Winter 12-19-2001

Navigating The Leading Edge: A Prototype Curriculum for Software Systems Management

Gregory Uulferts

Dan Shoemaker

Antonio Drommi

NAVIGATING THE LEADING EDGE: A PROTOTYPE CURRICULUM FOR SOFTWARE SYSTEMS MANAGEMENT

Gregory Uulferts
Email : ulfertgw@udmercy.edu
Dan Shoemaker
Email: Shoemaker@udmercy.edu
Antonio Drommi
Email: Drommia@udmercy.edu
College Of Business Administration
University Of Detroit Mercy
P.O. Box 19900
Detroit, MI 48219-0900
Phone No. (313) 993-1200

ABSTRACT

This article presents a meaningful and advantageous new direction for information technology education, embodying principles for systematically optimizing the functioning of the business.

Our curriculum was built on the thesis that every aspect of software systems management can be understood and described as a component of four universal, highly correlated behaviors: *abstraction, product creation, product verification and validation,* and process *optimization*. Given this, our model curriculum was structured to provide the maximum exposure to current best practice in six thematic areas, which taken together as an integrated set, makes-up the attributes that differentiate us from the other computer disciplines:

Abstraction*: understanding and description of the problem space*
Design*: models for framing artifact to meet criteria 3, 4, 5, and 6*
Process Engineering*: application of large models such as IEEE 12207*
Organizational Control Systems*: SQA and configuration management*
Evaluation with Measurement*: with an emphasis on testing and metrics*
Construction: *professional programming languages with emphasis on reusability*

Our teaching strategy approaches this as a hierarchy of similar activities. **In every course** we require the student to define and implement all three interfaces and be able to clearly communicate this as a logically consistent model before working out the details of the solution. The focus of all understanding is top-down from the information interface. Our curriculum centers on the application of software engineering standards (such as those promulgated by IEEE) and the software process improvement, or quality standards (such as those promulgated by SEI and ISO) under the assumption that this embodies the "common body

of knowledge and state of best practice" in software production and management.

The practical realization of this is an integration of the large subject areas of: software engineering (methods, models and criteria), process and product quality management (software quality assurance and metrics), software project management (work decomposition, planning, sizing and estimating), and software configuration management. Reconciliation of project and configuration management is accomplished by cross-referencing the problems, tools, notations and solutions (through explicit identification, authorization and validation procedures). As a side agenda, we have also stressed the need for re-engineering the vast number of software products currently on the shelves. This model plus germane simulated real-world experience introduces all of the relevant principles to the student within the (currently understood) framework. It allows them to develop and internalize their own comprehensive understanding and formulate a personal model of the disciplinary body of knowledge.

CURRICULUM
Software Systems Management

Foundation
CIS 501 Introduction to Information Systems – Visual Basic
Core
CIS 505 Project Management
CIS 510 Object Oriented Software Development
CIS 520 Software Requirements
CIS 530 Software Quality Assurance and Testing
CIS 540 Strategic Software Process Management
Electives *(6 required)*
CIS 502 Structured Development for the Internet
CIS 503 Software System Documentation
CIS 525 Software Design and Construction
CIS 535 Metrics and Models for Software Management
CIS 543 Software Lifecycle Documentation
CIS 553 Graphical User Interface Development
CIS 554 Software Maintenance Using Cobol

CIS 555 Data Base Design
CIS 557 Networks and Network Management
CIS 558 Distributed Software Development
CIS 559 Electronic Data Interchange
CIS 560 Electronic Commerce
CIS 565 Information And Society
CIS 566 Advanced Database Issues
CIS 580 Advanced Topics in Is
CIS 589 International Software Management
CIS 590 Leadership in Assessment
CIS 591 Audit

NAVIGATING THE LEADING EDGE: A PROTOTYPE CURRICULUM FOR SOFTWARE SYSTEMS MANAGEMENT

Introduction

The discipline of software systems management is a meaningful and advantageous new direction for information technology education. That is because it is focused on two pivotal issues for such organizations, cost control and production efficiency. Accordingly the primary distinction between software systems management and traditional areas of study such as computer engineering, computer science, and software engineering, lies in the fact that the former embodies principles for systematically optimizing the functioning of the *business*, whereas the latter concentrates on the *technology* itself.

In applied terms, the basic aim of software systems management is to instill sound business practices into IT operations. Its objective is to insure that the organization's people equipment and financial resources are utilized for the maximum benefit of the business and the satisfaction of its customers. This is a particularly relevant and important subject for the software industry at this time and place because, although it has a forty-year history of leadership and innovation in the production of quality goods and services, it has yet to prove that it can operate in a cost-effective manner.
In reality all of the evidence so far indicates the contrary. It is a fact that.... *Depending on size, between 25% and 50% of all IT projects fail, where "failure" means that the project is canceled or grossly exceeds its schedule estimates* (Laker, 1998). A recent Standish Group survey of 8,000 software projects found that the average one exceeded its planned budget by 90 percent and its schedule by 120 percent (Construx, 1998). A similar study conducted by KPMG Pete Marwick found that 87% of failed projects exceeded their initial schedule estimates by 30% or more. While at the same time 56% exceeded their budget estimates by 30% or more and 45% failed to produce expected benefits.

It would be incorrect to assume that this failure was a consequence of extreme project size, or complexity. In actuality 60% of the failed projects were categorized by KPMG as small. The fact is that small projects (e.g., those that are characteristic of the average mom-and-pop business) are almost always over schedule (92%). In fact the larger, more complex projects actually did better. KPMG found that only 86% of these had problems meeting their delivery dates (which is still a pathetic statistic). Moreover, this is not a new phenomenon. A study done by the GAO, which encompassed the entire decade of the 1980s, found that fully two-thirds of the software delivered to the federal government was never used and an additional 29% was *never delivered at all*. As a result, the GAO estimated that throughout the 1980s the federal Government's bill for worthless software topped $150 billion (Quoted in Humphrey, 1994).

Now, when 95% of the software delivered to the federal government is worthless you might expect some accountability. Yet studies since then document the same problems (SEI, 1997). According to Humphrey (2000) the cause lies in the fact that it is extraordinarily difficult to manage an activity that is creative and conceptual by nature using traditional techniques. Instead, effective management relies on extensive experience. Without which, "*Inexperienced, or inadequately trained managers are noted with distressing frequency on canceled projects and projects that experience cost overruns and missed schedules. Inadequate management training is also commonly associated with the problems of low productivity, low quality, and of course, management malpractice* (Jones, 1994)." *As such, "the world is beginning to realize that it needs people at the highest levels who can combine the skills of the technician with those of the manager"* (O' Brien, 1992).

Rationale: Why Study Software Systems Management?

The software industry banks close to a trillion-dollars annually (Boehm quoted in Humphrey, 1997). Yet, with the stakes that high the manager who is . . . *"knowledgeable in the realms of new technology is a rare breed"* (O' Brien, 1992). Brynjolfsson provides a very apt synopsis of the consequences of that condition: *"Productivity is the fundamental economic measure of a technology's contribution. With this in mind, CEOs and line managers have increasingly begun to question their huge investments in computers and related technologies. While major success stories exist, so do equally impressive failures"* The economist Robert Solow astutely sums up the problem this way: "*we see computers everywhere except in the productivity statistics"* (both quoted in Brynjolfsson, 1992). Or in simple terms, a shocking few people seem to fathom the significant business implications of the technology they were spending billions of dollars to acquire. O'Brien corroborates this:

> *"It has become fashionable to talk of competitive advantage and information technology in the same breath . . . yet it is clear that the number of professionally educated (to maximize competitive advantage using technology), fully trained and*

> *experienced information technologists is small."*

Higher education clearly hasn't found any answers. In 1985, Datamation conducted an extensive survey aimed at giving academia a report card. They polled a laundry list of experts from every conceivable area of business to find out whether college training translated to success in business. Needless to say, the grades were not good. Everyone took turns bashing the products of this country's post secondary education system for their total lack of leadership, business knowledge and communication skill. This was particularly true for computer science. The comments of one Fortune 200 executive were typical. He estimated that only "5 percent of the graduates *(hired by his company)* were adequately prepared." One would think that the pressures of competition would make higher education more responsive to the demands of industry, which is the ultimate consumer of its graduates. In addition, this survey was conducted almost sixteen years ago. Perhaps like fine wine the situation improved with age? A comprehensive study of university catalogues, conducted in 1997, found that 90% of business school curricula were at best . . . *"incomplete and lagging behind the state of the art by more than ten years . . . when it came to the requirements of managing software* (Jones 1997)". More pertinent, although almost eleven years separate these two studies the same symptoms were identified: "weak, or inadequate business preparation and impractical, or out-of-date curricula."

Both studies make it clear that the key to the formulation of a successful study of the discipline of software management hinged on the capability to amalgamate "current best practice" ideas into an optimally valid and accurate model of software process functioning. The dilemma lies in the fact that this best practice is prescribed by professional standards and, *...since 1976 the Software Engineering Standards Committee of the IEEE Computer Society has developed 19 standards in the areas of terminology, requirements documentation, design documentation, user documentation, testing, verification and validation, reviews and audits. And if you include all the major national standards bodies, there are in fact more than 250 software engineering standards. (IEEE).* Besides the implication that our eminent standards bodies need to be leashed, this astonishing productivity (roughly 10 new standards a year) underlines the crucial importance of a common conceptual framework that will help educators judge the boundaries of the body of knowledge in order to know what to teach. Therefore our first efforts were focused on developing such a unified frame of reference.

Disciplinary Model: Conceptual Basis

At the theoretical nucleus of our undertaking was the belief that software systems management has not been approached at the proper *(e.g., the highest possible)* level. Instead, technology centered approaches have always been introduced piecemeal. This is unsuitable because by definition proper management must incorporate methods for handling the problem as a whole. That implies understanding and mastery of all rational principles, and methods that optimize the software process as a complete and consolidated entity set. This required getting a proper understanding of the entire range of disciplines that compose the world of computing. Figure One summarizes this.

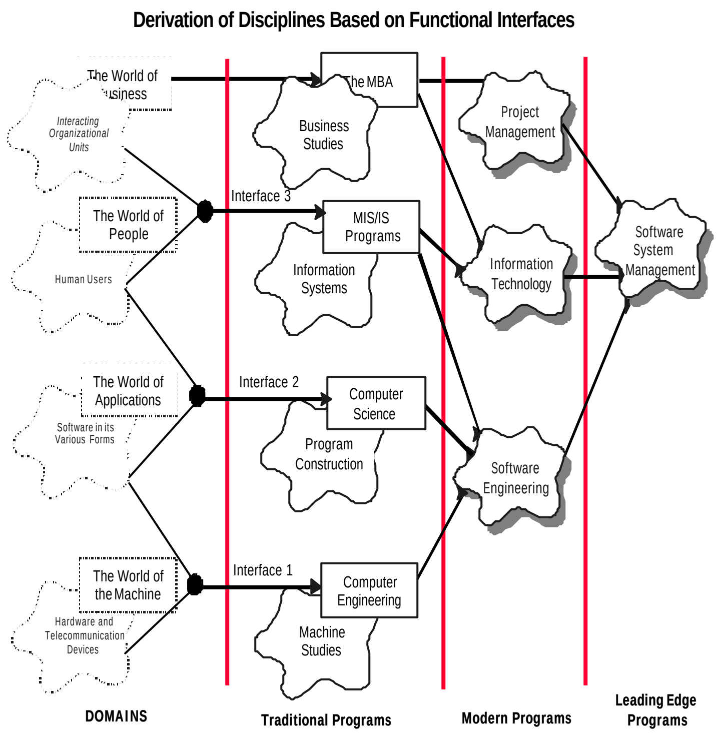

Figure 1

This diagram shows that computer systems function on three hierarchically differentiated, mutually inter-dependent interfaces. Bottom up these are the hardware/software interface (1), the human/machine interface (2), and the information/computing interface (3). At the hardware/software interface, tangible machine resources realize the functional design. Practically speaking, this translates into efficient run times and the other familiar, properties of good hardware and software architecture. At the human/machine boundary, information crosses the physical periphery and referential margin between human and machine. The tangible realization of this can be seen in effective user-friendly programs, and ergonomically designed equipment. At the information/computing interface, the organization's inherent and elemental structure is mapped into the domain of the computer. This interface views computer systems as embodying the organization's complete information infrastructure. It should also be clear that inefficiencies in either of these components (e.g., the computer system or the organizational system it interacts with) all conspire to make the total system less efficient. Optimization principles exist for all of these interfaces; the problem is that nobody sees the need to apply them uniformly across all of the interfaces in the entire system. The classic result is the 900-megahertz machine interacting with a human who processes a task every five minutes while entering three-day-old data.

Conceptual Model: A Review Of Current Approaches

In order to understand the implications of this within higher education we took a rigorous look at the basis of each of the disciplines that could be directly related to one of these interfaces. The oldest, best established, and most righteous of these is computer science. The principles embedded in this body of knowledge have been very successful; witness the advances in hardware over the past 40 years. Ideally, computer scientists focus on optimizing the software/hardware interface. It first emerged from the disciplines of electrical engineering, mathematics and logic back in the 1950's. Its strongest identification and disciplinary attachment is with the math department. A quote from the Carnegie Mellon Curriculum for Undergraduate Computer Science makes this very clear:

> *"Computer science is a mathematical discipline...so much so that the difference between computer science and mathematics is often quite hard to pin down. While both disciplines are concerned primarily with abstract structures, computer science is not simply a branch of mathematics. It relies on skills attitudes and techniques derived for mathematics, but it is concerned not so much with proofs and structures as it is with algorithms and the design and organization of structures."*

However, the next level up in the hierarchy, the human/machine interface, involves a much larger set of variables, most of them unknown and unknowable. Compounding this problem is the complexity of human behavior, which is intuitive, hence unpredictable and impossible to model mathematically. Accordingly, the central methods and principals of computer science, focusing on exact descriptions of the problem, have been of limited value in optimizing the higher (in terms of the abstraction ladder) interfaces.

Through the 1980s, partly because of the failure of traditional computer scientists to perform consistently, and effectively at the human machine interface, the computer industry itself evolved the discipline of software engineering (a useful reference date for this is the foundation of the Wang Institute of Graduate Studies, 1979, for the details see [Ardis, 1987]). This discipline takes a software/applications focus. It embodies a set of description and design principles based on engineering methodologies specifically oriented toward software development and management methodologies specifically oriented to software systems management (Ardis, 1987). Generally, at the undergraduate level the software engineering curriculum is indistinguishable from computer science. Where it differs is in the incorporation of an additional set of courses whose focus and content are delineated by common industry practices, such as software specification, design, testing and quality assurance, and software configuration management. In these courses, engineering principles shape the approach, and teamwork, instead of individuality, is stressed. Software engineering is different from computer science in two general aspects; it has an applied rather than theoretically abstract intent, and the training model is very much "world of work," instead of academic.

Practically speaking, the advantage of the software engineering approach over traditional computer science is that it supports the design, development, and operation of much larger and more complex real-time computer systems. The disadvantage is that, to be truly effective in a large, multifaceted organization, the interface has to be engineered on both sides. That is, the human system opposite the machine has to predictable and stable to some extent. So, software engineering techniques work well for complex, embedded systems in areas such as avionics and telecommunications and not at all in the low-tech, generic systems used in complex business organizations. Since most of the packaged software sold by the computer industry today falls into this second category, it seemed safe to conclude that to be universally applicable software engineering must evolve principles and techniques for engineering the system's human component. Thus came the fortuitous marriage of software engineering to information science, which is the foundation of our unique program.

Information science, which has always been viewed by the other two disciplines as more of an art than a science, began to appear on campuses in the 1970's. This discipline focuses on the development of a complete set of approaches to mesh organizational systems with computer systems. Given the practical human focus and the fact that the tools originated in the disciplines of operations research and industrial engineering, information science curricula have always tended to be based in business schools. This placement is unfortunate because curricular content focusing on the bottom two interfaces (or at least the machine part of the human/machine interface) tends to be almost completely non-existent in most of these programs.
Essentially, the information scientist defines the system as the flow and transformation of organizational data. Symbolic notational techniques can be used to discretely model system behavior on the interface. Since these tools are familiar to everybody in computing, modeling techniques such as UML or Data Flow can be drilled down to any level in description and definition of the system. The difference for the information scientist lies in where the level of application is begun. Here, the business process itself, instead of the processing of the data becomes the primary means by which interface events are defined. Since this data is independent of processing, its origination and handling can be defined external to the computer part of the system. The appropriateness of starting the design activity here should be intuitively obvious to everybody, since the computer doesn't function in a vacuum. Implicit is the requirement that the function and the computer system mesh as efficiently as possible.

The problem with information science lies in the fact that it is primarily oriented toward the business view rather than technology. So, it is traditionally a managerial rather than a technical discipline. As a result, the reward structure is geared differently. An extensive and definitive study of IT personnel carried out by Datapro Corporation, found that the three most critical predictors of success for this area were the *business criteria of,* ***corporate fit, corporate credibility*****,** and ***upward management skills***. Which indicates that, based on this industry-wide survey, the successful people working at the MIS management level don't need to be technically proficient.

A Pilot Study: Goals

As we said earlier, we differ from most approaches in that we view computer system development as an integrated activity that is wholly based on seamless abstraction of the system from the business to the operating reality. However, in an attempt to determine exactly where we fit in the spectrum of programs out there, the College of Business Administration funded (in the summer of 2000) the IT Education Baseline Project for the purpose of accomplishing the following explicit goal…

> *"The project will develop and enact a process for the systematic collection, evaluation and classification of software education programs for the purpose of developing explicit descriptive understanding of the various categories of programs in this area".*

In essence, this amounted to an attempt to characterize the current state of the art in terms of computer and software curricula and the pedagogy that supports these. Given that, the results were expected to serve to define a coherent and concrete characterization of the diffusion of innovation in the various types of computer, or software education programs in the US. It was assumed that this characterization could then be used for long-term curricular assessment and planning by any educational agency interested in technology transfer for the profession.

A Pilot Study: Methodology

This study was conducted in two phases. **The first phase** involved the identification of an appropriate set of institutions to conduct rigorous on site interviews at. We felt that this was necessary because, notwithstanding the presence of Ford's report *(SEI-94-TR-11)* there is very little consistent agreement about what constitutes a traditional computer science, or MIS versus a software engineering program. We particularly sought to identify programs at Carnegie One institutions and Jesuit schools (given our own foundation)

Following satisfactory completion of that phase of the project and the selection of a sample of 22 institutions, on site interviews were conducted to detail the pedagogy employed at each. Obviously we could have read each institution's course catalogue to learn what they offered. However, we felt that these interviews were necessary because the intent and even the details of implementation are never clear in a document such as a catalogue. Accordingly, we sought to acquire (at a minimum) syllabi, assignments and (hopefully) examples of student artifacts themselves. This process consumed the period from June 2000, to September 2000 and featured intensive visits to all 22 institutions. Following this data-gathering phase we began to perform a **comparative content analysis** of the courses, artifacts and assignments obtained. Since the intention of this project was to identify and characterize various program types as well as map the diffusion of innovation throughout the study of computing.

This report presents the preliminary findings of this initial analysis. A couple of things must be kept in mind as you read this. Although the survey was nationwide it is impossible to say for certain that all institutions that offer software engineering programs (labeled "modern" in the graph) were considered when then initial sample group was drawn. That is because the discipline itself is so loosely defined that there is simply no common registry of such programs. We used information provided by the Software Engineering Institute (SEI), the United States Military and IEEE to distinguish institutions that satisfied the general functional definition of a software engineering education program based on those three bodies' view of the world. The sample that was drawn from this lengthy list was composed of a range of institutions from research universities down to small regional colleges.

Preliminary Findings: The Software Engineering Education Baseline

Using the mass of detailed data that we had collected at each institution we sought to formulate a definitive characterization of program types based on our interface model and the assumptions that underlay our own disciplinary approach. From this we identified three distinctive types of programs. We labeled the first "**Traditional"** (e.g., programs that focus primarily on one interface such as computer engineering, computer science and MIS). We labeled the second **"Modern"** (e.g., programs that span two interfaces such as software engineering). We labeled the third type **"Leading Edge"** (e.g., programs that embody all interfaces (e.g., the software systems management approach). Given all of this we performed a simple count and percentage to reach our conclusions and the results were surprisingly stable. The following graph presents the range of program types by classification as revealed in our study.

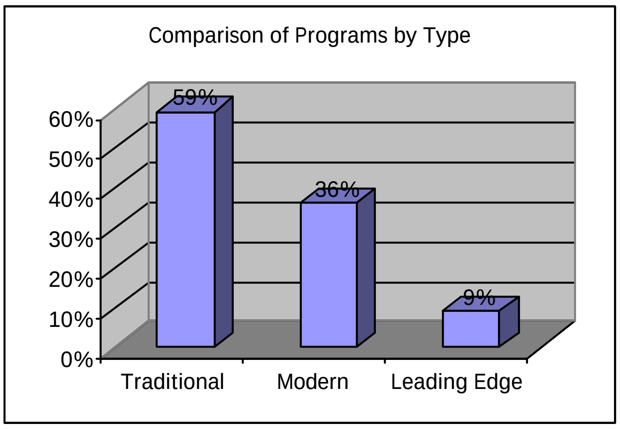

Figure 2

As can be seen. The programs representing traditional IT education still dominate although the number of legitimate software engineering programs appears to be on the rise. We actually discovered a program similar to ours at another institution in our sample and that is the reason for the unexpectedly high percentage score there.

Given that our approach is relatively unique, we felt that our disciplinary model needed to be explained in more detail. The final parts of this paper will outline our approach to software systems management. It is offered for the purpose of assisting any institution interested in such a curriculum in adopting such a model (the actual course list contained in Appendix A)

Disciplinary Model: Implementation

Our curriculum was built on the thesis that every aspect of software systems management can be understood and described as a component of four universal, highly correlated behaviors: *abstraction, product creation, product verification and validation*, and process *optimization*. Given this, our model curriculum was structured to provide the maximum exposure to current best practice in six thematic areas, which taken together as an integrated set, makes-up the attributes that differentiate us from the other computer disciplines:

Abstraction*: understanding and description of the problem space*
Design*: models for framing artifact to meet criteria 3, 4, 5, and 6*
Process Engineering*: application of large models such as IEEE 12207*
Organizational Control Systems*: SQA and configuration management*
Evaluation with Measurement*: with an emphasis on testing and metrics*
Construction: *professional programming languages with emphasis on reusability*

Our teaching strategy approaches this as a hierarchy of similar activities. **In every course** we require the student to define and implement all three interfaces and be able to clearly communicate this as a logically consistent model before working out the details of the solution. The focus of all understanding is top-down from the information interface. Our curriculum centers on the application of software engineering standards (such as those promulgated by IEEE) and the software process improvement, or quality standards (such as those promulgated by SEI and ISO) under the assumption that this embodies the "common body of knowledge and state of best practice" in software production and management.

The practical realization of this is an integration of the large subject areas of: software engineering (methods, models and criteria), process and product quality management (software quality assurance and metrics), software project management (work decomposition, planning, sizing and estimating), and software configuration management. Reconciliation of project and configuration management is accomplished by cross-referencing the problems, tools, notations and solutions (through explicit identification, authorization and validation procedures). As a side agenda, we have also stressed the need for re-engineering the vast number of software products currently on the shelves. This model plus germane simulated real-world experience introduces all of the relevant principles to the student within the (currently understood) framework. It allows them to develop and internalize their own comprehensive understanding and formulate a personal model of the disciplinary body of knowledge.

Summary And Conclusions

It is common industry practice that computer systems are defined and implemented by two very different types of personnel. IT workers, who function more like managers than technical staff, study the business operation. This is communicated to computer workers, who are usually more technical than managerial. They handle the details of actually developing and implementing the computer solution. This has always been an inefficient and cumbersome process containing numerous chances for misunderstanding and error. It is also a good explanation for the generally recognized low level of quality of most software. In the practical world computer system development no matter how inefficient is still a single activity. Therefore this would logically seem to be a single body of knowledge. Given the unquestioned recognition in the industry of the need for quality software, the value and advantage of an individual trained in one place in the methods and techniques of <u>both</u> areas should be intuitively obvious.

Although our program is innovative, in actuality it just takes the next logical step. Our curriculum is applied, not scientific and unlike scientists, we are entirely focused on the production of tangible artifacts, which can be used. Our success is judged by the proven quality of these products. Since tangibility is definitely not a requirement for "pure" science, the placement of our program in a college of business is appropriate. A very explicit goal of the

University of Detroit Mercy's College of Business Administration (or any other college as the case may be) is to produce leaders in the field. This leadership demands a knowledge and experience base obtained from focused study. No present degree seems to satisfy all aspects of the requirement. Business degrees satisfy it in traditional business. However, general business study simply doesn't fit in a technical area like computing. It might be argued that on a team, no one person should have the complete perspective, but the questions remains, who will supply the leadership? We believe that, as demands on computer systems become more and more complex, leaders will have to have a complete, top down perspective. Our program provides that view. We want to stress however, that we are not proposing new theory. Our principles are distilled from common elements found in all of these disciplines. It is simply our assumption that the framework for designing efficient computer systems needs to be fixed at the proper (e.g., the highest) level of abstraction. This is an appropriate concern because, by definition, design must incorporate techniques for dealing with the problem in its entirety. This is also not some fuzzy-minded, theoretical exercise. It is a critical issue with a very explicit, dollars and cents implication for every organization in the world. Right now, and even more so in the future, effective information will be the basis of an organization's ability to compete. In that respect computer technology will become the basis for a new competitive order in worldwide society. Effective systems will determine whether organizations will keep up with the competition or be left by the wayside in the marketplaces of the future. This effectiveness calls for a deliberate study of how computer systems can be made to meld naturally with our familiar, existing human systems. Our curriculum provides the basis for doing that.

REFERENCES

[1] M. O'Brien, Software Production Management, NCC Blackwell Ltd.: Oxford, U.K., 1992.
[2] Erik Brynnjolfson, "*The Productivity Paradox of Information Technology*", Communications of the ACM, Vol. 36, No. 12, pp. 67-77, December 1992.
[3] Construx Software Builders, web site @ www.construx.com, 1998.
[4] Edelstein, V., R. Fuji, C. Guerdat, and P. Sullo, "International Software Engineering Standards," *Software Engineering*, March/April.
[5] KPMG Technology and Services Group, web site at www.kpmg.ca 1998.
Laker Consulting, web site at www.laker.com.au., Sydney, 1998.
[7] Norman Fenton, Shari Lawrence Pfleeger, and Robert Glass, *"Science and Substance, a Challenge to Software Engineers*", IEEE Software, pp. 86-94, July, 1994.
[8] Watts S. Humphrey, *Managing the Software Process*, Addison-Wesley: Reading, MA, 1994.
[9] Capers Jones, *Assessment and Control of Software Risks*, Prentice-Hall: Englewood Cliffs, 1997, NJ.
[10] M. O'Brien, *Software Production Management*, NCC Blackwell Ltd.: Oxford, U.K., 1992.
[11] M. Paulk, B. Curtis, M. Chrissis, C.Weber, "*Capability Maturity Model, Version 1.1*," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1993.
[12] Software Engineering Institute, web site at www.sei.cmu.edu. 1998.
[13] Stephen S. Roach, "*Services Under Siege-The Restructuring Imperative*." Harvard Business Review, pp. 82-92, Sept.-Oct., 1991.

APPENDIX A

Software Systems Management Curriculum

Foundation

CIS 501 Introduction to Information Systems – Visual Basic

Core

CIS 505 Project Management
CIS 510 Object Oriented Software Development
CIS 520 Software Requirements
CIS 530 Software Quality Assurance and Testing
CIS 540 Strategic Software Process Management

Electives *(6 required)*

CIS 502 Structured Development for the Internet
CIS 503 Software System Documentation
CIS 525 Software Design and Construction
CIS 535 Metrics and Models for Software Management
CIS 543 Software Lifecycle Documentation
CIS 553 Graphical User Interface Development
CIS 554 Software Maintenance Using Cobol
CIS 555 Data Base Design
CIS 557 Networks and Network Management
CIS 558 Distributed Software Development
CIS 559 Electronic Data Interchange
CIS 560 Electronic Commerce
CIS 565 Information And Society
CIS 566 Advanced Database Issues
CIS 580 Advanced Topics in Is
CIS 589 International Software Management
CIS 590 Leadership in Assessment
CIS 591 Audit