The Case for the Study of Software Management

ABSTRACT: Software management represents a meaningful and advantageous new direction for traditional Information Systems curricula. The prevailing circumstance for I.S. education lends credence to the ancient curse..."may you live in interesting times." Change has become a stern task master. Hosts of fashionable ideas and newfangled innovations compete to influence the tenor and composition of I.S. training. Software management, as distinguished from software engineering and traditional I.S. study, offers a practical stratagem focused on a pivotal issue in I.S. practice, cost-effective software production. A complete set of principles and methods for efficient manufacture of software has never been studied as such. It isn't that "current best practices" don't exist. It is just that they are not cardinal elements in traditional studies of computing, which quite appropriately center on the technology itself. The University of Detroit Mercy's graduate curriculum establishes a consistent architecture for an academic program to prepare executive leaders expressly for the software industry. Pragmatically, the challenge was to adopt a reliable point of reference to identify and consolidate a valid course array. Buttressed by a review of the literature, we adopted the thesis that the conceptual framework currently employed to depict the rational management of software is incomplete. Instead, technology–centered approaches have been introduced piecemeal. This has begotten the "silver bullet" mentality. Consequently, we organized our model curriculum from a higher level of abstraction. This yielded six thematic areas that we believe encompass the entire problem. Taken together these comprise the attributes that differentiate software management from general business management and the other computer disciplines. We present a pragmatic model that details our successful graduate program.

Dan Shoemaker
Vladan Jovanovic
Computer and Information Systems
University of Detroit Mercy
Detroit, Michigan 48219

KEYWORDS: *I.S. Education, Software Management, Software Process Optimization*

INTRODUCTION

It is time to acknowledge that the study of software management offers a legitimate curricular alternative to traditional Information Systems education programs. As a regimen, software management can be the sovereign remedy for two of the industry's dominant woes, cost control and production efficiency. In practice however, modern software management resembles Dickens more than Demming. Which is understandable, since it's extraordinarily difficult to establish and maintain organizational control over an activity that is creative and conceptual by nature. Effectiveness demands strategic acumen that can only be gained through broad experience, or advanced training. Without this, "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 (6)." Consequently, "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" (7)

WHY STUDY SOFTWARE MANAGEMENT?

By 1995, the software industry plans to bank a half-trillion-dollars annually (5). Yet, with the stakes that high the manager who is..."knowledgeable in the realms of new technology is a rare breed" (7). To reinforce the premise that competent software management is vital to business, let's inspect the trenches. Brynjolfsson provides a very apt synopsis of their current state: "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" (1). For instance, a recent survey found that fully one-third of the government's software is unusable when delivered and 29% is never delivered at all (5). Over the last decade, the GAO estimates that the federal government's bill for worthless systems topped $150 billion (2). Industry is not exempt from this either. An authoritative study quoted in the Harvard Business Review (10) reveals a very telling statistic. During the 1980's, the sector that invested the least in information technology (manufacturing) achieved the greatest total increase in productivity. The business sector with the highest investment (services) realized no gain at all. The economist Robert Solow astutely sums up this well documented and generally appalling "productivity paradox" with the following quip: "we see computers everywhere except in the productivity statistics"(1). That fact raises a valid concern. If the point of information technology is to secure the competitive high ground, what's the unavoidable conclusion if that anticipated windfall turns out to be a faint zephyr? Based on an extremely rigorous review of the literature, Brynjolfsson found four possible explanations. Of these, the two most likely culprits were ineffective measures, or inefficient management. In either case, a shocking few people seem to fathom the significant business implications of the technology they are 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 (7)."

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. The results were less than auspicious. 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." James Martin was even more direct.

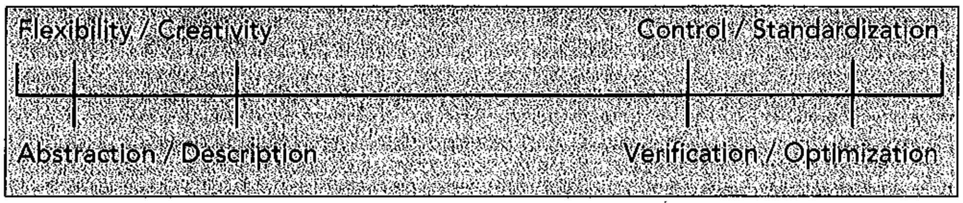
" Neither the business schools nor computer scientists have gotten near understanding what's happening... (in computing) ... let alone teaching it... I think that many top people in computing at the present time are extremely dismayed by what's happening. I was at an important facility of one of the world's largest computer manufacturers recently, and the general manager commented that he would literally, never hire a computer science graduate (4)."

All of this factors into a single problem, focus. Businesses want graduates with job skills matched to current practice, who can fit immediately into explicitly defined areas of need. University curricula have a long-term orientation stressing knowledge as a whole (4). 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. Contrast the following quotation, from a respected member of the higher education community, with Martin's earlier statements and draw your own conclusions.

" The impact of computer science graduates is not in the DP room. The DP manager and the DP room are undergoing a tremendous transition right now and I'm not sure we should be listening to them for our university work (4)."

So, we have consumers who want to buy apples and producers who are absolutely convinced that what's required is an orange. Of course, this survey was conducted almost ten years ago. Perhaps like fine wine the situation has improved with age? A comprehensive study of university catalogues, conducted in 1993, found that 90% of business school curricula were at best... "incomplete and lagging behind the state of the art by more than five years (6)"... when it came to the requirements of managing software. More pertinent, although almost ten years separate these two studies the same symptoms were identified: "weak, or inadequate business preparation and impractical, or out-of-date curricula (4)." The underlying basis is easy enough to spot. "The problem of inadequate software management itself is caused by the long time required to modernize university curricula in the face of major paradigm shifts" (6).

Figure 1. CONTINUUM OF PROCESS REQUIREMENTS AND ACTIVITIES


Flexibility / Creativity
Control / Standardization
Abstraction / Description
Verification / Optimization

Both studies make it clear that the key to the formulation of a successful software management curriculum relies on the ability to amalgamate "current best practice" ideas into an optimally valid and accurate model of software process management. The dilemma lies in the fact that,

...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. (3)

Besides the implication that our eminent standards bodies need to be leashed, this astonishing productivity (roughly 10 new standards a year) proves the crucial importance of a theoretical framework to help us judge... "which practices to standardize" (3).

BASIS

The challenge was to differentiate a reliable frame of reference through which a valid curricular array could be identified and related. As we've seen, it isn't that "current best practices" don't exist. It is just that they are not cardinal elements in traditional studies of computing, which quite appropriately center on the technology itself. It is this focus that begets the "silver bullet" mind-set. O'Brien places this in its social context.

" In today's world we see human beings circling around technology, we see them driven by that technology; indeed overpowered by the technology; yet we constantly try to appeal to that technology to save ourselves from its apparently dominating power. My thesis is simple, the technology should be seen to revolve around us; we are fully responsible for our own views and we control these views. By displacing ourselves from the centre we have managed to absolve ourselves of responsibility and simultaneously put ourselves at the whim of something apparently beyond us, yet that very something is made by us and ultimately controlled by us (7)."

At the theoretical nucleus of our undertaking is the belief that software process management's conceptual framework is not abstracted at the proper (e.g., the highest possible) level. Instead, technology centered approaches are 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 (11).

Practices that foster the software process have never been merged into a distinct, curriculum and studied as such. Our curriculum was built on the thesis that every aspect of software management can be understood and described as a component of four universal, highly correlated abstruse behaviors: ***abstraction, description, verification, and optimization.*** The purpose of software management is to maintain an optimum balance between maximum degrees of creative freedom and management control. Under that practical assumption, these universal behaviors can be laid out along a process continuum of dynamic activities (5) (see Figure 1 above):

This can be realized in many organizational forms. One concrete example is a major Inter-European research initiative (Esprit phase II). The strategists behind this are about to define a unified approach for the management of projects across Europe (9). They believe that software and information systems development are based on abstraction and driven by a common set of functions and activities. These pivotal processes are often confused with the

Figure 2. MODEL CURRICULUM IN SOFTWARE MANAGEMENT

Key Process: Abstraction and Description

Strategic Issues in Database Management
Topics include conceptual modeling, logical and physical design, data administration, benchmarking, database tuning, Entity-Relationship and object databases. (LECTURES)

Specification and Design
Specification of software systems; principles, methods and techniques of effective software development, including; structure representation and optimization, and software engineering environments. (CLINIC)

Object Oriented Software Development
Principles, and templates for developing effective software artifacts. This is not a programming course however, an object oriented language (C++ or Ada) will be used to demonstrate principles advanced.(DESIGN STUDIO)

Key Process: Optimization

Strategic Software Process Development
Capstone issues in software management: competitive analysis, organizational theory and development of process models. Principles and practices for strategic management of production, software process maturity, risk assessment, software acquisition. (DESIGN STUDIO)

Software Project Management
Software project management, tools and standards, including; organizational and project planning, WBS, sizing, estimation and scheduling, team formulation, group dynamics and training.(CLINIC)

Software Maintenance Management
Planning and management of maintenance; standards and tools, practical methods for reusability, maintenance plans, program re-engineering, understanding and documentation of existing programs. This is not a programming course, but some Cobol will be used as a practical demonstration of principles advanced. (CASES)

Key Process: Verification

Software Quality Management
Methods and principles employed to establish and manage a quality system in software production, including; quality metrics and improvement programs, SQA standards and documentation.(DESIGN STUDIO)

Software Configuration Management
Strategic software product management; processes, tools, standards and models for the identification, authorization, and verification of software configurations, management of software assets through libraries.(CLINIC)

SQA and Test Management
Comprehensive coverage of unit, module, system and acceptance test principles. Methods, models, standards and tools used in the process of testing. Management of a software quality assurance and testing process.(CASES)

Figure 3. EVALUATION MODEL

Competency Level	Evaluation Methods	Delivery Mechanisms
5. Optimizing	Design	DESIGN STUDIO (4-5)
4. Design	Presentation	CLINIC (3-5)
3. Application	Problem Analysis	CASE STUDY (2-4)
2. Discernment	Relationships	LECTURE/LAB (1-4)
1. Knowledge	Recall of Facts	LECTURE (1-5)

physical concerns of application that lie at a different level of operation. Esprit II merges both orientations into a single study, which recognizes the common conceptual origins of information systems management and software engineering.

ARCHITECTURE

The University of Detroit Mercy's graduate curriculum is a deliberate attempt to define a consistent archetype for the effective study of software management. We adopted an observation by Capers Jones as a practical touchstone for this endeavor:

" Universities and business schools that have established close ties to industry and particularly to leaders such as AT&T, IBM, Hewlett-Packard and Motorola... tend to have the most modern and dynamic curricula. Universities and business schools that are more or less detached from daily contact with actual software producing companies are the most susceptible... as of 1993 the numbers of software managers trained in functional metrics (the basis of good management practice) by private instructors and management consultants exceeds the number of students trained in university and business school courses by perhaps 1000 to 1". (6)

A straightforward distinction has to be made between software management and software engineering. The software manager builds on the software engineer, but does not have the same orientation. Pragmatically, it is not the same profession. By definition, the software engineer is focused on creating optimally efficient software artifacts. The software manager is concerned with creating optimally efficient processes within which these artifacts are built (5). Given this, our model curriculum has been 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 software management from general business management and the other computer disciplines:

1. **Abstraction:** *(abstraction) understanding and description of the problem space*
2. **Description:** *(description) models for framing the artifact to meet criteria 3, 4, 5, and 6*
3. **Optimization:** *(optimization) practices and tools*
4. **Quality Control:** *(verification) SQA and configuration management*
5. **Measurement:** *(optimization) management through metrics*
6. **Manufacture:** *(abstraction, verification and optimization) with emphasis on reusability*

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 softwares'. 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. Inevitably, most of our software management graduate students are already practitioners, who function at many levels in the industry (from CIO to programmer). Consequently, it was crucial to adapt the course delivery system to the student's learning style. The direction can be top-down; project planning through requirements and design, to validation, testing and code. The alternative starts at detailed programming and moves upward through requirements, testing and quality assurance to software management principles. 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. The curriculum this produces is represented in (Figure 2). Experience is the number one consideration for success.

General competency is evaluated based on only two factors: their ability to function and create artifacts within a simulated real-world situation, and their ability to move up through a set of learning/performance levels adapted from Bloom's taxonomy (e.g., from rote knowledge, through application). Figure 3 details that evaluation model. It provides explicit criteria for specification of the educational objectives and outcomes for each course. Standard syllabi enforce this by addressing the criteria specified for each level. Obviously, this is critically dependent on the development of a body of experienced teachers.

General curricular strategy centers on the ability to establish explicit relationships among competency levels required, evaluation methods needed to assure achievement of these competencies and adequate delivery mechanisms. Competency levels are analogous to the software process maturity levels defined by Humphrey (8). Software Management as an academic study represents a reorganization of the disciplines that represent the fields of Information Science and Software Engineering. Common problems, tools, notations, and solutions are cross referenced to achieve the synthesis. The result is a general disciplinary structure that integrates the functional areas of: Software Development and Maintenance, Software Quality Management, Software Project Management, and Software Configuration Management into a single study focused on optimization of the software process as a total entity.

CONCLUSION

The study of Software Management has the potential to pilot Information Systems curricula through the endless change that is a fact of modern computer education. Software management provides a unification of proven ideas that can generate productive and efficient information systems, while maintaining the required focus on software development and maintenance functions. This new study is not a business discipline, there is too much technical content to justify that, nor is it strictly speaking a technical discipline. It is not a panacea, or the only curricular option. However, as resources become tighter and demands on computer systems become increasingly complex, it is a realistic, viable and attractive alternative to traditional approaches. Fantastic advances in technology are self-indulgent until they are effectively applied. Effectiveness calls for a deliberate study of how the computer software development and maintenance process can be made to meld optimally with the competing needs of business organizations. We are *not* proposing new theory, or methods: that is exactly what we are not proposing. What we *do* propose is that Software Management's conceptual frame of reference provides the essential consolidation of all necessary components into a single productive direction.

REFERENCES

1. Erik Brynjolfsson, "The Productivity Paradox of Information Technology", *Communications of the ACM*, Vol. 36, No. 12, pp. 67-77, December 1992

2. "Federal Government Mismanages Computer Costs", *Detroit Free Press*, March 17, 1993, p. 1B

3. Norman Fenton, Shari Lawrence Pfleeger, and Robert Glass, "Science and Substance, a Challenge to Software Engineers", *IEEE Software*, pp. 86-94, July, 1994

4. Curtis Hartog, "Of Commerce And Academia", *Datamation*, September, 1985

5. Watts S. Humphrey, *Managing the Software Process*, Addison-Wesley: Reading, MA, 1993

6. Capers Jones, *Assessment and Control of Software Risks*, Prentice-Hall: Englewood Cliffs, 1994, NJ, pp. 217-218

7. M. O'Brien, *Software Production Management*, NCC Blackwell Ltd.: Oxford, U.K., 1992

8. M. Paulk, B. Curtis, M. Chrissis, C.Weber, "Capability Maturity Model, Version 1.1," Technical Report, Software Engineering Institute, Carnegie-Mellon University, 1993

9. Project Survey "Software Research in ESPRITs Second Phase", *IEEE Software*, November, 1989

10. Stephen S. Roach, "Services Under Siege-The Restructuring Imperative." *Harvard Business Review*, pp. 82-92, Sept.-Oct., 1991.

11. Tomayko, James E., "Software Configuration Management", Software Engineering Institute, Carnegie-Mellon University, 1987, p. 1-2

Authors' Biographies

Dan Shoemaker, Ph.D., is an Associate Professor and Coordinator of the Computer and Information System Program. He has extensive background as an MIS manager and information system developer.

Vladan Jovanovic, Ph.D., is an Assistant Professor of Computer and Information Systems. He has extensive experience in European and American software engineering practice.

ISSN 1055-3096