate Sustained Levels of Profit in a Medium Term. Bachelor Degree Thesis, Catholic San Pablo University, 2003.

[17] N.S. Coetzee. Free and Open Source Software in Africa. http://www.maailma.kaapeli.fi/africa.html. May 2004 (Last visited).

[18] Aristides Dasso. A Course on Formal Methods Using RAISE. Technical Report 114, UNU-IIST, P.O.Box 3058, Macau, June 1997.

[19] Department of Informatics at the Technische Universität München. Informatik Bachelor Courses. http://www.in.tum.de/studium/vvSS04-info.html#infbachelor. March 2004 (Last visited).

[20] Formal Methods Europe. Formal Methods Europe. http://www.fmeurope.org/. June 2004 (Last visited).

[21] Chris George, Peter Haff, Klaus Havelund, Anne E. Haxthausen, Robert Milne, Claus Bendix Nielsen, Soren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. Prentice Hall International (UK), 1992.

[22] GLADE. GLADE User Interface Builder - Official Web Site. http://glade.gnome.org. May 2004 (Last visited).

[23] IEEE. IEEE Transactions on Software Engineering. http://www.computer.org/tse/index.htm. June 2004 (Last visited).

[24] IEEE. Transactions on Computers. http://www.computer.org/tc/index.htm. June 2004 (Last visited).

[25] IEEE/ACM. Computing Curricula 2001-Overview of the CS Body of Knowledge. http://www.computer.org/education/cc2001/steelman/cc2001/chapter5.htm. March 2004 (Last visited).

[26] IEEE/ACM. Computing Curricula (CC2001). http://www.computer.org/education/cc2001/. March 2004 (Last visited).

[27] University of Leicester. B.Sc. Computer Science -modules academic year 2003-2004. http://www.cs.le.ac.uk/Modules/CO-03-04/CO-03-04.html. March 2004 (Last visited).

[28] PostgreSQL. PostgreSQL - Official Web Site. http://www.postgresql.org. May 2004 (Last visited).

[29] PYTHON. Python - Official Web Site. http://www.python.org. May 2004 (Last visited).

[30] Niranjan Rajani. Free as in Education: Significance of the Free/Libre and Open Source Software for Developing Countries. http://www.maailma.kaapeli.fi/FLOSSReport1.0.html. June 2004 (Last visited).

A The UCSP Curriculum

A.1 UCSP Curriculum

Table 14: UCSP current curriculum.

Course Name	*Lecture*	*Practice*	*Lab*	*Credits*
I Semester				
Basic Mathematics I	4	2		5
Algebra and Geometry	4	2		5
Basic Computing	3		2	4
Introduction to University Life	2			2
Spanish	3			3
Study Methodology	3			3
	19	4	2	22
II Semester				
Calculus single-variable	4	2		5
Linear Algebra	3			3
Discrete Mathematics I	4			4
Programming Language I	2		4	4
Introduction to Philosophy	3			3
Administrative Theory	2	2		3
	18	4	4	22
III Semester				
Calculus multi-variable	4	2		5
Discrete Mathematics II	4			4
Programming Language II	3		2	4
Processes Analysis	2	2		3
Culture History I	3			3
Philosophy Anthropology	3			3
	19	4	2	22
IV Semester				
Numerical Methods	4		2	5
Data Structures I	4		2	5
General Physics	4		2	5
Information Systems	3	2		4
Christian Education I	3			3
	18	2	6	22
V Semester				
Statistics mathematic	4			4
Analysis and Design algorithms	4		2	5
Logic and Knowledge	4			4
Operating System	3		2	4
Peruvian Reality Analysis	3			3
General Training Elective	2			2
	18		4	22

Course Name	*Lecture*	*Practice*	*Lab*	*Credits*
VI Semester				
Computing Theory	3			3
Data Structures II	4		2	5
Database I	3		2	4
System Analysis	4		2	5
Project I	2			2
Culture History II	3			3
VII Semester				
Compilers	3			3
Programming Mathematics	3			3
Database II	2		2	3
System Design	4		2	5
Architecture and machine organization	3			3
Project II	2			2
Social Doctrine of the Churc	3			3
	20		4	22
VIII Semester				
Stochastic Processes	3			3
Graphics Computing I	3		2	4
Artificial Intelligence	3			3
Software Engineering	4			4
Networking and data communication	2		2	3
Project III	2			2
Christian Education II	3			3
	20		4	22
IX Semester				
New Technologies	3		2	4
Managements Projects	3			3
Merchandising	3			3
Internet Services	3		2	4
Speciality Elective	4			4
General Training Elective	3			3
	19		4	21
X Semester				
Thesis Project	2		8	6
General Economics	3			3
Christian View	3			3
Ethics	2			2
Speciality Elective	4			4
General Training Elective	3			3
	17		8	21

A.2 Study Plan by Vocational Training

Table 15: Study plan by vocational training.

Obligatory Course	*Short-Name*	*Semester*	*Credits*
Computer Science			
Basic Computing	Bas Comp	1	4
Programming Language I	PL I	2	4
Programming Language II	PL II	3	4
Analysis and Design algorithms	ADA	4	5
Data Structures I	DS I	4	5
Logic and Knowledge	L&K	5	4
Data Structures II	DS II	6	5
Compilers	Cmpl	6	3
Computing Theory	Comp Theo	6	3
Graphics Computing I	Graph I	8	4
Artificial Intelligence	AI	8	3
			44
Social and Human Education			
Introduction to University Life	IUL	1	2
Spanish	Span	1	3
Study Methodology	Stu Meth	1	3
Introduction to Philosophy	Int Phil	2	3
Culture History I	Cul His I	3	3
Philosophy Anthropology	Phil Ant	3	3
Christian Education I	Chris For I	4	3
Peruvian Reality Analysis	PRA	5	3
Culture History II	Cul His II	6	3
Social Doctrine of the Church	SDC	7	3
Christian Education II	Chris For II	8	3
Christian view in our time	Chris View	10	3
Ethics	Ethic	10	2
General Economics	Econ	10	3
			40
Mathematics			
Basic Mathematics I	Bas Math	1	5
Algebra and Geometry	A&G	1	5
Linear Algebra	LA	2	3
Discrete Mathematics I	Disc I	2	4
Calculus single-variable	Cal I	2	5
Calculus multi-variable	Cal II	3	5
Discrete Mathematics II	Disc II	3	4
Numerical Methods	Num Meth	4	3
Statistics mathematic	Stat	5	4
Programming Mathematics	Prog Math	7	3
Stochastic Processes	Stoch	8	3
			46

Projects			
Project I	Pjt I	6	2
Project II	Pjt II	7	2
Project III	Pjt III	8	2
Management Projects	Manag Pjt	9	3
Thesis Project	Thesis	10	6
			15
Networking and Communications			
General Physics	Phys	4	5
Operating System	OS	5	4
Architecture and machine organization	Arch	7	3
Networking and data communication	Netw	8	3
Internet Services	Int Serv	9	4
			19
Information Systems			
Administrative Theory	Adm Theo	2	3
Processes Analysis	Proc Anl	3	3
Information Systems	IS	4	4
System Analysis	Sys Anl	6	5
Database I	DB I	6	4
Database II	DB II	7	3
System Design	Sys Dis	7	5
Software Engineering	SwE	8	4
New Technologies	New Tech	9	4
Merchandising	Mer	9	3
			38

Having given the classification of the courses by vocational training showed in Table 15 we summarize the percentage dedicated to each one in Table 16.

Table 16: Percentage by vocational training.

Obligatory Courses	*Total Credits*	*Percentage*
Computer Science	44	22%
Social and Human Education	40	23%
Mathematics	46	20%
Research and Projects	15	7%
Networking and Communication	19	9%
Information System	38	19%
	202	**100%**

A.3 Study Plan by Vocational Areas

Table 17: Study plan by vocational areas.

Course	*Short-Name*	*Semester*	*Credits*
Basic Mandatory Courses			
Basic Mathematics I	Bas Math	1	5
Algebra and Geometry	A&G	1	5
Basic Computing	Bas Comp	1	4
Programming Language I	PL I	2	4
Linear Algebra	LA	2	3
Discrete Mathematics I	Disc I	2	4
Calculus single-variable	Cal I	2	5
Administrative Theory	Adm Theo	2	3
Processes Analysis	Proc Anl	3	3
Calculus multi-variable	Cal II	3	5
Discrete Mathematics II	Disc II	3	4
Programming Language II	PL II	3	4
Numerical Methods	Num Meth	4	3
General Physics	Phys	4	5
Statistics mathematic	Stat	5	4
			63
Speciality Mandatory Courses			
Data Structures I	DS I	4	5
Analysis and Design algorithms	ADA	4	5
Information Systems	IS	4	4
Logic and Knowledge	L&K	5	4
Operating System	OS	5	4
System Analysis	Sys Anl	6	5
Data Structures II	DS II	6	5
Database I	DB I	6	4
Compilers	Cmpl	6	3
Project I	Pjt I	6	2
Architecture and machine organization	Arch	7	3
Database II	DB II	7	3
System Design	Sys Dis	7	5
Project II	Pjt II	7	2
Programming Mathematics	Prog Math	7	3
Graphics Computing I	Graph I	8	4
Artificial Intelligence	AI	8	3
Project III	Pjt III	8	2
Stochastic Processes	Stoch	8	3
Networking and data communication	Netw	8	3
Software Engineering	SwE	8	4
Management Projects	Manag Pjt	9	3
Internet Services	Int Serv	9	4
New Technologies	New Tech	9	4
Speciality Elective Course	Elective	9	4
Speciality Elective Course	Elective	10	4
Thesis Project	Thesis	10	6

			107
General Training Mandatory Courses			
Introduction to University Life	IUL	1	2
Spanish	Span	1	3
Study Methodology	Stu Meth	1	3
Introduction to Philosophy	Int Phil	2	3
Culture History I	Cul His I	3	3
Philosophy Anthropology	Phil Ant	3	3
Christian Education I	Chris For I	4	3
Peruvian Reality Analysis	PRA	5	3
General Education Elective Course	Elective	5	2
Culture History II	Cul His II	6	3
Social Doctrine of the Church	SDC	7	3
Christian Education II	Chris For II	8	3
General Education Elective Course	Elective	9	3
Christian view in our time	Chris View	10	3
Ethics	Ethic	10	2
General Economics	Econ	10	3
General Education Elective Course	Elective	10	3
			48

Having given the classification of the courses by vocational areas presented in Table 17 we summarize the percentage dedicated to each vocational area in Table 18.

Table 18: Percentage by vocational area.

Courses by Vocational Area	*Total Credits*	*Percentage*
Basic Mandatory Course	63	29%
Speciality Mandatory Course	107	49%
General Education Mandatory Course	48	22%
	218	**100%**

A.4 Elective Courses

Table 19: Elective courses.

Speciality Elective Courses			
Course	*Short Name*	*Semester*	*Credits*
Expert Systems	Expert	9	3
Concurrent Programming	Concur	9	3
Strategic Planning of IS	Strat	9	4
Systems Simulation	Simulation	9	3
Neural Networks	NeuralNet	10	3
Knowledge Management Technologies	KnowlMang	10	4
Graphics Computing II	Graph II	10	2
Distributed Systems	Distr Sys	10	3
Enterprise Systems	Enterp Systems	10	4
Systems Audit	Audit	10	4

General Education Elective Courses			
Music	Music	5	2
Theater	Theat	5	2
Arts-Plastics	Art	5	2
Oratory and Expression	Orat	9	3
General Psychology	Psic	9	3
History of the Church	His Church	10	3
Sociology	Socio	10	3
Universal Literature	Liter	10	2

B Computer Science Body of Knowledge

Table 20: Body of knowledge [25].

Discrete Structures(DS)		*Hrs.*	*43 core hours*
DS1	Functions, relations, and sets	6	Core
DS2	Basic Logic	10	Core
DS3	Proof Techniques	12	Core
DS4	Basic of Counting	5	Core
DS5	Graphs and trees	4	Core
DS6	Discrete probability	6	Core
Programming Fundamentals(PF)		*Hrs.*	*38 core hours*
PF1	Fundamental programming constructs	9	Core
PF2	Algorithms and problem-solving	6	Core
PF3	Fundamental data structures	14	Core
PF4	Recursion	5	Core
PF5	Event-driven Programming	4	Core
Algorithms and Complexity(AL)		*Hrs.*	*31 core hours*
AL1	Basic algorithmic analysis	4	Core
AL2	Algorithmic strategies	6	Core
AL3	Fundamental computing algorithms	12	Core
AL4	Distributed algorithms	3	Core
AL5	Basic computability	6	Core
AL6	The complexity classes P and NP		Elective
AL7	Automata theory		Elective
AL8	Advanced algorithmic analysis		Elective
AL9	Cryptographic algorithms		Elective
AL10	Geometric algorithms		Elective
AL11	Parallel algorithms		Elective
Programming Language(PL)		*Hrs.*	*21 core hours*
PL1	Overview of programming languages	2	Core
PL2	Virtual Machines	1	Core
PL3	Introduction to language translation	2	Core
PL4	Declarations and types	3	Core
PL5	Abstraction mechanisms	3	Core
PL6	Object-oriented programming	10	Core
PL7	Functional programming		Elective
PL8	Language translation systems		Elective
PL9	Type systems		Elective
PL10	Programming language semantics		Elective
PL11	Programming language design		Elective
Architecture and Organization(AR)		*Hrs.*	*36 core hours*
AR1	Digital logic and digital systems	6	Core
AR2	Machine level representation of data	3	Core
AR3	Assembly level machine organization	9	Core
AR4	Memory system organization and architecture	5	Core
AR5	Interfacing and communication	3	Core
AR6	Functional organization	7	Core
AR7	Multiprocessing and alternative architecture	3	Core
AR8	Performance enhancements		Elective

AR9	Architecture for networks and distributed systems		
Operating Systems(OS)		*Hrs.*	*18 core hours*
OS1	Overview of operating systems	2	Core
OS2	Operating system principles	2	Core
OS3	Concurrency	6	Core
OS4	Scheduling and dispatch	3	Core
OS5	Memory management	5	Core
OS6	Device management		Elective
OS7	Security and protection		Elective
OS8	File systems		Elective
OS9	Real-time and embedded systems		Elective
OS10	Fault tolerance		Elective
OS11	System performance evaluation		Elective
OS12	Scripting		Elective
Net-Centric Computer(NC)		*Hrs.*	*15 core hours*
NC1	Introduction to net-centric computing	2	Core
NC2	Communication and networking	7	Core
NC3	Network security	3	Core
NC4	The web as an example of client-server computing	3	Core
NC5	Building web applications		Elective
NC6	Network management		Elective
NC7	Compression and decompression		Elective
NC8	Multimedia and data technologies		Elective
NC8	Wireless and mobile computing		Elective
Human-Computer Interaction(HC)		*Hrs.*	*8 core hours*
HC1	Foundations of human-computer interaction	6	Core
HC2	Building a simple graphical user interface	2	Core
HC3	Human-centered software evaluation		Elective
HC4	Human-centered software development		Elective
HC5	Graphical user-interface design		Elective
HC6	Graphical user-interface programming		Elective
HC7	HCI aspects of multimedia systems		Elective
HC8	HCI aspects of collaboration and communication		Elective
Graphics and Visual Computing(GV)		*Hrs.*	*3 core hours*
GV1	Fundamental techniques in graphics	3	Core
GV2	Graphic systems	1	Core
GV3	Graphic communication		Elective
GV4	Geometric modeling		Elective
GV5	Basic rendering		Elective
GV6	Advanced rendering		Elective
GV7	Advanced techniques		Elective
GV8	Computer animation		Elective
GV9	Visualisation		Elective
GV10	Virtual reality		Elective
GV11	Computer vision		Elective
Intelligent Systems(IS)		*Hrs.*	*10 core hours*
IS1	Fundamental issues in intelligent systems	1	Core
IS2	Search and constraint satisfaction	5	Core
IS3	Knowledge representation and reasoning	4	Core

IS4	Advanced search		Elective
IS5	Advanced knowledge representation and reasoning		Elective
IS6	Agents		Elective
IS7	Natural language processing		Elective
IS8	Machine learning and neural network		Elective
IS9	AI Planning systems		Elective
IS10	Robotics		Elective
Information Management(IM)		*Hrs.*	*10 core hours*
IM1	Information models and systems	3	Core
IM2	Database systems	3	Core
IM3	Data Modeling	4	Core
IM4	Relational databases		Elective
IM5	Database query languages		Elective
IM6	Relational database design		Elective
IM7	Transaction processing		Elective
IM8	Distributed databases		Elective
IM9	Physical database design		Elective
IM10	Data mining		Elective
IM11	Information storage and retrieval		Elective
IM12	Hypertext and hypermedia		Elective
IM13	Multimedia information and systems		Elective
IM14	Philosophical frameworks		Elective
Social and Professional Issues(SP)		*Hrs.*	*16 core hours*
SP1	History of computing	1	Core
SP2	Social context of computing	3	Core
SP3	Methods and tools of analysis	2	Core
SP4	Professional and ethical responsibilities	3	Core
SP5	Risk and liabilities of computer-based systems	2	Core
SP6	Intellectual property	3	Core
SP7	Privacy and civil liberties	2	Core
SP8	Computer crime		Elective
SP9	Economic issues in computing		Elective
SP10	Philosophical frameworks		Elective
Software Engineering(SE)		*Hrs.*	*31 core hours*
SE1	Software design	8	Core
SE2	Using APIs	5	Core
SE3	Software tools and environments	3	Core
SE4	Software process	2	Core
SE5	Software requirements and specifications	4	Core
SE6	Software validation	3	Core
SE7	Software evolution	3	Core
SE8	Software project management	3	Core
SE9	Component-based computing		Elective
SE10	Formal Methods		Elective
SE11	Software reliability		Elective
SE12	Specialized systems development		Elective
Computational Science and Num Methods(CN)		*Hrs.*	*no core hours*
CN1	Numerical analysis		Elective
CN2	Operation research		Elective
CN3	Modeling and simulation		Elective

CN4	High-performance computing		Elective

C Leicester University Courses Descriptions

FIRST YEAR COURSES

CO1003-Program Design Basic Java concepts: Java virtual machine,byte-code, applications and applets, source, editors, compilers, development environments.Fundamentals of Java programming: types, classes, objects, packages, assignment.

Structured programming: methods and parameters; for-loops, while-loops, do-loops.Interactive input, file input and output. Selection with if-else; the switch statement.Introduction to exception handling. Strings and string handling, formatting. Overview of design and development concepts: requirements analysis, basic notions of specification.Fundamentals of object-oriented design. Software problems : errors, faults, and failures. Testing – structural testing: flow-graphs, structural coverage.

CO1015-Information Systems Information Systems: What is information and data, the need for information systems and databases. Data modelling: entities and attributes, relationships between entities, values and domains, keys.

Relational databases: basic mathematics sets, relations and classical logic. Relational modelling, relational algebra, views. Database design methodology: ER modelling, Enhanced ER modelling, connection traps, functional dependency, normal forms and normalization. Database implementation.

Microsoft Access. Creating a database and tables, data updating, querying the database, using forms, producing reports, and generating a simple GUI. MySQL. Creating a database and tables, simple queries (selection, projection and joins), reports.

CO1O16-Computer Systems Examples throughout the course will be based on the MIPS Instruction Set Architecture. The top level view of a modern computer: memory, processors, I/O, the fetch, decode, execute cycle. Memory layouts and the Endian systems. The memory hierarchy and simple details of cache memory. The binary number system, elementary logic, and truth-tables. Binary arithmetic: basic definitions, algorithms for computing arithmetic operations. 2s-complement integers. Overflow and correctness conditions. Basic digital electronics: gates for implementing (Boolean models of) simple logical propositions, and the composition of gates to make more complex circuits. Multiplexors, decoders, and related circuits. Clocks. Implementation of atomic Arithmetic Logic Units (ALUs) via digital circuits. Construction of a 32-bit ALU. Simple memory circuits, including caches. Register files.

The MIPS instruction set and simple MIPS programs. A subset of the MIPS language treated in detail at the assembly and machine levels. Semantics, machine fields, branch calculations, and assembly/machine translations. Construction of a simple data path via composition of atomic ALUs. Description of MIPS control program. The interaction of the data-path and control to make a processor. Computing performance: the performance equation and Amdahl's law.

CO1017-Operating Systems and Networks *Operating systems:* Overview; history; processes; hardware features; interrupts. Process management.Programs and processes; multitasking; the dispatcher; scheduling and scheduling policies.Memory management.Memory allocation methods; paging; virtual memory; segmentation; protection and sharing.Input/output. Organization of I/O; de-

vice independence; device handlers; semaphores; buffering. File management. Directory structure; file management techniques; sharing and security; integrity.

Networks: Introduction. Requirement for communication; different sorts of network; layered protocols; connection-oriented and connectionless services.The Physical Layer: Twisted-pair; coaxial cable; fibre optic cable; wireless transmission; limits to communication; representing binary data; the telephone system; multiplexing. The Data Link Layer: Error detection and correction; flow control; channel allocation; protocols for local area networks; bridges.The Network Layer: Datagrams and virtual circuits; routing; congestion control; internetworking; firewalls; the network layer in the Internet. The Transport Layer: Connection management; the transport layer in the Internet; optimizations and congestion control.The Application Layer: Security; Domain Name System; electronic mail; Usenet news; the World Wide Web.

CO1004-Algorithms and Data Structures Advanced object-oriented concepts: inheritance, interfaces, abstract classes.Structured data objects: stacks, linked lists, queues, hash tables,trees.Abstract data types and their implementation in Java. Graphical User Interfaces and their implementation in Java.Event-driven programming in Java. Introduction to applets.Algorithms to handle structured data objects: arrays; sorting and searching.

CO1006-Software Engineering and Professional Practice *Software development:* a brief history of software development, the problems of software development, the software development crisis, a solution- software engineering. The system development process developing large systems, the need for abstraction, the development process. The system development cycle managing the development process, how software is produced, frameworks for system development. Planning a software project describing project goals, evaluating project goals, choosing solutions to meet the project goals.

Quality assurance: what is quality, the development cycle and QA, documentation requirements, validation and verification, reviews and inspections, measuring the development process.

Testing and maintenance: testing the result, behavioural testing techniques, testing for quality.

The professional software engineer: what is software engineering, professional practice, ethics, ensuring quality, standards and procedures, standards and procedures, tools for management.

CO1011-Logic and Discrete Structures *Logic and Formal Specification:* An introduction to reasoning. The notion of proof and examples of proofs. Direct proofs. Proofs by contradiction and contraposition. Propositional logic. Syntax. Debracketing. Main connectives. Turning English statements into formal logic. Truth tables and valuations. Logical equivalence. Interdefinability of connectives. Valid arguments and tautologies. Satisfiability. Semantic tableaux: tableaux rules, open and closed tableaux. Predicate logic. Predicates and names. Quantifiers. Models and counterexamples. Predicate tableaux: tableaux rules, open and closed tableaux. Introduction to formal specifications. The idea of a specification. Basic constructs: types, attributes, initializations, invariants and methods. Pre-conditions and post-conditions. Sets. Elements, subsets and proper subsets. Operations on sets (union, intersection, difference). Basic properties of these operations (including De Morgan's laws). Power sets. Orders of sets. Use in specifications. Ordered pairs, Cartesian products. Relations. Functions: partial and total functions. Use in specifications. Non-determinism and determinism. Sequences. Concatenation and other operations. Use in specifications.

Discrete Structures and Algorithm Analysis : Number systems (integers, rationals, reals). Polynomials (including addition, subtraction and multiplication of polynomials). Solving linear and quadratic equations. Inductive definitions. Proof by induction. Summation of arithmetic series. Graphs. Paths, trees and branches. Composition of relations. Matrices (including adjacency matrices of graphs) and matrix operations. Equivalence relations. Partitions and equivalence classes. Modular arithmetic. Composition of functions. Injections, surjections and bijections. The identity map. Inverse of a function. Factorials. Exponentials, logarithms and their properties. Counting:

permutations and combinations. Pascal's triangle. Elementary probability. Simple recurrence relations and their solution. Summation of geometric series. Basic concepts of algorithm analysis applied to some simple sorting and searching algorithms.

SECOND YEAR COURSES

CO2004-Design and Analysis of Algorithms Examples throughout the course will be based on the MIPS Instruction Set Architecture.Review of the basic notions regarding the asymptotic analysis of algorithms. Review of the methods to design algorithms (divide and conquer, greedy algorithms, heuristics, dynamic programming).A variety of algorithms will be presented to illustrate these methods. They will be drawn from the following list of topics: sorting; searching; string matching; data compression; graph theory.

CO2005-Object-Oriented Programming Using C++ Converting Java to C++. Overview of C++ and comparison with Java, highlighting the differences between the type systems, inheritance, dynamic and static method invocations and exceptions. Memory management issues. Data Structures. Presentation of the basic abstract data types (set, sequence and mapping).Use of the basic abstract data types to model real world data structures. Implementation issues.The Standard Template Library of C++, and the basic structures in it (vector, list, set, map). Use of iterators and algorithms. Low level implementation issues when creating new structures from scratch.

CO2006-Software Engineering and System Development *Introduction:* Software crisis and historical background of Software Engineering. Features of modern software systems, software products and their characteristics: maintainability, dependability, efficiency and usability.

Software Development Process: Requirement analysis; system design; implementation and unit testing; integration and system testing; operation and maintenance; the waterfall model; evolutionary development.

Introduction to OO Development: The inherent complexity of software; mastering complex systems; examples of complex systems; function oriented vs object-oriented methods.Object-oriented requirement capture and analysis: Case study; requirement specification; use cases; conceptual models, use case based project planning; testing based on use cases.System Behaviour: System input events and system operations; contracts; from analysis to design.

OO Design: Interaction diagrams; UML notational issues. creating interaction diagrams, patterns for assigning responsibilities; connecting user interface objects to domain object; design class diagrams; use interaction for testing plan.Implementing a Design: UML notation for interface details; mapping a design to code; container/collection classes in code.

Advanced Modelling Concepts and Design Techniques: Iterative development process; generalization; abstract classes; associative classes; UML notation for packages; modelling behaviour in state diagrams; VDM specification of classes and objects; Configuration Control. Summing Up and Revision

CO2008-Functional Programming Basic types, such as Int, Float, String, Bool; examples of expressions of these types. Functions and declarations, with a high level explanation of a function with general type a1 a2 a3 ... an. Booleans and guards; correspondence of guards with if-then-else expressions. Pairs and n-tuples; fst and snd functions for dismantling pairs and tuples. Pattern matching and cases, especially defining functions on lists and tuples. Numeric calculation. Simple recursion,with examples on the natural numbers and lists; list comprehension;list processing examples which use patterns, recursion and comprehensions. Type inference, basic types, higher types and type variables; informal explanation of the Milner/Damas type inference algorithm; methods

for calculating types of expressions and declared functions. Higher-order functions, polymorphism and code re-use; examples such as the reversal of a list. Mathematical induction and list induction: explanation of the basic principles, together with examples of their application. New data-types such as error types and exception handling and declared data-types; recursively defined data-types. Examples such as lists and trees.

CO2011-Automata, Languages and Computation Revision of mathematical pre-requisites (sets, relations, graphs and functions). Strings. Formal languages.Operations on languages.Concatenation of strings. Kleene star. Finite automata. Language acceptors. Regular languages. Equivalence. Complete automata. The concepts of determinism and non-determinism. The pumping lemma for regular languages. Examples of non-regular languages.

Phrase-structure grammars. Terminals and non-terminals. Regular grammars. Equivalence of regular grammars and finite automata. Closure properties of regular languages. Empty moves. Regular expressions. Stacks. Pushdown automata and context-free grammars. Syntax diagrams and EBNF. Parse trees. Leftmost and rightmost derivations. Equivalence of pushdown automata and context-free grammars. Ambiguous grammars. Inherent ambiguity. Limitations of context-free grammars. Closure properties of context-free languages. Are programming languages context-free? Deterministic context-free languages. Parsing. LL-parsers.

Turing machines. Extensions of Turing machines. Non-determinism. Decision-making Turing machines. Recursive languages. Existence of Turing acceptable languages that are not recursive. The halting problem. The Church-Turing Thesis. Further examples of unsolvable problems. Complexity. Space and time complexity and the relationship between them. Decision making versus acceptance. Determinism versus non-determinism; P and NP. Hierarchy theorems. Reductions and completeness.

CO2014-Logic Programming Fundamentals of Prolog: facts and rules, recursion, backtracking, simple examples; execution mechanism: unification, resolution, SLD-trees and search strategies; list processing in Prolog; cut operator and efficiency issues; worked examples; negation as finite failure; arithmetic; system predicates for IO; assert and retract; debugging programs by tracing.

CO2015-Software Engineering Project The need for software engineering; the project life-cycle; management issues; defining project goals; system specification; system design; management of quality in specification and design; measurement of system attributes.

THIRD YEAR COURSES

CO3001-Formal Software Specifications *Specification:* The need for formalism and abstraction; specifying programs using pre- and post-conditions; sets, sequences and mappings; specifying states abstractly; state invariants; examples.

Program refinement: What is a proof of correctness; annotating programs; proving annotations correct; proof rules for assignment, sequential composition and alternation; iteration, invariants and variants; finding suitable invariants; examples, deriving common algorithms; deriving efficient programs; functions and procedures.

Data refinement: The concept of data refinement; implementing abstract data-types; the concept of a retrieve relation; transforming operation specifications to a new data-type; adequacy and totality of the representation; examples: stacks, queues, linked lists.

CO3007-Communication and Concurrency *Introduction:* An introduction to concurrent and distributed systems, the notions of concurrency, communication and mobility, and a motivation for a formal theory of communication and concurrency.

Modelling concurrency and communication: An introduction, by means of examples, to the basic ideas and principles involved in the modelling of concurrency and communication. Transition rules, inference trees and transition graphs.

Process Algebras: Syntax and operational semantics of CCS.

Equational laws and algebraic reasoning: Equational laws for CCS and their justification. Techniques for equational reasoning.

Bisimulations Strong and weak bisimulations, strong and weak congruences (observational congruence). Techniques for establishing bisimulation equivalences, differences and relationships between various bisimulation relations. Compositional reasoning.

Case studies. Specifications and designs of simple concurrent systems in CCS. Verification of correctness using the Concurrency Workbench tool.

CO3008-Semantics of Programming Languages Inductive definitions and proofs. Rule and structural induction. Transition and evaluation semantics for an imperative language. Proofs of deterministic computation and the equivalence of the transition and evaluation semantics. An abstract machine for the execution of the imperative language. A proof of machine correctness. Eager (call-by-value) evaluation semantics of a functional language with recursive function declarations and imperative features. Lazy (call-by-name) evaluation semantics of the same language. An extension of the functional language with abstractions and local declarations. Further type checking and inference; algorithm W. The SECD machine.

CO3012-Computer Science Project 1 The purpose of the Computer Science Project (Modules CO3012 and CO3013) is for the student to combine skills acquired in the other Computer Science modules in the production of a suitable project. In doing this, the student will assimilate information from a variety of sources and demonstrate the ability to pursue independent study. It is intended that the project should produce some end product for users other than the author. A collection of course exercises, a literature search or a descriptive evaluation would not be suitable.

CO3013-Computer Science Project 2 The purpose of the Computer Science Project (Modules CO3012 and CO3013) is for the student to combine skills acquired in the other Computer Science modules in the production of a suitable project. In doing this, the student will assimilate information from a variety of sources and demonstrate the ability to pursue independent study. It is intended that the project should produce some end product for users other than the author. A collection of course exercises, a literature search or a descriptive evaluation would not be suitable.

CO3095-Software Measurement and Quality Assurance *Quality:* Quality issues in the life-cycle model. Quality planning and management. Quality at the requirements stage: negotiation, setting achievable goals. Aspects of quality: reliability, maintainability, correctness, usability. Risk analysis and management.

Measurement: Theory of measurement. Project size/cost estimation. Algorithmic models: COCOMO, Putnam, etc. Function points and object points. Quality metrics, cost metrics and process metrics. Statistics: data collection and analysis.

Inspection: Code walk-through, inspections, reviews. Comparison of different approaches. Effective follow-up: collection of data, review feedback.

Testing: Different levels: unit testing, integration testing, system testing, validation. Structural testing: coverage techniques. Behavioural testing: domain testing, finite state testing. Mutation and fault seeding. Tools and instrumentation.

Process improvement: SEI process Capability Maturity Model. Standards: ISO 9001, SPICE (ISO/IEC 15504). Cleanroom approach. Formal methods.

CO3097-Programming Secure and Distributed Systems Distributed Programming: Networking in Java using sockets and streams, multi-threaded computation, liveness, safety, deadlock, livelock, mutual exclusion, communication protocols, semaphores and monitors, synchronous and asynchronous message passing in Java, client-server architectures.

Security: Security issues and concerns. Key management including generation, translation, agreement protocols and management paradigms. Authentication including message digests, MAC's, signatures, certificates. Symmetric and antisymmetric ciphers, eg DES, IDEA, RSA.

Java Support: Java security, Java Cryptography Architecture (JCA), Java Cryptography Extension (JCE), Java remote method invocation (RMI).

D TUM University Courses Description

Introduction to Informatics I Computer systems; classes and objects; algorithms and text-replacement systems; object-based and imperative programming style; orders, sorting and search; linked lists; functional programming style; applications of recursion: quicksort and trees; information and representation; object-oriented programming style; abstract methods (OO); design patterns; term-rewriting systems; semantics of functional programs; OCaml programming language (functional language with OO support).

Functional programming Definition, termination, correctness, central functional algorithms and data-structures, type theory, lambda calculus, fixpoint theory, machine-oriented implementation of functional languages;

Imperative programming: history and machine models, control and data-structures (references and dynamic data-structures), algorithms and imperative paradigms, Hoare-calculus, machine-oriented implementation of imperative languages;

Object-oriented modelling and component-based programming: modelling tools SADT, SA, UML, concepts (classes, objects, inheritance, aggregation, association), design patterns, abstract classes, interfaces, development tools (Eiffel, C++, Java);

Event-based programming: problem and history, communication and synchronisation, temporal logic and specification tools, examples of languages and environments;

Machine-oriented programming: history, machine architectures, typical assembler languages

E IS2002 Course Specification

IS2002.P0-Personal Productivity with IS Technology *Catalog:* Students with minimal skills will learn to enhance their personal productivity and problem solving skills by applying information technologies to problem situations and by designing and using small information systems for individuals and groups.

Scope: This prerequisite course enables students to improve their skills as knowledge workers. The emphasis is on personal productivity concepts using functions and features in computer software such as spreadsheets, databases, presentation graphics, and Web authoring. Although identified as a course, this material can be delivered as self-study modules, as modules associated with other courses using the software, or as a full course.

Topics: Knowledge work productivity concepts; advanced software functionality to support personal and group productivity such as templates and macros; reuse rather than build from scratch; organization and management of data (sorting, filtering) via spreadsheets and database tools; accessing organizational and external data; information search strategies; tool use optimization and personalization; professional document design; Web page design and publishing; effective presentation design and delivery.

IS2002.1-Fundamentals of Information Systems *Catalog:* Systems theory, quality, decision making, and the organizational role of information systems are introduced. Information technology including computing and telecommunications systems are stressed. Concepts of organizations, information systems growth, and process improvement are introduced.

Scope: This course provides an introduction to systems and development concepts, information technology, and application software. It explains how information is used in organizations and how IT enables improvement in quality, timeliness, and competitive advantage.

Topics: Systems concepts; system components and relationships; cost/value and quality of information; competitive advantage of information; specification, design, and re-engineering of information systems; application versus system software; package software solutions; procedural versus non-procedural programming languages; object oriented design; database features, functions, and architecture; networks and telecommunication systems and applications; characteristics of IS professionals and IS career paths; information security, crime, and ethics. Practical exercises may include developing macros, designing and implementing user interfaces and reports; developing a solution using database software.

Discussion: Students with practical end-user knowledge will study systems theory and quality concepts as an introduction to information technology concepts and information systems development. Structure and functions of computers and telecommunications systems will be examined. Standard systems purpose and organization will be introduced.The concept that information is of significance in stating and attaining organizational goals will be used as the basis for exploring the development of databases to store information. Information systems will be introduced to process and communicate the information. The dynamic nature of organizations and the necessity for growth and re-design of the organization as well as its information systems will be presented and used as the motivator for understanding information systems development methodologies.The development path for entry level to senior information systems professionals will be explained. Professional ethical expectations and obligations will be reviewed. The necessity for personal and interpersonal communications skills will be discussed.

IS2002.2-Electronic Business Strategy, Architecture and Design

*Catalog:*The course focuses on the linkage between organizational strategy and networked information technologies to implement a rich variety of business models in the national and global contexts connecting individuals, businesses, governments, and other organizations to each other. The course provides an introduction to e-business strategy and the development and architecture of e-business solutions and their components.

Scope: This course examines the linkage of organizational strategy and electronic methods of delivering products, services and exchanges in inter-organizational, national, and global environments. Information technology strategy and technological solutions for enabling effective business processes within and between organizations in a global environment are considered.

Topics: Electronic commerce economics, business models, value chain analysis, technology architectures for electronic business, supply chain management, consumer behavior within electronic

environments, legal and ethical issues, information privacy and security, transborder data flows, information accuracy and error handling, disaster planning and recovery, solution planning, implementation and rollout, site design, Internet standards and methods, design of solutions for the Internet, intranets, and extranets, EDI, payment systems, support for inbound and outbound logistics.

Discussion: A course in electronic business has during the recent years become an integral part of the Information Systems curriculum. The proliferation of Internet technologies has had a profound impact on the way business is conducted both in for-profit and not-for-profit organizations. It is essential that an IS curriculum prepares graduates to function in environments where the integration of computing and communication technologies is reshaping organizations and their fundamental processes, regardless of their industry.This course integrates various perspectives. First, it is essential to provide a good understanding of the changes in the business environment enabled by modern information and communication technologies. Thus, the course should discuss topics such as new business models, the economics of e-business, and value chains and value networks. In addition, it is essential that the students have a strong understanding of the operational issues that are critical to every successful e-business solution, such as marketing, logistics, and payment methods using the electronic tools.

Second, the students need to understand how e-business systems are linked to the organizational environment and how they affect and are affected by the context in which they are built. Therefore, the course needs to cover the legal and ethical aspects of the design and development of e-business solutions, special challenges related to global e-business systems (e.g., transborder data flows, differences in privacy legislations, operating in a multicurrency environment), and the societal effects of the widespread usage of e-business technologies. Third, the course should provide the students with an introduction to the technical architecture and the detailed technology solutions that are required to implement reliable and efficient e-business solutions. This includes infrastructure (computing hardware and software, networking hardware and software, support, disaster recovery), development methods and technologies specific to the e-business environments, web interface development, and e-business solution deployment.

IS2002.3-Information Systems Theory and Practice

Catalog: Students who have constructed personal information systems will be exposed to the theory of the Information Systems discipline. Application of these theories to the success of organizations and to the roles of management, users, and IS professionals are presented. *Scope:* This course provides an understanding of organizational systems, planning, and decision process, and how information is used for decision support in organizations. It covers quality and decision theory, information theory, and practice essential for providing viable information to the organization. It outlines the concepts of IS for competitive advantage, data as a resource, IS and IT planning and implementation, change, and project management. *Topics:* Systems theory and concepts; information systems and organizational system; decision support; quality; level of systems: strategic, tactical, and operational; system components and relationships; information systems strategies; roles of information and information technology; roles of people using, developing, and managing systems; IS planning and change management; human-computer interface; IS development process; evaluation of system performance; societal and ethical issues related to information systems design and use. *Discussion:* Students who have end-user skills who have implemented personal productivity systems using personal productivity work tools will be prepared to use the information systems theory presented in this course. The course presents the basic concepts for use in subsequent courses; the systems point of view, the organization and development of a system, information flows, the nature of information systems, and basic techniques for representing systems structure. Learning, goal setting and achieving, decision making, and other characteristics of individuals, groups, and teams are explored. Organizational models and planning are presented. Quality concepts are explained.

IS planning and development activities are explored in the organizational context of management and users. Cross-functional management and user teams are discussed.

IS2002.4-Information Technology Hardware and System Software

Catalog: Principles and application of computer hardware and software will be presented through lecture of the theoretical underpinnings, installation, configuration, and operational laboratory experiences.

Scope: This course provides the hardware/software technology background to enable systems development personnel to understand tradeoffs in computer architecture for effective use in a business environment. System architecture for networked computing systems and operating systems.

Topics: Hardware: CPU architecture, memory, registers, addressing modes, busses, instruction sets, multi processors versus single processors; peripheral devices: hard disks and other storage devices, video display monitors, device controllers, input/output; operating systems functions and types; operating system modules: processes, process management, memory and file system management; examples and contrasts of hardware architectures and operating systems.

Discussion: Students who are knowledgeable of and have developed personal information systems will gain an in-depth exposure to information technology hardware and software components and their interaction.A systems view of computer systems will be utilized in identification of computer system components. Peripheral devices will be identified and principles of operation will be studied and learned. The operating system software, including I/O drivers and extensions to the operating system will be examined, learned and utilized in the laboratory. Organization of the operating system will be studied to understand how concurrent processes, scheduling, memory management, and I/O are accomplished. The flow of information in the operating system in relation to the computer and to application software will be considered.Standards, standard organizations and resulting hardware and software consequences will be identified and studied. General principles will be expressed.

IS 2002.5-Programming, Data, File and Object Structures *Catalog:* This course presents object oriented and procedural software engineering methodologies in data definition and measurement, abstract data type construction and use in developing screen editors, reports and other IS applications using data structures including indexed files.

Scope: This course provides an exposure to algorithm development, programming, computer concepts, and the design and application of data and file structures. It includes the use of logical and physical structures for both programs and data. *Topics:* Data structures and representation: characters, records, and files; precision of data; information representation, organization, and storage; algorithm development; programming control structures; program correctness, verification, and validation; file structures and representation. Programming in traditional and visual development environments that incorporate event-driven, object-oriented design.

Discussion: Students will gain in-depth understanding of defining and measuring events that produce data, both simple and complex, and principles, concepts, and practices of successful software development.Formal problem solving strategies will be presented. Program design methods and strategies including top down implementation will be discussed and implemented. Graphic programming environments will be explored. Capabilities of a number of programming languages will be presented. Skill will be developed in at least one language supporting an indexed file system. Software engineering principles will be practiced with a systems view. Students will learn to recognize objects and abstract data types, concepts of event driven and data flow models, module identification, modularity including parameters, module naming, cohesion, coupling desired and erroneous practices, and testing. Verification and validation methods will be presented and practiced by generating small modules and larger programs. Specific data structures including arrays,

records, stacks, queues, and trees will be created and used. The course will provide an introduction to the use of predefined user interface components.

IS2002.6-Networks and Telecommunication *Catalog:* Students will gain in-depth experience of networking and telecommunications fundamentals including LANs, MANs, WANs, intranets, the Internet, and the WWW. Data communication and telecommunication concepts, models, standards, and protocols will be studied. Installation, configuration, systems integration and management of infrastructure technologies will be practiced in the laboratory.

Scope: This course provides an in-depth knowledge of data communications and networking requirements including networking and telecommunications technologies, hardware, and software. Emphasis is upon the analysis and design of networking applications in organizations. Management of telecommunications networks, cost-benefit analysis, and evaluation of connectivity options are covered. Students learn to evaluate, select, and implement different communication options within an organization. *Topics:* Telecommunication configurations; network and Web applications; distributed systems; wired and wireless architectures, topologies, and protocols; installation, configuration, and operation of bridges, routers, switches, and gateways; network performance tuning; privacy, security, firewalls, reliability; installation and configuration of networks; monitoring and management of networks; and communications standards.

Discussion: Students who have used networking technologies to complete assignments in previous courses and who are knowledgeable of the significance of information technology in facilitating information systems will be given an opportunity in this course to gain considerable depth in networking, both theoretically and through practical laboratory experience. Students will learn about some of the significant networking standards and the organizations that have developed the standards. The ISO seven-layer model and the TCP/IP model will be used as organizing frameworks. The ITU and IEEE standards will be reviewed and global telecommunication policies and competing standards will be examined.The technology supporting communications providers, satellite communications, as well as local and metropolitan systems will be explored. Devices including media, modems, multiplexers, computer interfaces, switches, and routers will be studied.Acquisition, installation, configuration, and other details of management of the various technologies will be studied.

IS2002.7-Analysis and Logical Design *Catalog:* Students with information technology skills will learn to analyze and design information systems. Students will practice project management during team oriented analysis and design of a departmental level system.

Scope: This course examines the system development and modification process. It emphasizes the factors for effective communication and integration with users and user systems. It encourages interpersonal skill development with clients, users, team members, and others associated with development, operation, and maintenance of the system. Structured and object oriented analysis and design, use of modeling tools, adherence to methodological life cycle and project management standards. *Topics:*Life cycle phases: requirements determination, logical design, physical design, and implementation planning; interpersonal skills, interviewing, presentation skills; group dynamics; risk and feasibility analysis; group-based approaches: project management, joint application development (JAD), and structured walkthroughs; structured versus object oriented methodologies; RAD, prototyping; database design; software package evaluation, acquisition, and integration; global and inter-organizational issues and system integration; professional code of ethics.

Discussion: Students with the basic skills of information technology will learn to gather information in order to identify problems to be solved. They will determine system requirements and a logical design for an information system. A project of limited scope will be designed during this

course.Students will investigate alternative solutions, and will determine feasibility of solutions. They will identify value added by the completion of the system. Students will be exposed to methods to support each stage of the development process. While automated tools are not a substitute for understanding of the processes involved, they may be used to ensure that a particular methodology is used rigorously. If manual methods are used, it is important to define the methodology thoroughly. Project management will be taught and used to control the team project. Team concepts including personal and interpersonal skills will be discussed and monitored. Empowerment concepts will be used and measured. Scheduling and completing individual and group actions will be used to ensure project milestone completion.

IS2002.8-Physical Design and Implementation with DBMS

Catalog: Students successfully completing the analysis and logical design course will continue in this course to learn to develop the detailed physical design and implementation of a logical design requiring implementation.

*Scope:*This course covers information systems design and implementation within a database management system environment. Students will demonstrate their mastery of the design process acquired in earlier courses by designing and constructing a physical system using database software to implement the logical design.

Topics: Conceptual, logical, and physical data models, and modeling tools; structured and object design approaches; models for databases: relational and object oriented; design tools; data dictionaries, repositories, warehousing, and data mining; database implementation including user interface and reports; multi-tier planning and implementation; data conversion and post implementation review. *Discussion:* Students who have completed the information analysis and logical design course will engage in the physical design and implementation process for an information system of a limited scope.Automated tools or manual methods will be used within a team oriented project environment to design and implement a departmental information system requiring an enterprise level database.A data model will be developed to guide the detailed design process for database construction. A corresponding functional analysis of the problem will be completed. Program specifications will be developed and utilized in construction of the physical system. Unit testing, integration, and integration testing of the final system will be accomplished. Tools will be used to measure the complexity of solutions; quality assurance measures implemented as project standards will be used to control project quality and risk. Code generators or libraries will be used to facilitate rapid development of the desired system. Existing project management software will be used to manage user expectation and completed work.

IS2002.9-Physical Design and Implementation in Emerging Environments *Catalog:* Students who have completed the analysis and logical design course will extend their knowledge by implementing an information system in an emerging systems environment. Teams will use project management principles to implement an information system.

*Scope:*This course covers physical design and implementation of information systems applications. Implementation in emerging distributed computing environments using traditional and contemporary development methodologies.

Topics: Topics may include selection of development environments and standards; structured, event driven, and object oriented application design; testing; software quality assurance; system implementation; user training; system delivery; post implementation review; configuration management; maintenance; multi-tiered architectures and client independent design.

Discussion: Students will utilize a contemporary development environment to implement a project that spans the scope of the previous courses. If object-oriented programming has not been taught to

the students earlier in the curriculum, then it should be used here. If only object-oriented methods have been used, some procedural methods should be employed. System or object representation, modular design, use of control structures with proof of correctness, verification, testing and validation should be integral components of software quality assurance. Implementation standards should be developed by the students and used rigorously as project teams complete a significant system. A conversion and training plan should be developed and implemented involving hardware, data, people, and software systems. Project management tools should be used to ensure timely completion of the project. Interdependence skills should be practiced and evaluated. A contemporary methodology should guide the project sequence.

IS2002.10-Project Management and Practice *Catalog:* Advanced IS majors operating as a high-performance team will engage in and complete the design and implementation of a significant information system. Project management, management of the IS function, and systems integration will be components of the project experience.

Scope: This course covers the factors necessary for successful management of information systems development or enhancement projects. Both technical and behavioral aspects of project management are applied within the context of an information systems development project.

Topics: Managing the system life cycle: requirements determination, design, implementation; system and database integration issues; network management; project tracking, metrics, and system performance evaluation; managing expectations of managers, clients, team members, and others; determining skill requirements and staffing; cost-effectiveness analysis; reporting and presentation techniques; management of behavioral and technical aspects of the project; change management. Software tools for project tracking and monitoring. Team collaboration techniques and tools. *Discussion:* This is the capstone course for IS majors who have completed the systems analysis and design sequences. It focuses on engaging in and completing a major system development project.Within the project context management of IS, systems integration is an explicit requirement for students to address. The project is a team effort and allows a final opportunity to practice personal and interdependence skills to ensure team member empowerment and success. Project management tools will be employed by the team to ensure tracking of the project and communication of project goals and accomplishments to the client. Automated development tools may or may not be used depending on available resources. However, standards will be developed for all project deliverables. Software quality assurance methodologies will be employed to ensure a successful outcome for the project.On-going presentation of project planning, analysis, design, conversion plan, and other documentation will be done by the team. Each team member should play a significant role in some aspect of presentation.

F Adding Courses

F.1 Formal Software Specifications Course - Leicester

Table 21: Functional programming course - Leicester.

CO3001 Formal Software Specifications			
Credit : 20			Semester: 5
Prerequisites:	essential: CO1003, CO1011, CO2006		
Assessment:	Coursework: 30%	Three hour exam 70%	
Lectures:	36	**Problem Classes:**	12
Tutorials:	none	**Private Study:**	90
Labs:	none	**Seminars:**	none
Project:	none	**Other:**	none
Surgeries:	12	**Total:**	150

SUBJECT KNOWLEDGE

Aims: To appreciate the benefits of writing abstract specifications describing the desired behaviours of computer systems. To appreciate the benefits of using formal techniques to develop programs.

Learning Outcomes: At the end of the course the student should be able to:

- Write an abstract specification describing the requirements of a computer system
- Develop a correct program from a specification
- Implement an abstract datatype.

Methods Class: Class sessions together with course notes, recommended textbook, worksheets, printed solutions, and some additional hand-outs.

Assessment: Unassessed coursework, assessed coursework, traditional written examination.

EXPLANATION OF PRE-REQUISITES: This course uses mathematics to specify and reason about computer programs. The course builds upon the object-oriented specification and design techniques, and the introduction to VDM seen in CO2006. Some of the algorithms derived will previously have been seen in CO2004.

COURSE DESCRIPTION: This course demonstrates the use of mathematics to formally specify the requirements of a computer system, and then to develop correct code meeting those requirements. One of the most common causes of errors in computer systems is that the requirements are incorrectly understood. The purpose of a formal specification is to set down concisely and unambiguously what is required. A good specification concentrates on specifying what is required, not how it will be achieved. To this end, states are specified abstractly (for example, using sets and mappings rather than concrete datatypes such as linked lists or arrays), and specifications are written for the required operations,

describing how the state should change. The process of going from a specification to an implementation is known as refinement. There are two forms of refinement: data refinement and program refinement. Data refinement is concerned with the replacement of the high-level abstract datatypes (e.g.sets and mappings) by the implementable data-types found in languages such as Pascal (e.g.linked lists and arrays). Program refinement is the other part of software development: that of turning specifications into algorithms.

SYLLABUS

Specification: The need for formalism and abstraction; specifying programs using pre- and post-conditions; sets, sequences and mappings; specifying states abstractly; state invariants; examples.

Program refinement: What is a proof of correctness; annotating programs; proving annotations correct; proof rules for assignment, sequential composition and alternation; iteration, invariants and variants; finding suitable invariants; examples, deriving common algorithms; deriving efficient programs; functions and procedures.

Data refinement: The concept of data refinement; implementing abstract datatypes; the concept of a retrieve relation; transforming operation specifications to a new datatype; adequacy and totality of the representation; examples: stacks, queues, linked lists.

F.2 Functional Programming Course - Leicester

Table 22: Functional programing course - Leicester.

CO2008 Functional Programming			
Credit : 20			Semester: 3
Prerequisites:	essential: CO1003, CO1004, CO1011		
Assessment:	Coursework: 40%	Three hour exam 60%	
Lectures:	36	**Problem Classes:**	none
Tutorials:	none	**Private Study:**	78
Labs:	24	**Seminars:**	none
Project:	none	**Other:**	none
Surgeries:	12	**Total:**	150

SUBJECT KNOWLEDGE

Aims: The module will give the student a thorough grounding for programming in the functional style using the language Haskell.

Learning Outcomes: Students will be able to demonstrate: skilled use of basic functions and techniques to solve problems in practical applications; the use of higher order functions in designing functions which allow code re-use; the basics of polymorphic type inference; and understanding Haskell's mechanism for defining new datatypes. They should be able to apply polymorphism to design functions which allow code re-use.

Methods Class: Class sessions together with lecture slides, recommended textbook, worksheets, printed solutions, and some additional hand-outs and web support.

Assessment: Marked coursework, traditional written examination.

SUBJECT SKILLS

Aims: To teach students how to methodically solve problems given the techniques available to them.

Learning Outcomes: Students will be able to: produce a variety of work in different formats; breakdown a problem to identify essential elements; apply prior knowledge of subject to analyse problems; generate and evaluate a range of strategies to address a problem; design a plan to solve a problem; implement a planned solution and evaluate the implementation.

Methods: Class sessions together with worksheets.

Assessment: Marked coursework, traditional written examination.

EXPLANATION OF PRE-REQUISITES: It is essential that students have taken a first course in basic (imperative) programming, which includes skills involving both coding and design, to a level which includes data structures such as lists and trees. A grounding in the basic mathematical concepts of sets, functions and relations is required, but detailed knowledge of other areas of elementary discrete mathematics and logic is not essential.

COURSE DESCRIPTION: Many of the ideas used in imperative programming arose through necessity in the early days of computing when machines were much slower and had far less memory than they do today. Languages such as C(++) and Pascal carry a substantial legacy from the past. Even Java, despite its OO features, has been devised to look 'a bit like C'. If one were to start again and design a programming language from scratch what would it look like?

For many applications, the chief concern should be to produce a language which is concise and elegant. It should be expressive enough for a programmer to work productively and efficiently but simple enough to minimize the chance of making serious errors. Rapid development requires the programmer to be able to write algorithms and data structures at a high level without worrying about the details of their machine-level implementation. These are some of the criteria which have led researchers to develop the functional programming language Haskell.

The flavour of programming in Haskell is very different from that in an imperative language. Much of the irrelevant detail has been swept away. For example, there are at least three different uses for a variable in Pascal: as a storage location, as parameter in a function or procedure, as a temporary name for another variable (a 'var parameter'). There is only one use for a variable in Haskell: it stands for a quantity that you don't yet know, as is standard in mathematical practice. The constructs in Pascal include expressions, commands, functions and procedures (the last two are confused in C); whereas in Haskell there are only expressions and functions. The meaning of a program in Pascal or C is understood by the effect it has on the 'state' of the machine as it runs. Haskell does away with the idea of 'state' – the meaning of a program is understood by the values it computes.

On the other hand, Haskell is a very expressive language. The type system allows functions to be written polymorphically so that the same code can be re-used on data of different types, e.g. the same length function works equally well on lists of integers as on lists of reals or lists of strings. Furthermore, it allows one to write functions which take other functions as parameters. These are known as higher order functions and they give a second form of code re-use. There are powerful mechanisms for introducing

user-defined datatypes such as trees, sets, graphic objects, etc. Haskell also makes a great deal of use of recursion. The combination of these features makes for very clean, short programs, which, with some experience, are easier to understand than many imperative programs.

This course teaches how to program in Haskell, which exemplifies the functional style, and also a little of the ideas on which functional programming is founded.

SYLLABUS: Basic types, such as Int, Float, String, Bool; examples of expressions of these types. Functions and declarations, with a high level explanation of a function with general type a1 a2 a3 ... an. Booleans and guards; correspondence of guards with if-then-else expressions. Pairs and n-tuples; fst and snd functions for dismantling pairs and tuples. Pattern matching and cases, especially defining functions on lists and tuples. Numeric calculation. Simple recursion, with examples on the natural numbers and lists; list comprehension; list processing examples which use patterns, recursion and comprehensions. Type inference, basic types, higher types and type variables; informal explanation of the Milner/Damas type inference algorithm; methods for calculating types of expressions and declared functions. Higher-order functions, polymorphism and code re-use; examples such as the reversal of a list. Mathematical induction and list induction: explanation of the basic principles, together with examples of their application. New datatypes such as error types and exception handling and declared datatypes; recursively defined datatypes. Examples such as lists and trees.

G Analysis CC2001/IS2002 and UCSP Curriculum

In this Appendix taking CC2001 and IS2002 as reference models, we show a comparative analysis between UCSP curriculum and the Body of Knowledge CC2001, the Introductory Courses CC2001, the Advanced Courses CC2001, and the IS2002 Courses.

Figure 1: CC2001 body of knowledge and UCSP curriculum.

			Semester	1	1	2	2	3	3	4	4	5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	9	9	9	10	10	10
			Course UCSP	Bas Comp	Bas Math	PL I	Disc I	PL II	Disc II	DS	IS	ADA	OS	L & K	Stat	DS II	Comp Theo	DB I	Sys Anl	Cmpl	DB II	Sys Dis	Arch	Grap h I	AI	Sw E	Netw	New Tech	Int Serv	Exp ert	Graph II	Neural Net	Distr Sys
Discrete Structures (DS)		***Core Hrs***																															
DS1.	*Functions, relations, and sets*	*6*					*6*																										
DS2.	*Basic logic*	*10*		10																													
DS3.	Proof techniques	*12*					*12*																										
DS4.	*Basics of counting*	*5*							5																								
DS5.	*Graphs and trees*	*4*							4																								
DS6.	*Discrete probability*	*6*													4																		
		43		*41*																													
Programming Fundamentals (PF)																																	
PF1.	*Fundamental programming constructs*	*9*	**7**		**2**																												
PF2.	*Algorithms and problem-solving*	*6*	**6**																														
PF3.	*Fundamental data structures*	*14*	**4**		**4**					**6**																							
PF4.	*Recursion*	*5*										**5**																					
PF5.	***Event-driven programming***	*4*						**0**																									
		38		*34*																													
Algorithms and Complexity (AL)																																	
AL1.	*Basic algorithmic analysis*	*4*										**4**																					
AL2.	*Algorithmic strategies*	*6*										**6**																					
AL3.	*Fundamental computing algorithms*	*12*										**12**																					
AL4.	***Distributed algorithms***	*3*										**0**																					
AL5.	*Basic computability*	*6*															**6**																
AL6.	*The complexity classes P and NP*																																
AL7.	*Automata theory*																																
AL8.	*Advanced algorithmic analysis*																																
AL9.	*Cryptographic algorithms*																																
AL10.	*Geometric algorithms*																																
AL11.	*Parallel algorithms*																																
		31		*28*																													

** For this analysis, only a subset of UCSP courses showed in Appendix A has been considered. The analyzed courses are those that keep a close relation with the units presented in the Body of Knowledge showed in Appendix B.*

** The Underlined Knowledge Units correspond to those that are not at present included in the UCSP curriculum or those that do not cover the total of core-hours*

** Shadow rows correspond to the CC2001 core-hours*

Figure 2: Figure 1. (continued)

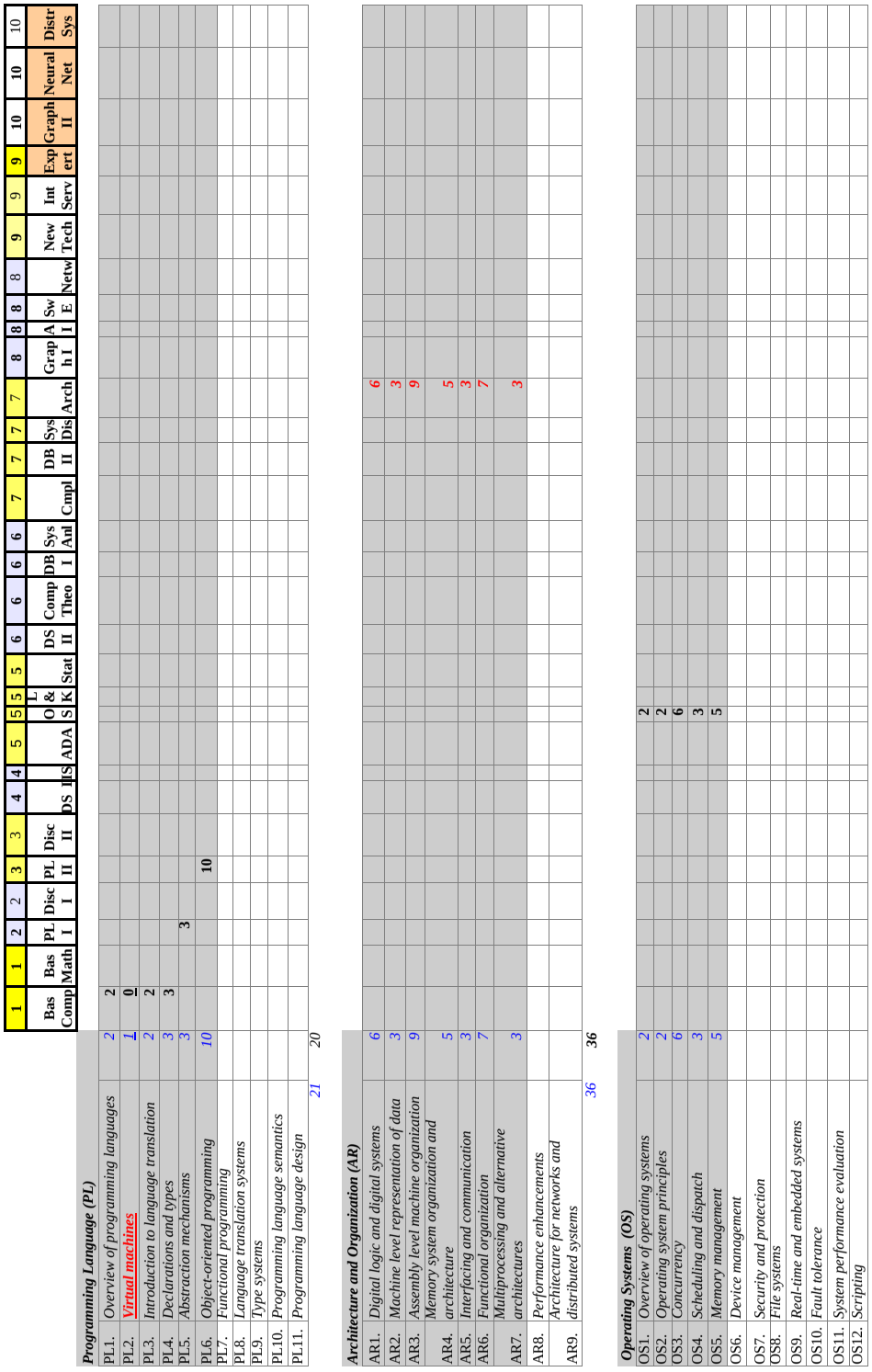


Figure 3: Figure 1. (continued)

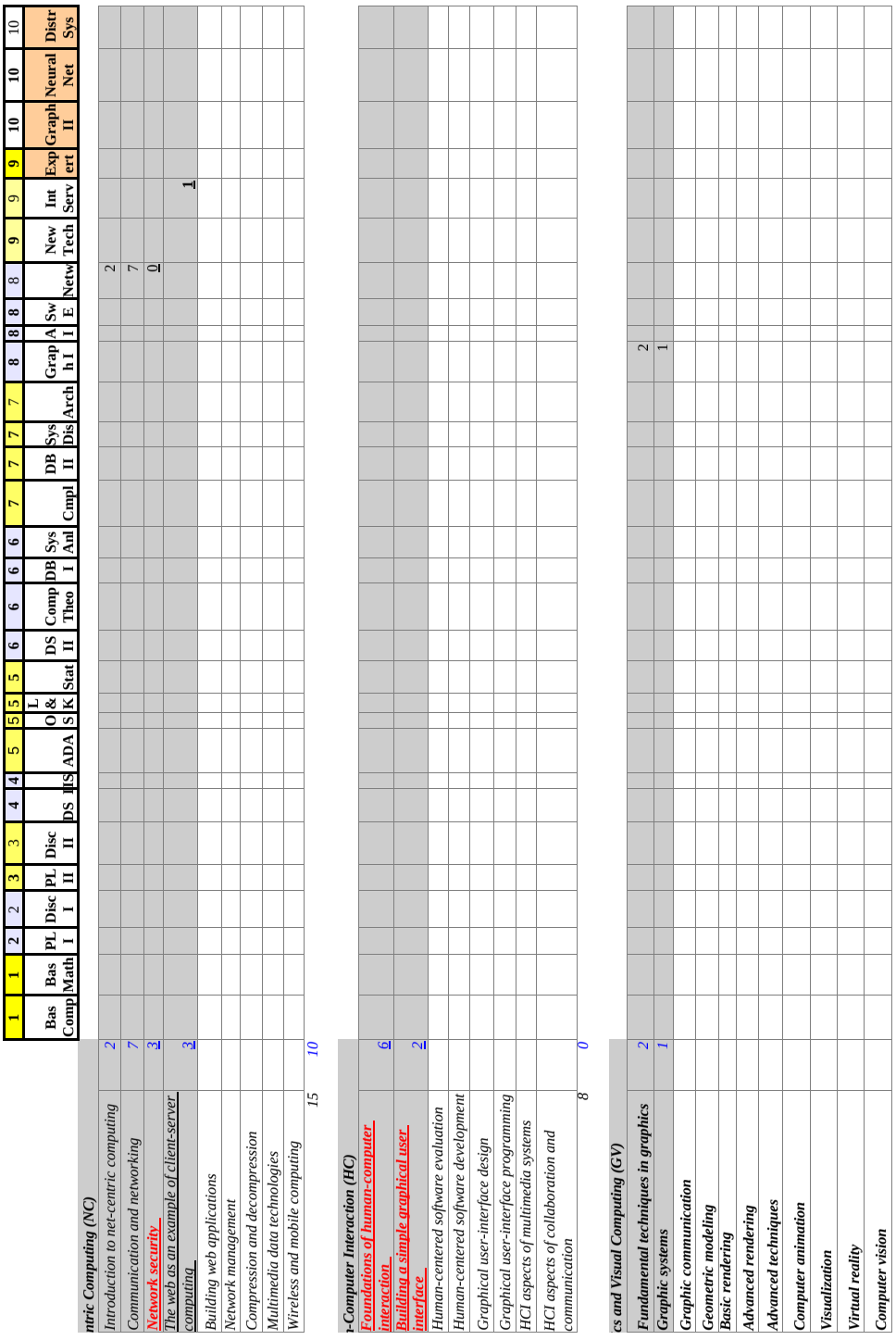


Figure 4: Figure 1. (continued)

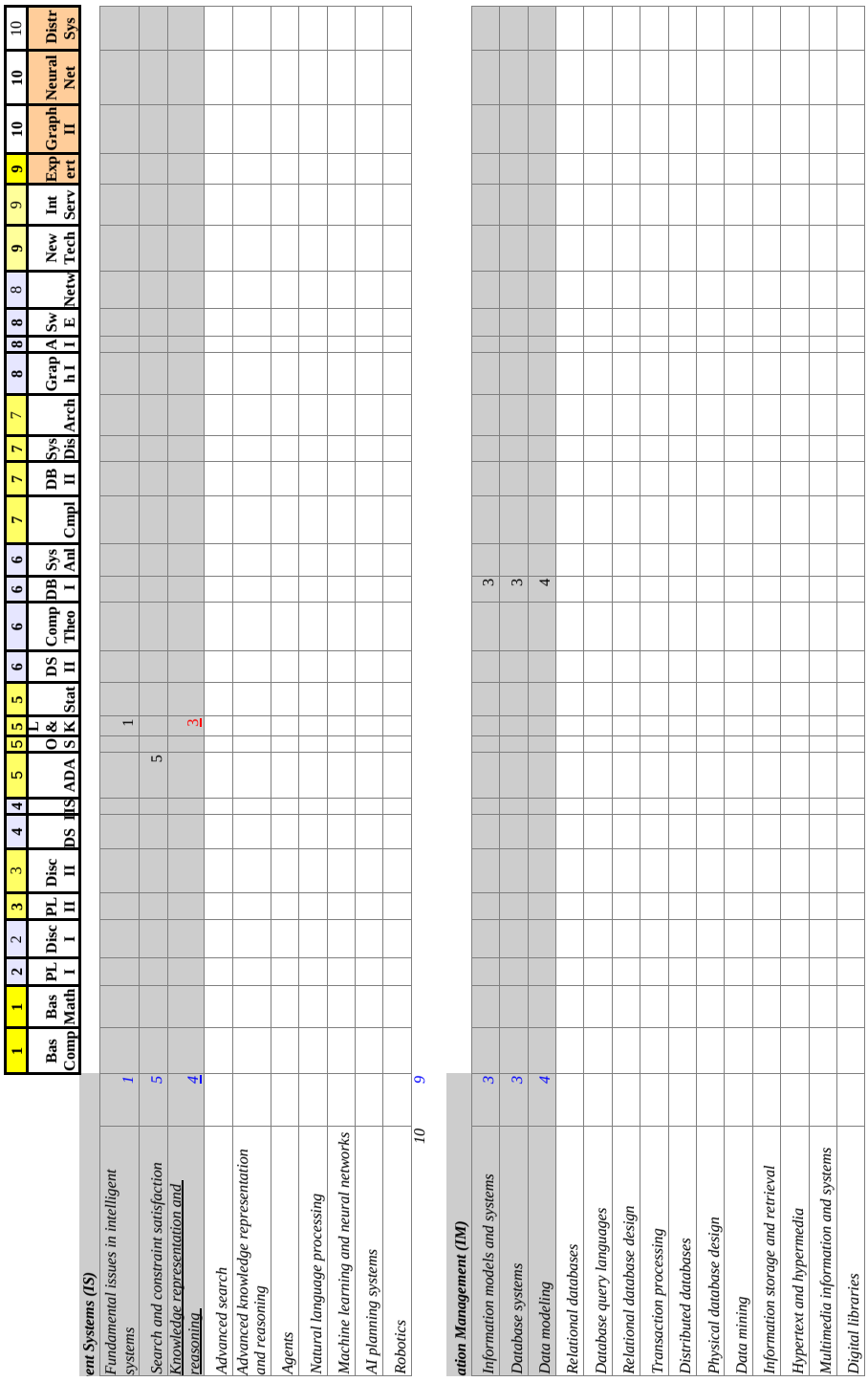


Figure 5: Introductory courses CC2001 - units for wich all topics must be covered.

			Introductory Courses								Intermediate Courses																				
		Semester	1	1	2	2	3	3	4	4	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	9	9	10
		Course USP	Bas Comp	Bas Math	PL I	Disc I	PL II	Disc II	DS I	IS	ADA	SO	L & K	DS II	Comp Theo	DB I	Sys Anl	Cmpl	DB II	Sys Dis	Arch	Graph I	AI	Sw E	Netw	New Tech	Int Serv	Graph II
	Knowledge Units for which all topics must be covered	***Core Hours***																													
DS1.	*Functions, relations, and sets*	*6*				*6*																									
DS2.	*Basic logic*	*10*		*10*																											
DS4.	*Basics of counting*	*5*						*5*																							
DS6.	***Discrete probability***	***6***				*4*																									
PF1.	*Fundamental programming constructs*	*9*	*7*		*2*																										
PF4.	*Recursion*	*5*									*5*																				
PL1.	*Overview of programming languages*	*2*	*2*																												
PL2.	***Virtual machines***	***1***					***0***																								
PL4.	*Declarations and types*	*3*	*3*																												
PL5.	*Abstraction mechanisms*	*3*	*3*																												
	Total core-hours for which all topics must be covered	***50***	*42*	84%																											

** For this analysis, only a sub-set of courses of the UCSP curriculum has been considered. The analyzed courses are those that keep a close relation with the units showed in the Body of Knowledge in Appendix B.*

Figure 6: Introductory courses CC2001 - units for which a subset of topics must be covered.

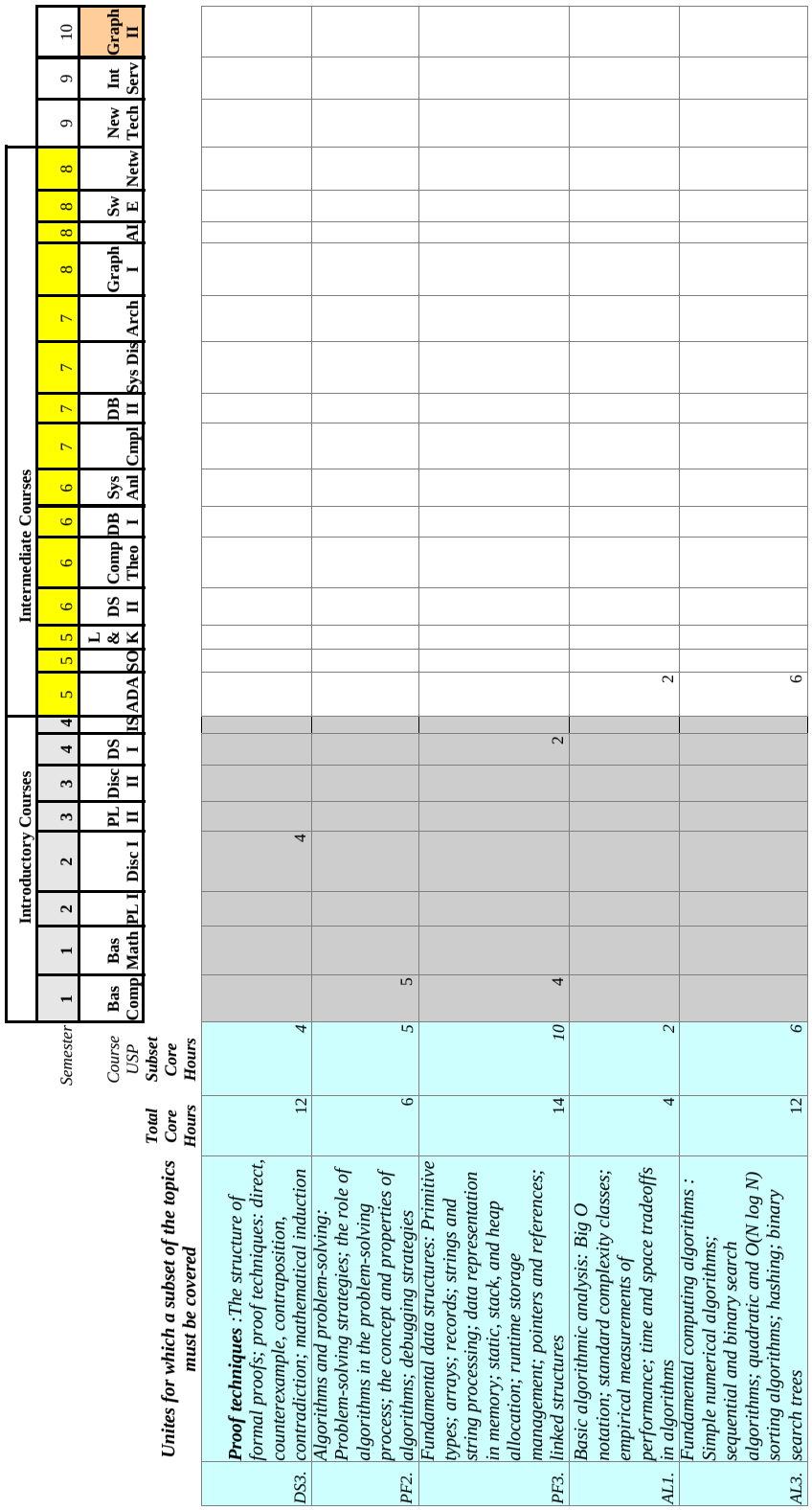


Figure 7: Figure 3. (continued)

				Introductory Courses								Intermediate Courses																				
				1	1	2	2	3	3	4	4	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	9	9	10			
				Bas Comp	Bas Math	PL I	Disc I	PL II	Disc II	DS I	IS	ADA	SO	L & K	DS II	Comp Theo	DB I	Sys Anl	Cmpl	DB II	Sys Dis	Arch	Graph I	AI	Sw E	Netw	New Tech	Int Serv	Graph II			
PL6.	*Object-oriented programming: Object-oriented design; encapsulation and information-hiding; separation of behavior and implementation; classes, subclasses, and inheritance; polymorphism; class hierarchies*	10	*8*				**8**																									
AR1.	*Digital logic and digital systems : Logic gates; logic expressions*		*1*																			1										
SE1.	*Software design :Fundamental design concepts and principles; object-oriented analysis and design; design for reuse*	8	*4*																		4											
SE2.	*Using APIs: API programming; class browsers and related tools; programming by example; debugging in the API environment*	5	*4*																		0											
SE3.	*Software tools and environments: Programming environments; testing tools*	3	*1*																		0											
SE5.	*Software requirements and specifications : Importance of specification in the software process*	4	*1*															1														
SE6.	*Software validation:Testing fundamentals; test case generation*	3	*1*																						1							
	Total core-hours for which a subset of the topics must be covered		**47**	23			48.94%																									

		Intermediate Courses																						Advanced Course						
Semester		4	4	4	5	5	5	5	6	6	6	6	7	7	7	7	7	8	8	8	8	8	9	9	9	9	10	10	10
Course UCSP		DS	ISI	Num Meth	ADA	OS	L & K	Stat	DS II	Comp Theo	DB I	Sys Anl	Cmpl	DB II	Sys Dis	Arch	Prog Math	Graph I	AI	Sw E	Netw	Stoch	New Tech	Int Serv	Simulation	Expert	Graph II	Neural Net	Distr Sys
ms and Complexity (AL)	***Elective Hrs***																												
Basic algorithmic analysis																													
Algorithmic strategies																													
Fundamental computing algorithms																													
Distributed algorithms																													
Basic computability																													
The complexity classes P and NP	*12*				6																								
Automata theory	***12***									12																			
Advanced algorithmic analysis	*12*																												
Cryptographic algorithms	*12*																												
Geometric algorithms	*12*																												
Parallel algorithms	*12*																												
ıming Language (PL)																													
Overview of programming languages																													
Virtual machines																													
Introduction to language translation																													
Declarations and types																													
Abstraction mechanisms																													
Object-oriented programming																													
Functional programming	*12*																												
Language translation systems	***12***									12																			
Type systems	*12*																												
Programming language semantics	*12*																												
Programming language design	*12*																												

** For this analysis, only a subset of UCSP courses showed in Appendix A has been considered. The analyzed courses are those that keep a close relation with the units presented in the Body of Knowledge in Appendix B.*

** The assignment of 12 elective-hours implies that a whole advanced-level course is dedicated to that unit*

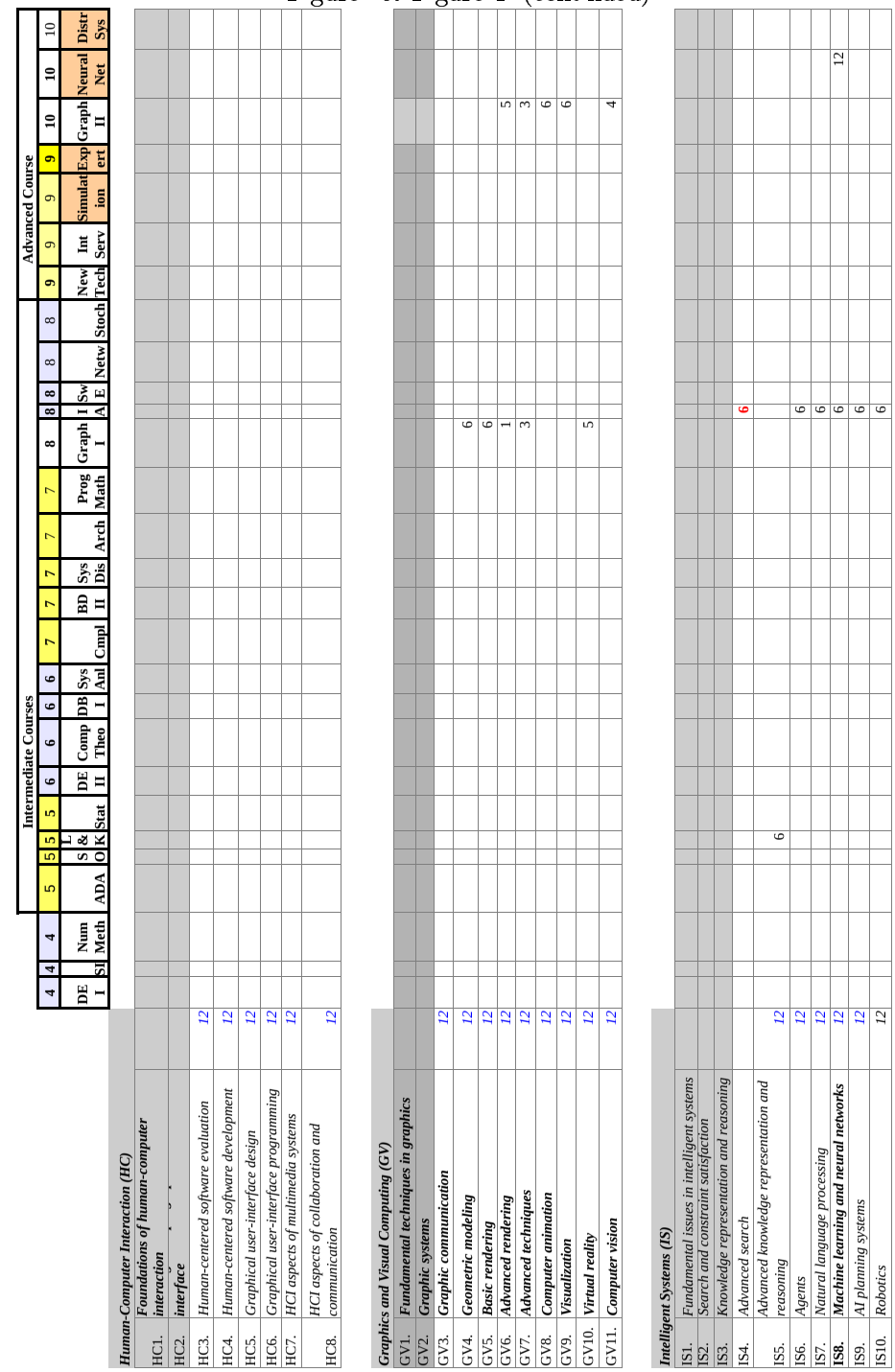
** The assignment of 6 elective-hours implies that elective hours are dedicate to that unit within some other course*

Figure 8: Advanced courses and UCSP curriculum.

			Intermediate Courses																							Advanced Course						
			4	4	4	5	5	5	5	6	6	6	6	7	7	7	7	7	8	8	8	8	8	9	9	9	9	10	10	10
			DE I	SI	Num Meth	ADA	SO	& K	Stat	DE II	Comp Theo	DB I	Sys Anl	Cmpl	BD II	Sys Dis	Arch	Prog Math	Graph I	I A	Sw E	Netw	Stoch	New Tech	Int Serv	Simulation	Expert	Graph II	Neural Net	Distr Sys
Architecture and Organization (AR)																																
AR1.	*Digital logic and digital systems*																															
AR2.	*Machine level representation of data*																															
AR3.	*Assembly level machine organization*																															
AR4.	*Memory system organization and architecture*																															
AR5.	*Interfacing and communication*																															
AR6.	*Functional organization*																															
AR7.	*Multiprocessing and alternative architectures*																															
AR8.	*Performance enhancements*	*12*																														
AR9.	*Architecture for networks and distributed systems*	*12*																														
Operating Systems (OS)																																
OS1.	*Overview of operating systems*																															
OS2.	*Operating system principles*																															
OS3.	*Concurrency*																															
OS4.	*Scheduling and dispatch*																															
OS5.	*Memory management*																															
OS6.	*Device management*	*12*																														
OS7.	*Security and protection*	*12*																														
OS8.	*File systems*	*12*																														
OS9.	*Real-time and embedded systems*	*12*																														
OS10.	*Fault tolerance*	*12*																														
OS11.	*System performance evaluation*	*12*																														
OS12.	*Scripting*	*12*																														
Net-Centric Computing (NC)																																
NC1.	*Introduction to net-centric computing*																															
NC2.	*Communication and networking*																															
NC3.	***Network security***																															
NC4.	*The web as an example of client-server computing*																															
NC5.	***Building web applications***	***12***																							6							
NC6.	***Network management***	***12***																							6							
NC7.	*Compression and decompression*	*12*																														
NC8.	*Multimedia data technologies*	*12*																														
NC9.	*Wireless and mobile computing*	*12*																														

Figure 9: Figure 4. (continued)

Figure 10: Figure 4. (continued)



		Intermediate Courses																Advanced Course									
		4	4	4	5	5	5	6	6	6	6	7	7	7	7	7	8	8	8	8	8	9	9	9	9	10	10
		DE I	SI	Num Meth	ADA	S & O	L & K	Stat	DE II	Comp Theo	DB I	Sys Anl	Cmpl	BD II	Sys Dis	Arch	Prog Math	Graph I	AI	Sw E	Netw	Stoch	New Tech	Int Serv	Simulation	Expert	Graph II	Neural Net	Distr Sys
Human-Computer Interaction (HC)																												
	Foundations of human-computer interaction																											
HC1.																												
HC2.	***interface***																											
HC3.	*Human-centered software evaluation*	12																										
HC4.	*Human-centered software development*	12																										
HC5.	*Graphical user-interface design*	12																										
HC6.	*Graphical user-interface programming*	12																										
HC7.	*HCI aspects of multimedia systems*	12																										
HC8.	*HCI aspects of collaboration and communication*	12																										
Graphics and Visual Computing (GV)																												
GV1.	***Fundamental techniques in graphics***																											
GV2.	***Graphic systems***																											
GV3.	***Graphic communication***	12																										
GV4.	***Geometric modeling***	12																6										
GV5.	***Basic rendering***	12																6										
GV6.	***Advanced rendering***	12																1								5		
GV7.	***Advanced techniques***	12																3								3		
GV8.	***Computer animation***	12																								6		
GV9.	***Visualization***	12																								6		
GV10.	***Virtual reality***	12																5										
GV11.	***Computer vision***	12																								4		
Intelligent Systems (IS)																												
IS1.	*Fundamental issues in intelligent systems*																											
IS2.	*Search and constraint satisfaction*																											
IS3.	*Knowledge representation and reasoning*																											
IS4.	*Advanced search*																		6									
IS5.	*Advanced knowledge representation and reasoning*	12				6																						
IS6.	*Agents*	12																		6								
IS7.	*Natural language processing*	12																		6								
IS8.	***Machine learning and neural networks***	12																		6							12	
IS9.	*AI planning systems*	12																		6								
IS10.	*Robotics*	12																		6								

			Intermediate Courses																						Advanced Course							
			4	4	4	5	5	5	5	6	6	6	6	7	7	7	7	7	8	8	8	8	8	9	9	9	9	10	10	10	
			DE I	SI	Num Meth	ADA	SO	L & K	Stat	DE II	Comp Theo	DB I	Sys Anl	Cmpl	BD II	Sys Dis	Arch	Prog Math	Graph I	IA	Sw E	Netw	Stoch	New Tech	Int Serv	Simulation	Expert	Graph II	Neural Net	Distr Sys	
Information Management (IM)																															
IM1.	*Information models and systems*																														
IM2.	*Database systems*																														
IM3.	*Data modeling*																														
IM4.	*Relational databases*	*12*										6																			
IM5.	*Database query languages*	*12*										6																			
IM6.	*Relational database design*	*12*										6																			
IM7.	*Transaction processing*	*12*										6																			
IM8.	*Distributed databases*	*12*													6																
IM9.	*Physical database design*	*12*										6																			
IM10.	*Data mining*	*12*																						6							
IM11.	*Information storage and retrieval*	*12*																													
IM12.	*Hypertext and hypermedia*	*12*																													
IM13.	*Multimedia information and systems*	*12*																													
IM14.	*Digital libraries*	*12*																													
Social and Professional Issues (SP)																															
SP1.	*History of computing*																														
SP2.	*Social context of computing*																														
SP3.	*Methods and tools of analysis*																														
SP4.	*Professional and ethical responsibilities*																														
SP5.	*Risks and liabilities of computer-based systems*																														
SP6.	*Intellectual property*																														
SP7.	*Privacy and civil liberties*																														
SP8.	*Computer crime*	*12*																													
SP9.	*Economic issues in computing*	*12*																													
SP10.	*Philosophical frameworks*	*12*																													

Figure 11: Figure 4. (continued)

Figure 12: Figure 4. (continued)

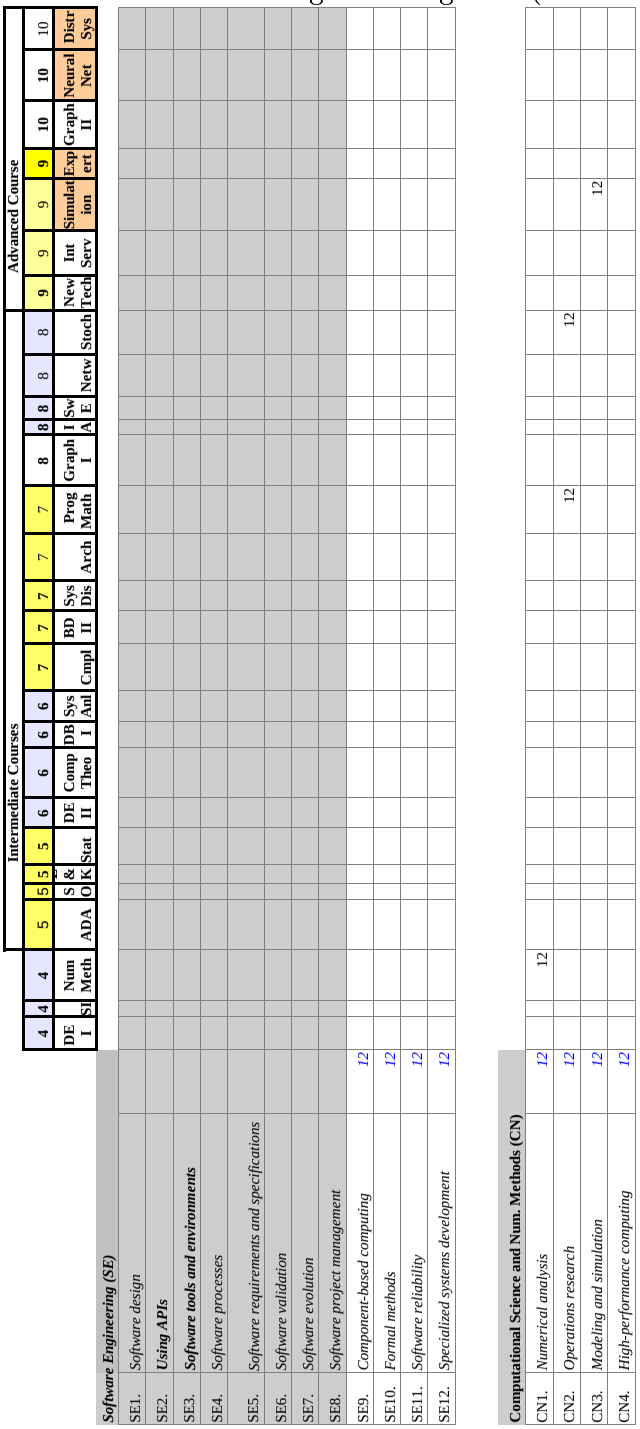


Figure 13: IS2002 courses and UCSP curriculum.

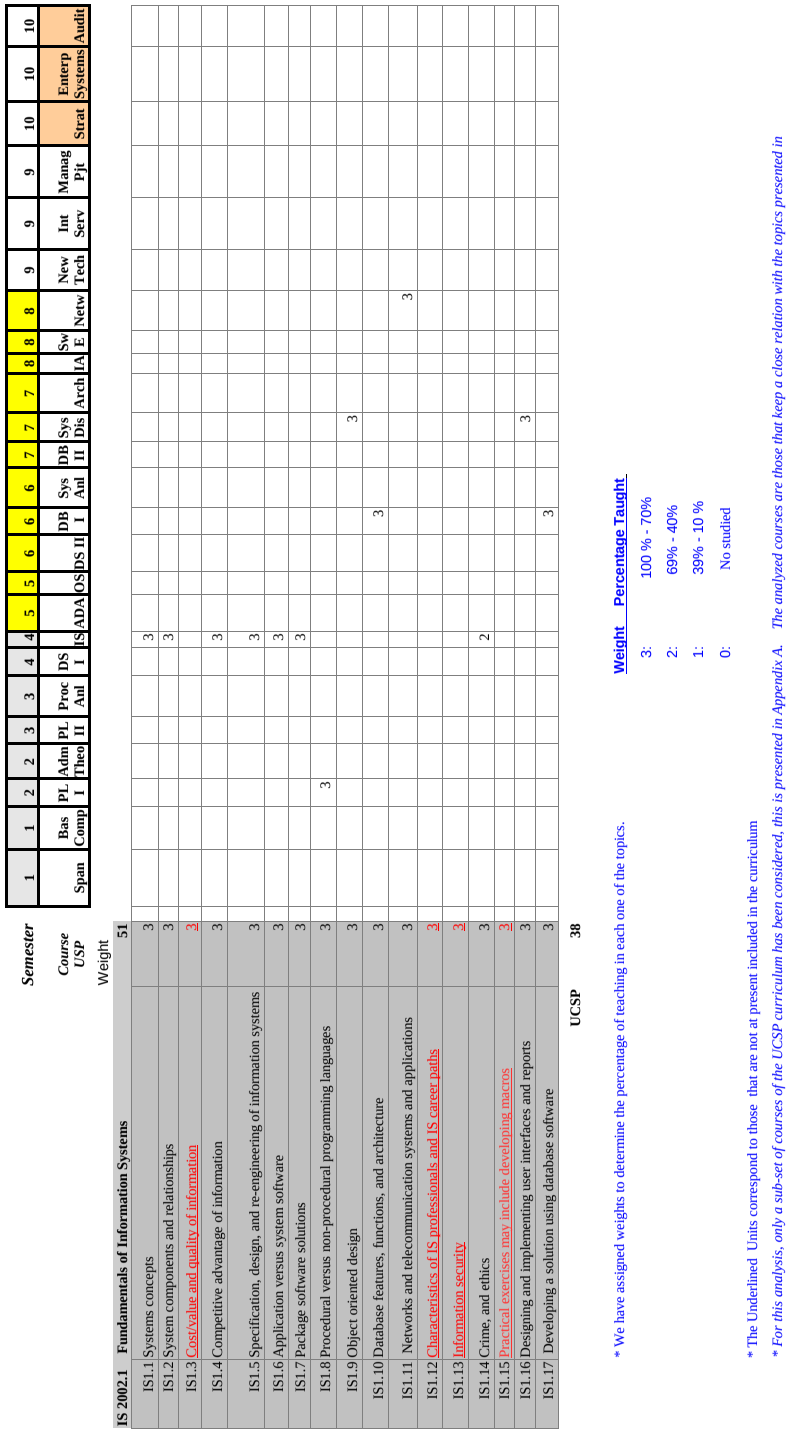


Figure 14: Figure 5.(continued)

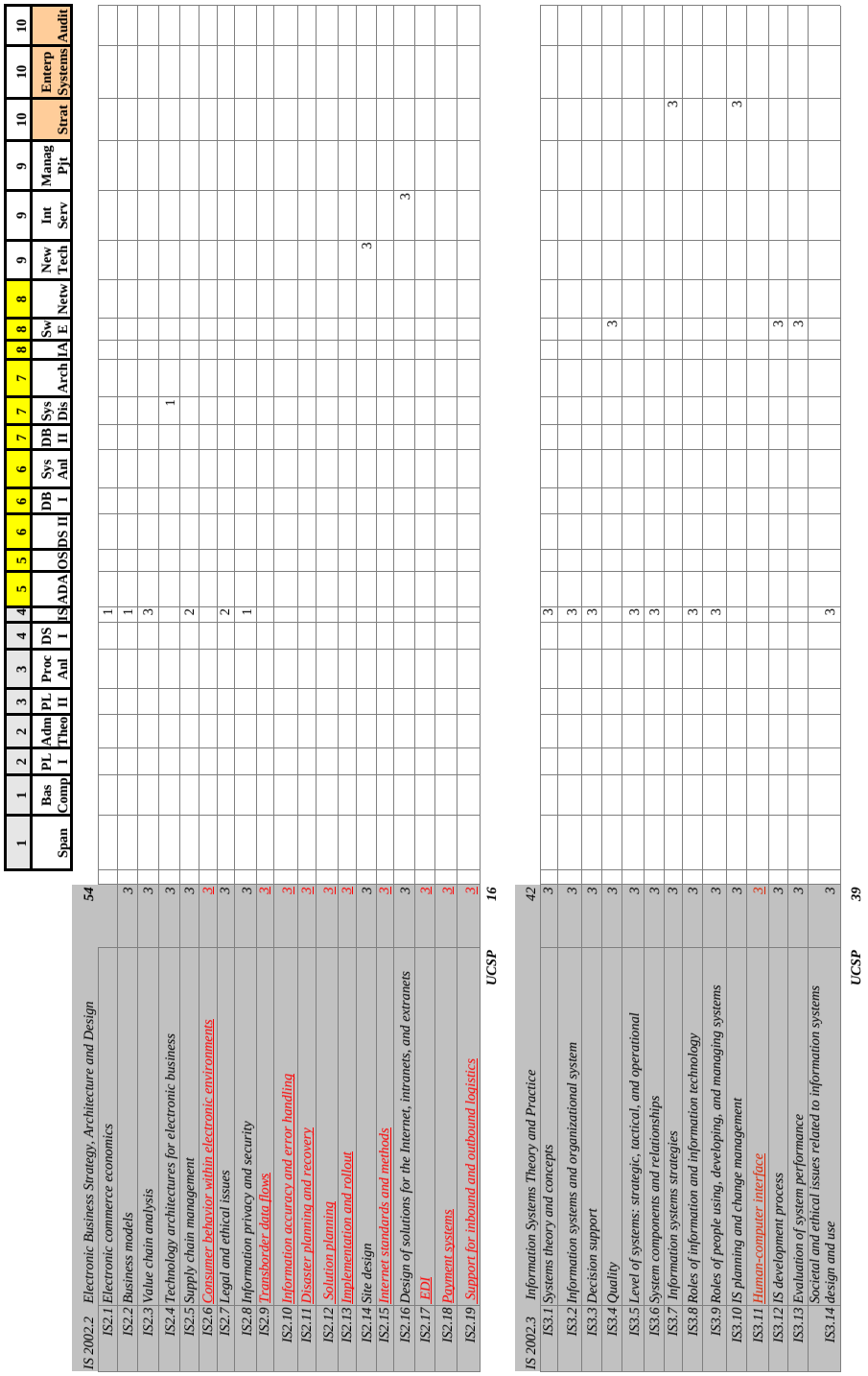


Figure 15: Figure 5.(continued)

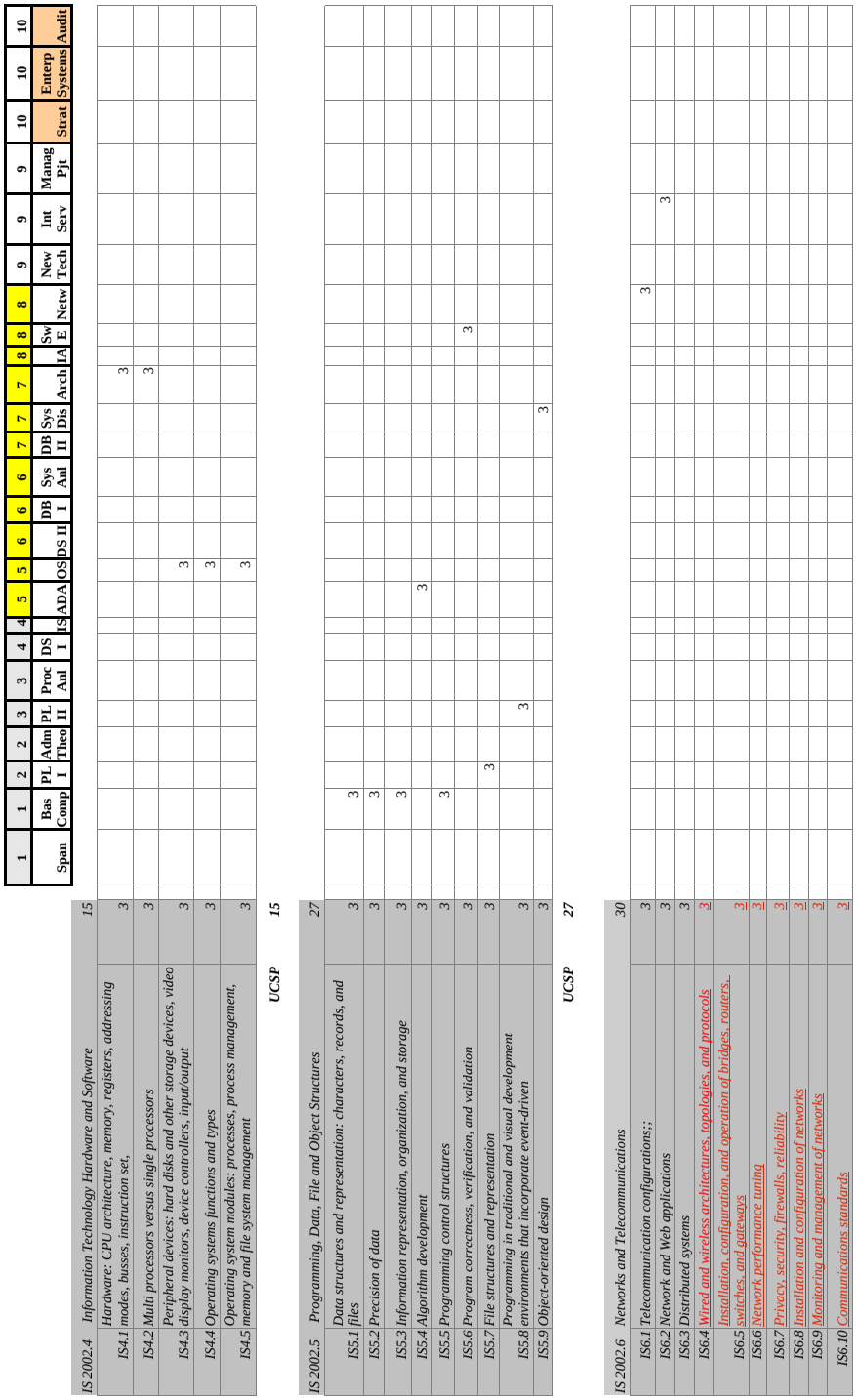


Figure 16: Figure 5.(continued)

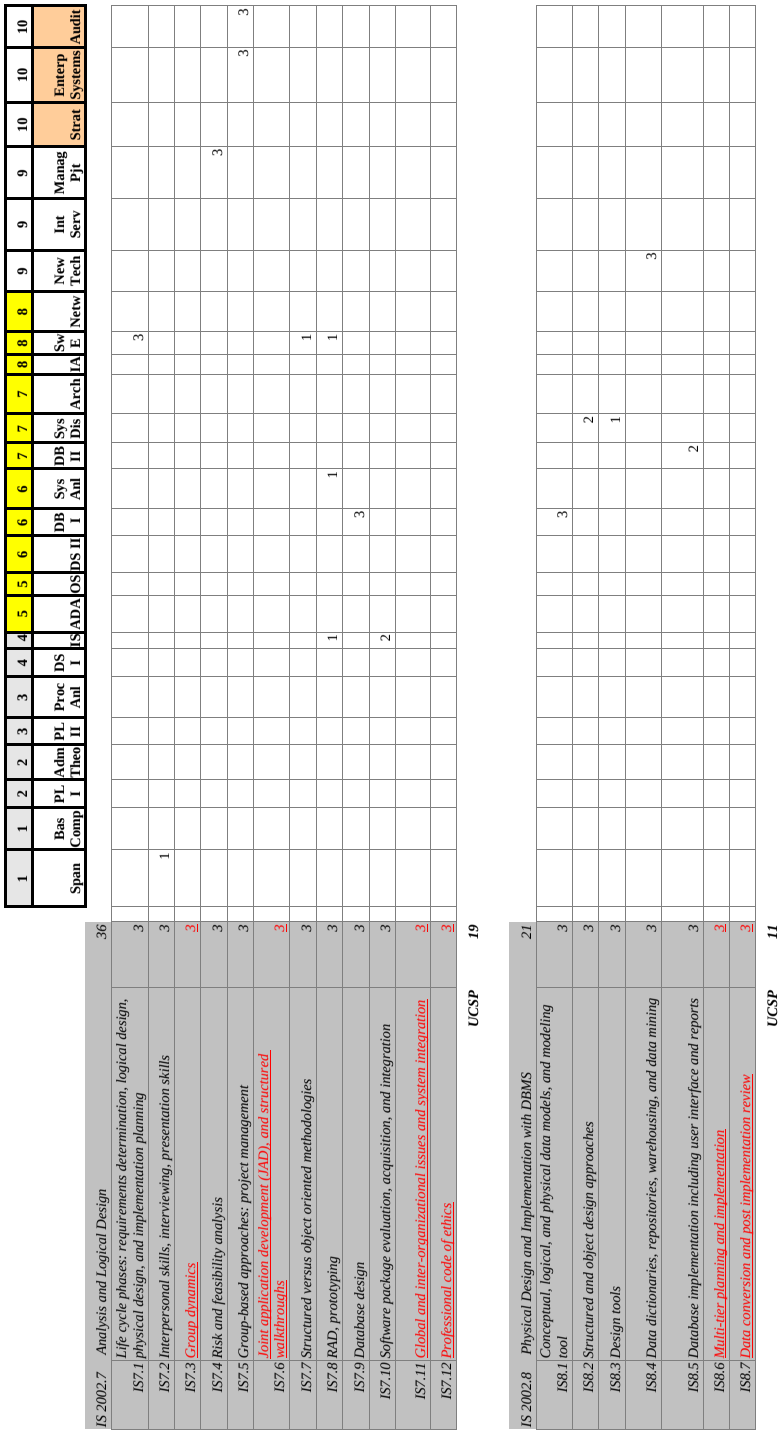
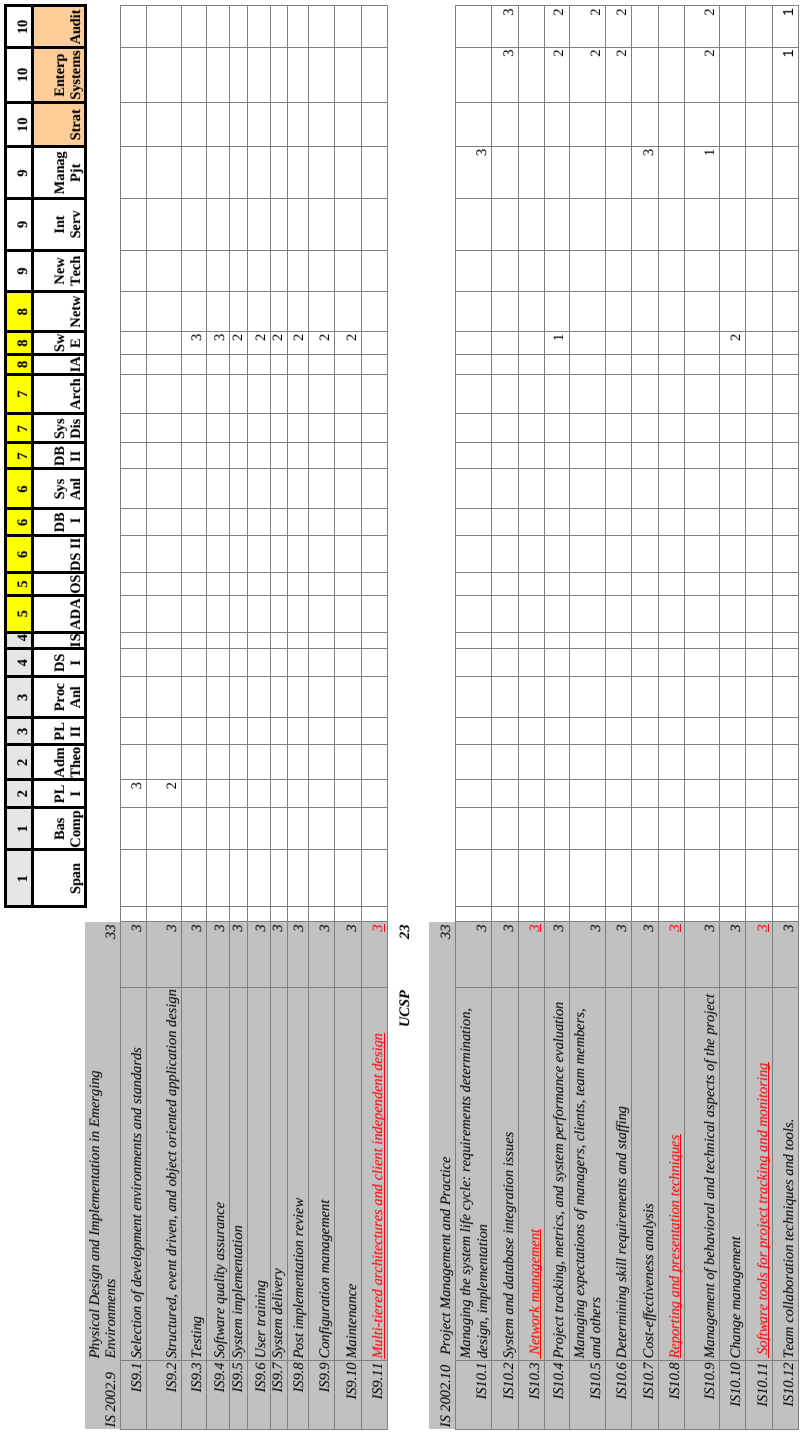


Figure 17: Figure 5.(continued)



		Span	1 Bas Comp	2 PL I	2 Adm Theo	3 PL II	3 Proc Anl	4 DS I	4 IS	5 ADA	5 OS	6 DS II	6 DB I	6 Sys Anl	7 DB II	7 Sys Dis	7 Arch	8 IA	8 Sw E	8 Netw	9 New Tech	9 Int Serv	9 Manag Pjt	10 Strat	10 Enterp Systems	10 Audit
IS 2002.9	*Physical Design and Implementation in Emerging Environments*	33																								
IS9.1	*Selection of development environments and standards*	3		3																						
IS9.2	*Structured, event driven, and object oriented application design*	3			2																					
IS9.3	*Testing*	3																	3							
IS9.4	*Software quality assurance*	3																	3							
IS9.5	*System implementation*	3																	2							
IS9.6	*User training*	3																	2							
IS9.7	*System delivery*	3																	2							
IS9.8	*Post implementation review*	3																	2							
IS9.9	*Configuration management*	3																	2							
IS9.10	*Maintenance*	3																	2							
IS9.11	*Multi-tiered architectures and client independent design*	3																								
		UCSP	***23***																							
IS 2002.10	*Project Management and Practice*	33																								
IS10.1	*Managing the system life cycle: requirements determination, design, implementation*	3																					3			
IS10.2	*System and database integration issues*	3																							3	3
IS10.3	*Network management*	3																								
IS10.4	*Project tracking, metrics, and system performance evaluation*	3																	1						2	2
IS10.5	*Managing expectations of managers, clients, team members, and others*	3																							2	2
IS10.6	*Determining skill requirements and staffing*	3																							2	2
IS10.7	*Cost-effectiveness analysis*	3																					3			
IS10.8	*Reporting and presentation techniques*	3																								
IS10.9	*Management of behavioral and technical aspects of the project*	3																					1		2	2
IS10.10	*Change management*	3																	2							
IS10.11	*Software tools for project tracking and monitoring*	3																								
IS10.12	*Team collaboration techniques and tools.*	3																							1	1
		UCSP	***22***																							

TOTAL IS2002 **342**

TOTAL UCSP 216 **63.16%**

H Detail of the Missing Units

H.1 Body of Knowledge Core-Units Missing

Discrete probability: finite probability space, probability measure, events, conditional probability, independence, Bayes' rule, integer random variables, expectation.

(PF5) Event-driven programming: event-handling methods, event propagation, exception handling.

(PL2) Virtual Machines: the concept of a virtual machine, hierarchy of virtual machines, intermediate languages security issues arising from running code on an alien machine.

(AL4) Distributed Algorithms: consensus and election, termination detection, fault tolerance, stabilisation.

(NC4) The web as an example of client-server computing: server-side programs, common gateway interface (CGI) programs, client-side scripts, applets, web servers, file management, capabilities of common server architectures web protocols, support tools for web site creation and web management,developing internet information servers

NC3) Network security: fundamentals of cryptography, secret-key algorithms, public-key algorithms, authentication protocols, digital signatures.

(HC1) Foundations of human-computer interaction: contexts for HCI (tools, web hypermedia, communication), human-centered development and evaluation, human performance models (perception, movement, and cognition), principles of good design and good designers, engineering tradeoffs, introduction to usability testing.

(HC2) Building a simple graphical user interface: principles of graphical user interfaces (GUIs), GUI toolkits.

(SE2) Using APIs: API programming, class browsers and related tools, programming by example, debugging in the API environment, introduction to component-based computing.

(SE3) Software tools and environments: programming environments, requirements analysis and design modeling tools, testing tools, configuration management tools, tool integration mechanisms.

(SP1) History of computing: history of computer hardware, software, networking; pioneers of computing.

(SP2) Social context of computing: social implications of computing and networked communication; growth of, control of, and access to the Internet; gender-related issues; international issues.

(SP3) Methods and tools of analysis: making and evaluating ethical arguments, identifying and evaluating ethical choices, understanding the social context of design, identifying assumptions and values.

(SP4) Professional and ethical responsibilities: various forms of professional credentialing and the advantages and disadvantages, 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.

(SP5) Risk and liabilities of computer-based systems: historical examples of software risks, implications of software complexity, risk assessment and management.

(SP6) Intellectual property: foundations of intellectual property; copyrights, patents, and trade secrets; software piracy; software patents; transnational issues concerning intellectual property.

(SP7) Privacy and civil liberties: ethical and legal basis for privacy protection, privacy implications of massive database systems, technological strategies for privacy protection, freedom of expression in cyberspace, international and intercultural implications.

H.2 Introductory Core-Units Missing

(PL2) Virtual Machines: the concept of a virtual machine, hierarchy of virtual machines, intermediate languages security issues arising from running code on an alien machine.

(DS6) Discrete probability: finite probability space, probability measure, events, conditional probability, independence, Bayes' rule, integer random variables, expectation.

(PF4) Recursion: concepts, recursive mathematical functions, simple recursive procedures, divide-and-conquer strategies, recursive backtracking, implementation of recursion.

(PF3) Fundamental Data Structures: primitive types; arrays; records; strings and string processing; data representation in memory; static, stack, and heap allocation; runtime storage management; pointers and references; linked structures; implementation strategies for stacks, queues, and hash tables; implementation strategies for graphs and trees; strategies for choosing the right data structure.

(AL1) Basic algorithmic analysis: 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.

(AL3) Fundamental computing algorithms: 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.

(AR1) Digital logic and digital systems: overview and history of computer architecture; fundamental building blocks (logic gates, flip-flops, counters, registers, PLA); logic expressions, minimization, sum of product forms; register transfer notation; physical considerations (gate delays, fan-in, fan-out).

(SE1) Software Design: fundamental design concepts and principles, design patterns, software architecture, structured design, object-oriented analysis and design, component-level design, design for reuse.

(SE5) Software requirements and specifications: requirements elicitation, requirements analysis modeling techniques, functional and nonfunctional requirements, prototyping, basic concepts of formal specification techniques.

(SE6) Software validation: validation planning; testing fundamentals; including test plan creation and test case generation; black-box and white-box testing techniques; unit, integration, validation, and system testing; object-oriented testing; inspections.

(SE2) Using API's: API programming, class browsers and related tools, programming by example, debugging in the API environment, introduction to component-based computing.

(SE3) Software tools and environments: programming environments, requirements analysis and design modeling tools, testing tools, configuration management tools, tool integration mechanisms.