Familiar Passages

Do I even need to say it? There is a problem out there with software development! Within the last twenty years, the cost of developing software has exceeded the cost of developing the hardware platforms it executes on for computer-based systems [7]. During that time frame, the software technology effectiveness, that is, the cost-performance ratio, for computer-based systems has increased at a rate of 1000 to 1 every 10 years [8], resulting in the movement of additional functionality from hardware to software. Over the last 30 years, software technology has seen an increase in performance-price gains of six orders of magnitude [9]. Today, it is estimated that over 90 percent of computer system costs are associated with software [10], and that software development and maintenance represents an industry cost of over $300 billion a year [11].

Despite the economic and functional importance of software, developing software is still generally thought of as a high-risk effort, and little progress has been made in improving the management of these risks. One example illustrating the lack of software risk management is the perception that an overwhelming majority of the problems with late deliveries of software are related to software technology problems (such as the development of complex algorithms), over which there is little control. Research negates this perception, and shows that 45 percent of all causes for delayed software deliveries are organization-related problems which management can control [12].

The lack of successful software management has contributed to the realization (and self-fulfilling prophecy) that software development is an uncontrolled activity which is not being managed in an effective manner. From a business viewpoint, the success criteria used to measure software development efforts include return on investment, time to market, and customer satisfaction. From a tech-
nology viewpoint, the success criteria used to measure software development efforts include meeting functional requirements, usability of the product, and future support. Based on a survey conducted by the US General Accounting Office, the results of software development efforts and the management of the risks using these success criteria are not too impressive. In this survey, 50.4 percent of the vendors supplying software to the US Government indicated that software development is associated with cost overruns, and 62 percent indicated that software development is also associated with schedule overruns [13]. A similar study performed in the United Kingdom (one that involved 60 companies and more than 200 software projects) indicated that 55 percent had experienced cost overruns, and 66 percent also experienced schedule slippages [3].

The Industrial Viewpoint

Software management can be viewed from two macro perspectives: an industrial view and a practitioner view. The industrial view sets an overall picture of how software development process efforts, and their associated risks, relate to software products. The practitioner's view influences the activities, or lack of, performed during software development that relate to the project's success. Both of these views independently contribute to the symptoms of poor software management: that is, late deliveries, budget overages, and erroneous or missing technical performance expectations.

Although there are many problems associated with software development, the three most prevalent from an industrial viewpoint are:

1) The perception that the production of software is still not recognized as a major development effort. As identified previously, 90 percent of all computer system costs is associated with software. Most companies developing software, especially software embedded in hardware platforms, track the identity and associated project costs with the delivered end item, which in most cases, is hardware. The inability to identify software costs occurs because it is buried in the cost account for the hardware product. This happens when the Work Breakdown Structure (WBS) [10] is not organized to identify software development as a separate item. In more recent times, major corporations developing software
have recognized this and are keeping better cost data. For software being developed by medium- and smaller-sized companies, however, software development is still often not recognized as a major development effort.

2) Despite many advances in software engineering as a discipline since it was first defined in 1968 [14], there is still the perception that software is a creative endeavor—that it cannot be “engineered” like other disciplines. One Air Force decision-maker made this observation: “You software guys are too much like the weavers in the story about the Emperor and his new clothes. When I go out to check on a software development the answers I get sound like, ‘We’re fantastically busy weaving this magic cloth. Just wait a while and it’ll look terrific.’ But there’s nothing I can see or touch, no numbers I can relate to, no way to pick up signals that things aren’t really all that great. And there are too many people I know who have come out at the end wearing a bunch of expensive rags or nothing at all” [15]. Inconsistent application and management of the software engineering discipline still contribute to this perception.

3) The perception is that even if it is recognized that software development contains risk, nothing can be done to control the risks. History has shown most software endeavors are late, over budget, and have poor or missing technical performance. This assertion has been documented in many research findings, including research by Suding [16] which shows that the typical software development project is 47.5 percent over budget. In another study, Dinitto states that in some extremes cost growth can reach 1000 percent [17]. Putnam states the typical software development project is 200–300 percent over budget and 100 percent over schedule [18]. In a study of five large software telecommunications projects, Karolak found software projects were up to 158 percent over budget and 89 percent over schedule [19]. Other examples from industry include data on cost overruns and schedule slippages from Allstate Insurance, The City of Richman, Business Men’s Assurance, and Blue Cross & Blue Shield United of Wisconsin [20]. Davis [4], using government data, demonstrated that out of nine software projects funded by the US Government, 29 percent of the software costs was spent on software that was never delivered, and another 19 percent contained errors and
missing requirements resulting in extensive rework. Davis determined that only 2 percent of the total costs in the study was spent for software that initially met the customer requirements and was used without modification (as shown in Figure 1-1).

**The Practitioner Viewpoint**

From a practitioner viewpoint, there are several problems associated with the risks of software development. The four most prevalent ones are:

1) *The lack of understanding/education about what is involved in the management of software.* The concept of software development management and the risks involved in this type of product are not readily addressed in our university educational system or commercial training seminars. Management concepts can be found in university business programs, which only occasionally addresses the concept of risk as it relates to investment or insurance decisions.
However, these concepts are not easy to associate with the software engineering process.

2) **The lack of discipline when implementing good management techniques.** Even if one is educated in good software management practices and risk management techniques, it requires much rigor and discipline to identify, calculate, determine, plan, collect, and report software risk items. It can be a laborious task, especially when one is constrained by budget and schedule pressures.

3) **The lack of tools needed to perform software risk management.** Although more project management tools are currently being made available for software project management, the tools to implement a software risk management program are not well identified, not automated, not easily accessible, and not available to most software professionals. If the tools are not readily available and easy to use, software management techniques that consider risks will not be implemented.

4) **The limited view of software risk management and the failure to integrate software management initiatives that involve risks.** Little information associated with software risk management has been identified in specific areas during the software development life cycle, such as cost estimation or testing. Thus, most people who have been introduced to the activities of software risk management have a limited view of when, what, and how it can be performed. To be more effective, software risk management must be viewed holistically throughout the entire software development life cycle.

**The Example**

So what does this mean to you? Let us see if the following scenario sounds familiar. You are a software manager leading a software project that is twelve months into an eighteen month development period, in which the software developers are coding and performing some unit testing. You are already four weeks behind schedule, but you are not too worried because you will figure some way to make up the time spent in a slow start getting the project off the ground. The software development team is in good spirits.
Now it is month sixteen. You are supposed to be well into integration testing. You have started some integration but many software modules are still behind schedule. As a matter of fact, your overall schedule is now estimated to be seven weeks behind. You are working the software developers overtime hoping to catch up on the schedule but that has not worked. Your budget for the project has been overrun, and you are expecting a significant cost growth. You have used all of your hours saved under management contingency. Your management is worried. You are worried. But you hope that a plan, one that is based on all the time you spent in reviews and updating documents, will pay off in the end because of fewer errors during testing.

It is month eighteen. You have not completed integration. You are still continuing to find software bugs. This is a fixed-price job. The projection for any profit is lost. The company is eating the cost. Pressure to ship the software increases, but the software does not meet the requirements. You hear the following comments: “The problems are in the software, not the hardware”; “The software is late again, as usual”; and “Those software people can never meet a schedule or a budget!” You are asked when the software will be completed. Best projections are two more months. The customer is upset because their product will not be shipped to them on time. You call your lead software developers together. As a team, you reflect on what you did. You followed the software engineering procedures that took over a year to develop and learn. Those procedures were even audited by an outside agency. You wrote the specifications, documented the designs, held the reviews, developed test plans and procedures, and your estimates were based on the software developers and your best judgments. Sure you took some shortcuts, but nothing significant. It was not good enough.