Software Development Project Failures

The use of information technology (IT) has become pervasive in the management of organizations in the public and private sectors in today's information economy; it is impossible to imagine any organization, however small, not using computers in some manner to stay competitive. One of the main elements of the all-consuming influence of IT is software. *Software* is the collective term used to describe the set of program instructions that have turned the computer into such a formidable extension of the human capacity to store and process vast quantities of data. The increasing pressures of the information economy to make organizations more efficient and productive in the marketplace have driven executives in almost all organizations to look for that competitive and strategic advantage through the judicious use of software. Software has thus come to be viewed as a vital component of any organization's efforts to remain competitive in the information economy. New software projects are constantly being proposed for development in organizations to meet their computing and information needs. Despite the best efforts of these organizations, their well-thought-out plans for various software projects may end up as failures—sometimes monumental failures with unforeseen consequences, and with blame and recriminations throughout the organization. In some rare instances the software failures have even been known to endanger the very survival of the organization (the case of FoxMeyer Drug Company, reported in the *Wall Street Journal*, provides an example (Wysocki 1998)).

Organizations in both the public and private sectors have often been forced to cancel software projects that have far exceeded their original cost estimates and schedule or time-of-completion estimates, or that have failed to achieve the desired minimal functionalities. The costs of such project cancellations have been staggering in recent years, owing in part to the increased size and complexity of new software technology. Most of the software failures, especially in private-sector organizations, remain unreported and are written off as part of the cost of doing business. However, in some cases the costs involved and the circumstances surrounding the failure are so spectacular that the details are reported in the news media, as happened for example in the cases of AMR Inc.'s CONFIRM (Oz 1994); FoxMeyer's Delta project, which contributed to its bankruptcy (Jesitus 1997; Wysocki 1998); Hershey Foods' ERP project (Collet 1999; Stedman 1999; Osterland 2000); IRS's Tax System Modernization (TSM) (GAO, 1995, 1996); Denver International Airport's Baggage Handling System (BHS) (Rifkin 1994); Bank of America's Masternet System (Szilagyi 1998); Pacific Gas & Electric's Customer Information System ("PG&E Dumps CIS Project," 1997); and the Medicare Transaction System of the Department of Health and Human Services (GAO, 1997). A number of studies have estimated the dollar drain on the U.S. economy to be in excess of several billion dollars a year. For example, in 1995 a study by the Standish Group of U.S. companies and government agencies found that 32 percent of corporate IT projects were *abandoned* before completion at a staggering cost of $81 billion. A later study by the Standish Group, reported on by Rick Whiting (1998, 20) in *Software Magazine*, showed similar figures for 1998. The failure record overall is not very encouraging, as they state: "In the United States, we spend more than $250 billion each year on IT application development of approximately 175,000 projects. . . . A great many of these projects will fail." They go on to describe the chaotic state of software development, warning that we can no longer ignore the problem of software failures (also see Standish Group, 1995; Johnson 1995; Wysocki 1998).

Other studies have confirmed that the problem of software failures is not unique to the United States. In the United Kingdom, a study by the KPMG Consulting Company titled *Runaway Projects* found that "runaway projects have become the bane of organizations' attempts to transform their business" (Cole 1995, 3). The study goes on to provide more detailed statistics to support this claim, which we will discuss in a later section. Another UK study by a special interest group (OASIG) concerned with the organizational aspects of IT concluded that most investments in computer-based systems fail to achieve the objectives of the organizations using them (OASIG, 1996). Again, we examine the details of the study in a later section. A study of Canadian federal government IT projects reported on in *Infosystems Executive* by Erik Heinrich (1998) found similar disturbing news of several billion Canadian dollars spent on projects with questionable results. In some cases outright cancellation occurred, as with the Canadian Automated Air Traffic Control System (CAATS), which ended up costing more than one billion Canadian dollars over a ten-year period. Software failures are not limited to any geographic region, industry, or market group, nor are they restricted to specific organizational size. The Standish Group study reports failures in organizations of all sizes.

Outline

This book offers empirically grounded explanations of the variety of organizational, managerial, and other relevant factors that have contributed to project failures, based on survey data and validated by analyses of different cases of failed software projects reported in the public domain. The book is made up of three parts. The first part lays out the concepts, issues, and methods at the core of the software development process. It discusses the conceptual explanatory factors underlying software development that make them vulnerable to problems, including failure from requirements analysis and determination, to design, and failure in implementation. Chapter 1 broadly explains why issues of management and organization are still at the heart of software development. Chapter 2 expands on this analysis by examining the inherent characteristics of software that produce software development crises, despite the remarkable progress made in software development methods over the last several decades. Chapter 3 analyzes the traditional life-cycle method of software development, the challenges it poses in the development process, and its critical influence on project outcomes. To provide a perspective on the multidimensional nature of software project abandonment, the book presents a framework of factors that contribute to the abandonment decision. It draws attention to the risks and uncertainties present in each phase of the systems development process that may contribute to project failure. Throughout, it emphasizes the significance of the organizational and managerial issues at the root of many software project failures. These issues, and the project failures they cause, have persisted since the birth of software development as an engineering science at the NATO conference of 1968, in spite of major developments in the field.

The second part of the book examines the multiplicity of cofactors—organizational and managerial—that make software development a risky undertaking whose outcome often cannot be predetermined. The discussion covers the organizational, managerial, economic, and technical and technological problems at the root of the failed projects, and presents empirical data obtained through surveys of organizations. Chapter 4 deals with the set of empirically derived socioorganizational factors and problems arising in the development organization, which have the potential to derail software development projects. This is followed in chapter 5 by an enumeration of the technical and technological factors that may be at the core of abandoned software projects. Finally, chapter 6 explores the significance of economic issues that contribute to decisions to abandon projects. In each chapter, the empirical data are validated with analyses of actual reported cases of abandoned software projects in organizations. Chapter 7, the final chapter of this part, reveals the views of users—in a case study of an abandoned software development project in an electronics distribution company. Particular emphasis is placed on the role of users in the development process and on the organizational politics associated with the project development. Each chapter of this part of the book ends with an assessment of the significance of each factor to the different states of the software development process.

The third part of the book puts forward lessons that project organizations can learn from the analyses of failed projects and shows how those lessons can be used to improve software development practices. Chapter 8 offers a learning paradigm. This paradigm is intended to encourage widespread learning and sharing of experiential knowledge about failed projects and to collectively improve the practice of software development industrywide. Chapter 9 provides a paradigm of software development that outlines some of the critical issues organizations must pay attention to during project development in order to maximize the chances of successful project outcomes. The discussion highlights the promise of the evolutionary approach to software development, which emphasizes modularization and intensive collaboration between developers and users, who provide constant feedback on early versions of the software. Published studies by MacCormack and colleagues (MacCormack 2001; MacCormack, Verganti, and Iansiti 2001) suggest that the evolutionary development methodology is particularly appropriate in development environments where the technology and the requirements for systems development are dynamic and unstable. Finally, chapter 10 examines the aftermath of project failures. It both elucidates the empirical factors that critically affect project outcomes and suggests ways decisions on project terminations should be handled to minimize damage to employees' and project teams' morale and to safeguard organizational resources for future projects. The book ends by focusing attention on telltale signs of problematic projects that need to be aggressively scrutinized by project developers and organizational executives.

What Is Software Development Failure?

The problem of software development failure has been with us since the early days of the computing revolution and will likely persist in the future. The hope is that with time, as software developers become more informed and understanding of the intricacies of the technology, the incidence of software failures will subside substantially until they are relatively rare. Until that time arrives, we need to do more to understand the problem and ways to tackle it.

Software failure can be classified into two broad categories, the first dealing with the inability of the implemented software to perform to the expectations of the users and the second dealing with the inability of the software developers to produce a working or functioning system for the users. These categories are two sides of the same problems, but our main focus in this book is on the latter issue—that is, on the developers' failure to deliver a working or functioning system to the users. Thus software failure can be formally described as a development failure. It can be further characterized in a number of more specific ways. The failure may be with respect to meeting the original cost or schedule estimates, or it may lie in an inability to achieve the functional objectives of the project. Software development failure can be described as the occurrence of one or more of the above types of failure, and such failures can occur at any stage of the development life cycle.

Software product development is a labor-intensive, intellectually demanding creative activity. The challenge is to be able to manipulate symbols and words to construct a logical flow of the sequence of activities or tasks to be performed to achieve the desired goal or objective for the software product. The difficulty stems in part from determining what information or data requirements will enable the specified objectives to be achieved. A further difficulty arises from the need to determine what resources—that is, technical and technological, financial capital, and others—are needed to be able to attain the desired software product objectives within some previously estimated time frame.

Underlying the above difficulties are the risks and uncertainties associated with the entire software development effort. When the software development fails to attain any of the targeted goals of the project—that is, the cost and schedule estimates for completion, or the satisfaction of the project's functional objectives—the development effort can be described as a failure. It is important to point out that whether the original cost and schedule estimates were inaccurate or indefensible is not relevant because once the project is begun, under whatever agreed-on conditions, the inability of the project team to achieve the targeted goals makes the project development effort a failure.

Others characterize the types of development failure described above in a variety of ways. For example, the KPMG study used the term *software runaways* to describe instances of development failure in which the cost overruns exceeded the original estimates by at least 30 percent and/or the project failed to significantly achieve its objectives (Cole 1995). Robert Glass (1998, 3) found the 30 percent figure too restrictive, instead offering a more expansive definition of what constitutes "software runaway" as "a project that goes out of control primarily because of the difficulty of building the software needed by the system."

The Standish Group (1995), on the other hand, categorized software failure in two ways: as projects that are "challenged"—that is, where even if the projects are completed, they are invariably "over-budget, over the time estimate, and offer fewer features and functions than originally specified"—and as projects that are "impaired," in other words, "where the projects are canceled at some point during the development cycle."

Still another conception of software development failure is offered by Lyytinen and Hirschheim (1987), who focus on the expectations of the stakeholder groups. If the software development effort or process fails to meet the stakeholders' expectations as to the systems objectives and/or cost and time-of-completion estimates vis-à-vis the benefits to be derived from the system, and if the project is allowed to continue, it is considered a failure. I will use this concept of software failure in my discussion because it seems to capture the essence of the problem more closely than the concepts mentioned earlier, for the following reasons. First, it involves the interaction of the relevant stakeholders engaged in the project development effort. Second, it is more encompassing of the various descriptions of software failure described above, because it appropriately captures the interacting relationships existing in the software development process among the underlying entities of systems objectives, development costs, and other resources needed to produce a functioning system within the estimated time frame to the satisfaction of the stakeholder groups. Indeed, software failure has been aptly described, in Lyytinen and Hirschheim's (1987, 274) analysis of IS failure, as "an extremely complex web of social and technical phenomena."

In summary, it can be said that software development failure is characterized by several factors; it is multifaceted and multidimensional, and any one of the various contributing factors may be sufficient to bring down the entire edifice of the software structure. However, the software development enterprise is a purely abstract and conceptual endeavor, and as such places an undue burden on all the stakeholders to collaborate with a clear vision of what is to be achieved, how it is to be achieved, and at what cost and in what time frame. When that vision falters or is somehow impaired, failure of the project becomes a distinct possibility. When the software development process becomes so imperiled that continuation is pointless, abandonment of the project becomes a necessary option. It is this class of software project failures or abandoned software development projects that this book explores.

Software Engineering and Software Development Failures

Software development as an engineering science was born at the 1968 NATO conference convened to address the problems encountered in the software crises of the 1960s (Wasserman 1980; Blum 1994; Robinson et al. 1998). The emergence of software engineering as a systems development discipline was based on the perception that the modern scientific method—with its emphasis on formalism, rationality, and objectivity—could provide the correct approach to solving the host of problems associated with software systems development. Some of the proponents of this approach, including Dijkstra (1968), Hoare (1984), and Lehman (1989), argued that the adoption of a formal logical or structured approach would provide the "silver bullet" that would aid the discipline in overcoming the crises of systems development. For example, they argued that applying the objective standards of the modern scientific method to computer programs would enable systems developers to "intersubjectively" test their programs for correctness. Over the years, considerable energy was devoted to formulating concepts and solutions involving proofs of program correctness, stepwise refinements, functional decomposition, structured analysis and design, modular structure, information hiding, and structured programming, in a concerted effort to reduce the inherent complexities associated with formulating and solving the right systems problems (Sommerville 1997; Blum 1994; Marciniak 1994). In time, several different process models were developed to improve the practice of software development. The common theme underlying the various process models is the abstraction of the sequence of activities needed to create a software product from the user's problem description. At present, as Sommerville (1997, 2260) has rightly observed, there are "no accepted standards for describing process models." The majority of software development models in use in organizations are based loosely on some generic process model for generating the documentation required to communicate with the project team, clients, and management. Robinson et al. (1998, 366), in a critique of this era in systems development, has characterized this approach as one based on the erroneous belief that "software engineering embodies a view of the world as being composed of unitary problems, each capable of rational solution via the application of technology." This analysis of the use of "formal conceptual schema" to model the complexities of the real-world problem domain is based on the premise that natural language—with its inherent ambiguities and varied ways of describing a problem—is somewhat unsuited to handling the rigor of the "deep semantic structure of a given" problem context. Although considerable progress has been made over the last several decades in tackling systems development problems, the software crises are still with us.

The view of software as a purely objective artifact or product that is created devoid of context is somewhat misleading. Context—be it organizational, cultural, managerial, or technological—shapes the software development process and the final product created. The weakness of the modernist approach to software development, with its emphasis on rationality, highlights the need to pay adequate attention to all the issues that legitimately direct the development process. The significant evolution of the tools and techniques of software development has tended to emphasize the rational or formal problem-solving perspective. However, the complexities of the cofactors that shape the "solution space" of the software development process have not diminished over the years. Hence the persistence of software crises and the continual search for ways to reduce their incidence in software development practice.

In Brooks's (1987) view, no "silver bullets" exist capable of coping with the inherent difficulties associated with the nature of software and the manifestations of those difficulties in the systems development process. Brooks's realism leads him to opine that as far into the future as he can see, there exists "no simple development in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity" in software development (p. 10). In essence, the software crises will always be with us. Thus the best we can do is to find ways to reduce their impact on the software development process. Harel (1992, 10), commenting on Brooks's pragmatic but somewhat pessimistic verdict, offers a more encouraging recitation of developments in software engineering that may provide "a vanilla framework for system modeling" and that indicate good prospects for the improvement of systems development. Robinson et al. (1998), however, see a breakdown in the "modern" scientific or formal approach as employed to tackle systems development. They argue that the formal systems derived by this rational/logical approach must eventually be "connected" to the real world they purport to represent in order to be validated. "It is at these points of connection that problems may occur," they suggest, "for this is where the implications of the epistemologies of the system and the world connect. The system's success depends on whether these epistemologies collide or co-operate" (p. 368). These are precisely the problems inherent in the nature of software that Brooks describes as the "essence" and "accidents" of software development. Adding to these problems are the inherent difficulties users have of communicating their tacit knowledge of the problem domain in the form of systems requirements to the systems developers. Moreover, as Eischen (2002, 39) observes, the users' tacit problem-domain knowledge is often "undefined, uncodified and developed over time" and is "constantly evolving," especially in a dynamic organizational environment.

Software development from a postmodern perspective as described by Robinson et al. (1998, 368) is "an amalgam of various analytical, design, implementation, predictive and managerial activities, embedded in dynamic social systems, replete with already developed sites of cooperation and conflict." This view of software development explicitly recognizes that a "multiplicity of cofactors" are always at work in the software development process, any combination of which may be problematic and may be at the root of software failures, whether during development or after implementation. Thus in software development the complexity of the systems problem and its real-world organizational context must all be accounted for—from the requirements-determination phase through to the implementation. This view of software development pays the requisite attention to the management, the organizational, and the non-technical dimensions of the software development process, areas that the modernist approach tends to ignore.

In the remainder of this chapter we briefly review the extent of software development project failures, then pursue the nature of the problem. We end with a detailed description and analysis of abandoned software projects. *Abandonment* refers to the cancellation of the projects prior to their complete implementation in the organization, and thus can occur at any juncture in the systems development process.

Motivation for the Book and Readership

The motivation for this book was a desire to understand the causes of the software crises. Many authors have pointed to the existence of these crises. McFarlan (1981), in an article outlining a portfolio approach to IS development in organizations published in the *Harvard Business Review*, provided anecdotal evidence of the extent of the problem. He documented the cost overruns and schedule delays of even well-structured and routine systems development projects such as payroll projects in established companies. Glaser (1984, 45) offered further support for McFarlan's claims, when he reported in an article dealing with the management of computer projects that "in the computer industry . . . [project] objectives are missed and schedule and cost targets overrun with distressing regularity and, at times with equally distressing results." But a search for published empirical studies on the problem yielded no results. Consequently, in the late 1980s I undertook a comprehensive research effort to empirically examine the nature and extent of the problem in both public and private organizations. Since then a number of books have been published on the subject, but they have all dealt with specific cases of project failures (Glass 1998; Flowers 1997; Yourdon 1997; Sauer 1993). In the 1990s various studies of an empirical nature were published that shed more light on the widespread nature of the problem and showed the extent of organizational resources consumed by failed projects (Ewusi-Mensah and Przasnyski 1991, 1994, 1995; Ewusi-Mensah 1997; Johnson 1995; OASIG 1996; Cole 1995; Standish Group 1995, 1998). But more attention to the root causes of the project failures is needed.

By focusing on the underlying causes of the software crises, this book tries to fill this gap. I attempt to convey in one volume what I have learned thus far from the published empirical literature, and what that literature says about management and organizational issues in particular. The book does not address issues of systems failure after their implementation in organizations. Lucas's book *Why Information Systems Fail* (1975) provides an early window into the problem of implementation, which he determined from his analysis of a "large amount of data" to be primarily "organizational behavior patterns" (p. 2). I believe that an understanding of the socioorganizational and managerial cofactors that cause abandonment may provide additional insights into the process of software development and may help minimize the future incidence of software project development failures.

The book is primarily addressed to software developers in industry and academia. I believe software practitioners will benefit from the discussion and analysis of the root causes of software project failures covered in this book. For students of software engineering at all levels, the book offers a picture of the plurality and diversity of factors involved in software development, outside of the technical "prescriptive and deterministic" view of software development typically covered in the software engineering and computer science curriculums. This will hopefully provide students with a broader perspective and appreciation of the complexity of the cofactors that are the real bane of software development projects. Such exposure early in their professional training will better prepare students to tackle software development projects in their careers. I believe that the comprehensive analysis the book provides of the factors that make for successful software projects will give professional software developers a deeper appreciation of the complexities of software development as well as insight into improved ways of handling these factors at different stages of the development process. Over time, as software developers learn from the failure factors discussed in this book, the art and practice of software development may be improved industrywide. The secondary audience for the book consists of senior management and executives responsible for software development projects. Senior managers need to be aware of the nontechnical issues associated with management matters in software development and of the role they can play in minimizing the risk of software development failures in their organizations.

Abandoned Software Projects

Abandoned software projects are failed projects because organizations do not routinely invest substantial organizational resources in a project unless certain criteria are met. First, an organization has to have identified a need for the system within the framework of its operational and strategic goals; second, it has to have determined that it has the requisite capabilities to get the system developed and implemented. Under such a scenario, when the organization decides to cancel the project later in the software development process, it can reasonably be surmised that the project was a failure because the organization did not achieve its original objectives with respect to the project. An enumeration of the factors that may contribute to the decision to cancel a project, the intensity of each factor, and the extent of their contributions to the cancellation decision is the focus of this book. The extent to which the factors are or may be repeated in other projects in different organizational settings or environments is equally significant in helping to shed light on the totality of the circumstances at the heart of abandoned projects.

In its description of software failures, the Standish Group (1995) differentiates what it considers "challenged" software projects from "impaired" ones. The "impaired" software project is eventually "*canceled* at some point during the development life cycle" (emphasis added). The "challenged" project is not canceled; rather, it may eventually be completed, albeit over budget, over schedule, and possibly with limited functionality. Thus the distinction between the two types of software projects has to do with whether a project is canceled or not. There are, in fact, many similarities between software failures in general and abandoned software projects. The abandoned software project may be a consequence of a perceived failure of the project prior to its full implementation. Thus in anticipation of that fact, the stakeholder group—that is, senior executives with the responsibility for safeguarding organizational resources, sometimes in consultation with the other stakeholder groups—decides to "pull the plug" on the project. In this sense, we can characterize software project abandonment as an endemic part of the much broader organizational problem of software failures described above (Ewusi-Mensah and Przasnyski 1991).

Boehm (2000), however, challenges the Chaos report's (Standish Group, 1995) conclusions by arguing that some project terminations are justified and, in fact, even necessary to safeguard organizational resources. He arrives at this conclusion on the basis of personal experience, knowledge, and "review of about 20 term papers per year on failed industry projects," among other sources of information. He lists the sources of project termination in the Chaos report and provides explanations as to why termination was desirable in each case. His basic concern is that project termination should not be used as a scapegoat to damage the careers of project managers who might in the future be more likely, as a result, to continue with projects whose strategic and competitive value to a company may have diminished as a result of changes in the project's original assumptions. Consequently, Boehm argues that not all project terminations should be equated with project failures, especially in rapidly changing technological, organizational, and market environments. Project terminations of 31 percent, in his view, should not be considered too high in dynamic development environments, as opposed to stable environments.

While I share Boehm's basic concerns, I feel that project terminations of 31 percent are, in fact, indicative of fundamental problems that require thorough investigation. It is not satisfactory to suggest that such high termination rates are the risk-acceptance rate for undertaking software development projects, even in a rapidly changing development environment. I believe that proper understanding of the multiplicity of cofactors at work in software development projects, as well as knowledge of the risk-acceptance rate of particular organizations, will give software developers, project managers, and their senior management and end users a deeper appreciation of the uncertainties of software development and provide for better decision making in project selection and management.

Subsequent chapters shed light on this looming problem of abandoned software projects in organizations. In particular, they discuss the variety of factors contributing to decisions to abandon software projects and show how abandonment decisions are made in practice. What characteristics do abandoned software projects have in common? What benefits, if any, do organizations derive from past decisions to abandon projects? And finally, what guidelines and lessons can be offered to management to aid in software project selection and project management to minimize the frequency of abandonment decisions in organizations?

Types of Abandonment

What, in essence, constitutes software project abandonment? Software project abandonment occurs whenever senior management decides, for whatever reason, to discontinue temporarily or retire permanently a software project under development. Three types of software project abandonment are identified by Ewusi-Mensah and Przasnyski (1991) and will form the basis of the classification in this discussion.

Total Abandonment

The first and most frequent case of software project abandonment is total abandonment. *Total abandonment* describes the instances where there is complete termination of all activity on the software project prior to full implementation. This classification is analogous to the "impaired" project category of the Standish Group's three-resolution categorization, consisting of completed, challenged, and impaired projects. The incidence of this type of abandonment reported in published studies ranges from a low of 10 percent (Cole 1995) to a median of 31 percent (Standish Group, 1995) to a high of 40 to 44 percent (OASIG, 1996; Ewusi-Mensah and Przasnyski 1994). It is also one of the most costly expenditures organizations incur in their software project portfolio, with estimates ranging in the area of several billion dollars for both public- and private-sector organizations (Standish Group, 1995; Wysocki 1998; Heinrich 1998).

Substantial Abandonment

The second category of abandoned software projects is described by Ewusi-Mensah and Przasnyski (1991) as substantial abandonment. *Substantial abandonment* refers to instances in the software development process where a major truncation or simplification of the project occurs prior to full implementation to make it radically different from the original specifications. This category is approximately analogous to the

Standish Group's "challenged" project classification, because the "truncation or simplification of the project" may result in "fewer features and functions than originally specified," as the Standish Group study suggests. Substantial abandonment may also be similar to the "software runaway" description offered by the KPMG study, and may even come under the rubric of Glass's (1998) more expansive definition of what constitutes "software runaway." The KPMG study found that 62 percent of the respondents had experienced a "runaway project" and that the number had not changed from a previous study five years earlier. In Glass's view, abandonment is not much of a solution to the problem of software failure or runaway. Judging by his somewhat dismissive remark of "some remedy" in commenting on abandonment of the project as discussed in the KPMG study, total abandonment should only occur if "it stops or rapidly slows project bloodletting, such as money and resources" (Glass 1998, 242). Still, Glass grudgingly concedes that "most of our case studies in Part 2 [i.e., of his book] resulted in some form of project cancellation." Thus, by Glass's own description of what constitutes a "software runaway," the abandoned project can belong to either of the above categories—that is, total or substantial abandonment.

In substantial abandonment, projects may still be continued or carried to completion even if cost overruns and schedule delays occur, so long as the reduction in systems functionality will enable the organization to salvage some of its investment in the project. This is perhaps the most significant distinction that can be made between substantial and total abandonment. However, in the case of total abandonment, reduction in systems functionality may not achieve the desired goals even if management allows for the cost and schedule overruns, thus making complete termination of the project the only viable option.

Partial Abandonment

The third category of abandonment is identified by Ewusi-Mensah and Przasnyski (1991) as partial abandonment. *Partial abandonment* describes the case in which a reduction of the original scope of the project is made without entailing major changes to the project's original specifications prior to its full implementation. This classification provides a limited, indeed a restrictive view of abandonment to reflect in essence some reduction in systems functionality but nothing so significant as to drastically affect the original project specifications. Such cases seem to occur frequently in practice, albeit with some cost overruns and schedule delays. In fact, the KPMG study (Cole 1995, 4) provides data to indicate that about 30 percent of "software runaways" can be attributed to "reduction in scope of [the] project." In addition, this class of abandoned projects is probably in general to be expected, because organizations would rather make changes to the scope of their projects than risk total abandonment or even substantial abandonment if they believe such changes would enable them to achieve their original project's specifications. This would be a more rational use of organizational resources in the development of software projects than the other two alternatives would be.

All three types of abandonment occur in organizations in both the public and private sectors. However, of the three types of abandonment, the most costly and at times the most disastrous is the total-abandonment category, as many reports bear out (see, for example, Cole 1995; Capers Jones 1995; Glass 1998; Standish Group, 1995; Wysocki 1998; Gibbs 1994; OASIG, 1996; Heinrich 1998; "PG&E Dumps CIS Project," 1997; Ewusi-Mensah and Przasnyski 1991, 1994).

Table 1.1 provides a summary of the data on abandoned software projects from four published studies, and figure 1.1 summarizes the same

Table 1.1
Summary data on software project–abandonment

	Type of abandonment		
Published study	Total (%)	Substantial (%)	Partial (%)
Standish Group (1995, 1998)*	31 (28)*	52.7 (46)*	Not available
KPMG (Cole 1995)	10	28	24
OASIG (1996)	40	25	80
Ewusi-Mensah and Przasnyski (1994)	44	16	9

*Reported in Whiting 1998

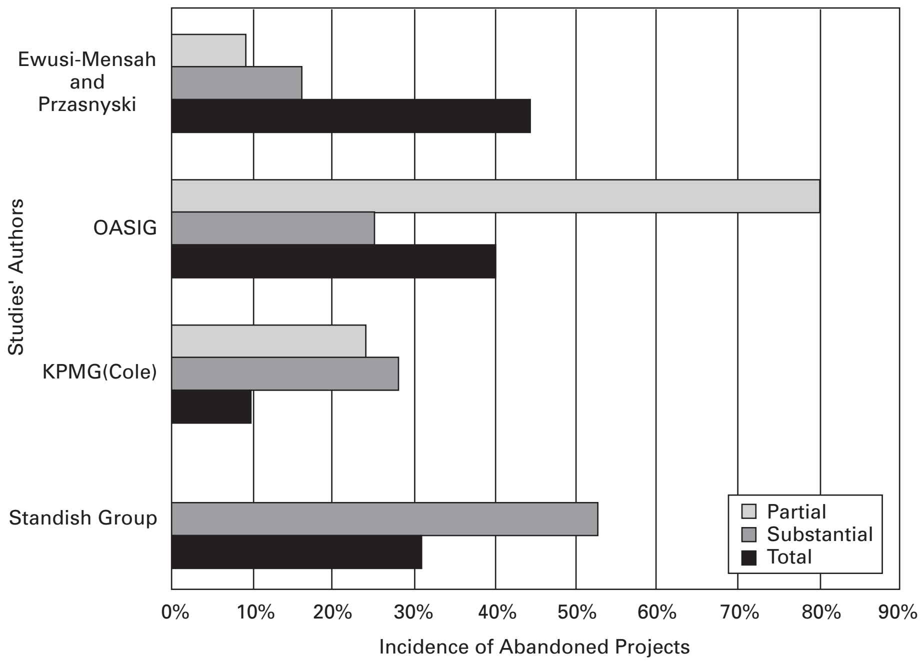

Figure 1.1
Summary data: Abandoned software project studies

data in graphic form. The totals do not add up to 100 percent because the incidence of abandonment reflects different projects in different organizations.

The data deal only with abandoned or "runaway" projects; successfully completed projects are not included in the data analysis reported. The total-abandonment figure from the Standish Group, with a total sample size of 365 respondents in the United States, breaks down as follows according to the size of the organization: 37.1 percent of most projects in medium-sized companies (i.e., companies with annual revenues of between $200 and $500 million); 29.5 percent in large companies (i.e., companies with annual revenues of more than $500 million); and 21.6 percent of projects in small companies (i.e., companies with less than $200 million in annual revenues) (Standish Group, 1995). No comparable figures are provided for the substantial-abandonment projects.

The KPMG study, based on a sample size of 120 organizations responding in the United Kingdom, does not provide any equivalent statistics other than to indicate that 62 percent of the respondents experienced runaway projects as defined in the study. The 10 percent total-abandonment figure was the "remedy" applied to the runaway projects by some respondents, as was the 28 percent given for substantial abandonment, defined as "reduction in scope of project." Some other remedies applied to the runaway projects were "more money" (45%), "more people" (55%), and "more time" allotted to the projects (80%). Among other measures, more than 55 percent also instituted "better project management procedures."(See Cole 1995, 4, for further details.)

The OASIG (1996, 12) study was also based in the United Kingdom and "drew on the expertise of leading management and organizational consultants and researchers" covering about 14,000 organizations in all major sectors of the economy. There is no detailed statistical breakdown along the lines of those provided by the Standish Group and the KPMG study. The OASIG study states that around 40 percent of developments fail or are abandoned; "less than 25% properly integrate business and technology objectives" (presumably this deals with the scope of the project); and "about 80% of systems are delivered late and over budget."

Finally, the Ewusi-Mensah and Przasnyski (1994) study was based on 82 responses from Fortune 500 companies in the United States that had experienced abandoned software development projects. In this study, the three categories of abandonment were defined and respondents were asked to indicate which of the categories appropriately fit the description of the abandoned project in the organization. The figure that provided for each category of abandonment in table 1.1 was based on the total number of respondents who indicated that they had experienced a specific type of abandonment. Of the 78 usable responses received, 34 respondents indicated that they had experienced total abandonment, about 44 percent; another 13 respondents indicated they had experienced substantial abandonment, about 16 percent; and yet another 7 respondents indicated they had experienced partial abandonment, about 9 percent. Finally, 24 respondents indicated they had experienced no abandonment—about 31 percent. Further discussion of the results of this study is provided in later chapters, notably in chapters 4 through 6.

The data from the above studies confirm that abandonment is not confined to any particular market segment or type of organization, nor is it "associated with any particular business sector or size of organization," as the KPMG study (Cole 1995, 3) concludes. In general, software abandonment occurs "when problems arise in perceiving, analyzing, designing, or configuring the system objectives; when the technological basis for the system and its behavioral, political or organizational (including economic) issues directly or indirectly affect ways that could bring the project to a successful completion, within the estimated budget and/or schedule constraints; and/or when the organizational environment factors combine to reduce the project's expected benefits or increase its expected costs" (Ewusi-Mensah and Przasnyski 1991, 68).

Failed Software Projects: Two Cases

I briefly describe and analyze two reported cases of failed software development projects—Highmark's HighBAR project and Hershey Foods' ERP project—to illustrate the complexities of the underlying managerial and organizational issues that contribute to project failures.

Highmark's HighBAR Project Highmark, a $5 billion health insurance company based in Pittsburgh, Pennsylvania, contracted with the KPMG consulting group in May 1999 to develop an electronic billing and accounts receivable system for the company at an estimated cost of $15 million. The project was to be completed by the end of the first quarter in 2002.

The project, named HighBAR, was divided into four phases, and work on the first phase was begun in July 1999. In April 2001, KPMG requested $8 million to complete the first phase in addition to the $12 million already paid by Highmark for what was only 20 percent of the project. When Highmark refused payment, KPMG ceased work in June 2001. Highmark sued KPMG on charges of fraud, malpractice, misrepresentation, and breach of contract (Mearian 2001a, 2001b).

The allegations Highmark made in its suit dealt with the failure of KPMG to properly inform Highmark of progress on the project. Specifically, Highmark alleged that KPMG "failed to 'satisfactorily' complete several aspects of the project, including database design, software design specifications and a detailed application architecture "(Mearian 2001a, 2). Coding was allowed to proceed on the project even though the design was still undergoing revisions, which resulted in several thousand hours of coding having to be redone due to changes in the design. The net result was that the project was more than a year behind schedule and several million dollars over budget. The project, according to Highmark, never progressed beyond the second stage. KPMG responded that the project was a collaborative effort between itself and Highmark and that the latter failed to carry out its part of the collaboration "to ensure a successful implementation of the payment system on time and on budget" (Mearian 2001a, 2).

HighBAR illustrates several of the managerial and organizational factors identified as contributors to abandoned projects. In particular, although there was a consensus on the overall project goals and objectives, the failure of Highmark senior management to stay engaged with the project development may have contributed to KPMG's alleged failure to keep Highmark informed of progress with respect to project costs, labor hours, and deadlines. A full 80 percent of the estimated cost of the project—that is, $12 million—was paid to KPMG for what was less than 20 percent of the project, which was alleged to have been completed in an unsatisfactory manner (Mearian 2001a). The changes in the design, probably due to changes in requirements, may have played a role in this scenario, because several thousand hours of coding had to be jettisoned. Cost overruns and schedule delays are generally symptomatic of more fundamental problems in the project life cycle. For example, Highmark alleged that KPMG fraudulently claimed it had the requisite knowledge and experience in object-oriented analysis and design to enable it to successfully complete the project. KPMG's alleged misrepresentation of its experience may have led to the frequent changes in the design, which in turn resulted in substantial recoding. Highmark senior management must also share the blame for failing to thoroughly investigate the technical experience and expertise of KPMG. In addition, had management remained engaged and asked persistent questions early in the life of the project, some of the problems might have been uncovered earlier and dealt with before they mushroomed into the reported fiasco.

Hershey Foods' ERP Project Enterprise Resource Planning (ERP) systems is a comprehensive applications systems package that organizations use to automate their back-office operations. For example, it permits order processing and distribution functions to be fully integrated for smooth operation of the business. It is with these ambitious goals in mind that Hershey Foods embarked on its ERP project in 1996: to integrate the disparate legacy applications systems for handling order processing, inventory, and human resource operations at Hershey into one computing platform. Hershey engaged the services of four different consulting firms to work with its IT department to install SAP AG R/3 software in conjunction with software from two other vendors—Manugistics Group for its planning applications software, and Siebel Systems for its pricing-promotion-package software. The project was estimated to cost between $112 and $115 million (Osterland 2000; Stedman 1999).

In July 1999, with implementation already three months behind schedule, Hershey went live with the conversion from the old legacy system to the new SAP AG R/3 system. Two months after the implementation was supposed to be completed, Hershey was still finding fixes to the system. The company's sales dropped in its third quarter—its busiest sales season of the year—by more than 12 percent. With it, there was a corresponding drop of more than 18 percent in revenues compared to the previous year; Hershey experienced a 29 percent increase in project inventory; and its mean delivery time for orders increased from five to twelve days, as reported in several publications. (See, for example, Osterland 2000; Collett 1999; Stedman 1999; 2000; Songini 2000; "ERP Stumbles," 2000; Wheatley 2000; Nicholson 1999.)

The implementation period was originally estimated at four years but was later shortened to thirty months—for unspecified reasons—despite the fact that the R/3 software modules were to be integrated with software from two other vendors. The consequence of the compressed implementation time frame was that insufficient time was devoted to test runs of the modules and training of the users. The implementation problems often associated with ERP systems reported in the media are more often than not attributed to these overly ambitious time frames and inadequate user training, which combine to create havoc in the organization when the system is fully activated. Because of the complexity of the R/3 software, any implementation that involves customization with other vendors' software frequently has unintended and/or unanticipated consequences that are difficult and time consuming to find and correct. It took about a year for Hershey to fix all the related implementation bugs and to get the company back to normal operating conditions and profitability (Songini 2000).

In the Hershey ERP R/3 project problems, we see echoes of FoxMeyer's Delta project, analyzed in detail in chapters 4 through 6. For example, the drastic reduction of the implementation time from four years to thirty months is generally identified as a major contributor to the problems the project experienced when it went live. FoxMeyer experienced similar problems, which contributed to its bankruptcy. However, Hershey was able to avoid a similar fate because of its size and the superior resources it could summon to deal with the emergencies resulting from the implementation failure.

Internet- and Web-Based Software Projects (and Project Failures)

With the collapse of a significant number of e-commerce Internet- and Web-based companies in 2000 and 2001, there has been a general perception, often erroneous, that the failure of the companies can perhaps be blamed on the technology. This misperception is unjustified because, as growing evidence reveals, the failure of the companies was due more to the failure of the underlying business models on which the entire future and competitive strategies of the companies rested. Indeed, as Berghel (2001, 17) has argued, "The problem that caused the Y2K e-commerce meltdown is . . . an over-reliance on technology to overcome the weakness of a bad business model." Clearly, the often-irrational foundation on which the companies derived their wealth based on capitalization of their stock-market shares even when they were freguently operating in debt was not a failure of the technology. In fact, it was the promise and capabilities of the Internet- and Web-based technologies that tended to sustain what in a normal business environment would have been untenable business practices. The lure of the Internet and the Web, fueled by speculations in the financial markets, gave free reign to the rise of the companies' stock value until investors realized that the trend of ever-increasing share prices could not be sustained forever. For example, a company like Amazon.com had a share price of $427.12 at its peak market valuation on April 27, 1999, and three years later a share price of $13.96 on April 23, 2002 (<Yahoo.finance.com>), even though it had never turned a profit, because investors were led to believe that the revolutionary nature of the underlying Internet- and Web-based technologies was the wave of the future. And so despite evidence to the contrary, investors continued to invest heavily in the company's future. Amazon's operating business paradigm was widely replicated in other e-commerce companies; these eventually collapsed because the business models were not sustainable, notwithstanding the capabilities of the Internet- and Web-based technologies.

The Internet- and Web-based technologies that formed the core of the companies' business strategies are therefore not the culprits in the failures of those companies. The flawed business models and the companies that staked their existence and survival on them produced the failures of the software development projects, which were intended to exploit the e-commerce phenomena. There is nothing innate in the Internet- and Web-based technologies that poses special problems for the software development process. For example, Internet- and Web-based software projects are invariably based on client-server architecture often implemented or deployed in a network environment. A number of the failed software project cases discussed in the book, including Confirm, FoxMeyer's Delta, DIA's BHS, and the Hershey Foods' ERP project, are all based on such an architectural framework. Hence we do not focus on Internet- and Web-based software projects in our discussion, because such an approach would not provide any additional insights into the causes or contributors to the failure of those projects beyond what is applicable to complex software development projects. In addition, as discussed in Ewusi-Mensah and Przasnyski 1991, 1994, some software projects may fail as a result of changes in the organization's structure or competitive business environment; these changes may require a reexamination of the basis for the continued funding of the projects in question. For instance, some projects may fail as a result of mergers and acquisitions; these may call for a reexamination of organizational priorities involving those projects, among others, and for a reassignment of organizational resources to satisfy different priorities. We can therefore surmise that some of the e-commerce software project failures can be classified as similar to the aforementioned types of projects with respect to their complexity, innovativeness, and dynamic development environments, among other factors.

Conclusion

This chapter has examined the nature and extent of software development failures. We have discussed the role software engineering as a science has played in helping to provide solutions to the software crises. We have also described the problem of software abandonment and provided three classifications of abandonment. In addition, we have pointed out the inherent characteristics of software projects that make them susceptible to failure in general and abandonment in particular, and explained why the main focus of the book would be on management and organizational issues in software development projects. The main audience for the book, we suggested, consists of students of software engineering and professional software developers as well as their corporate managers, who should be aware of the organizational and managerial issues that play a significant role in decisions affecting successful project development outcomes. We have briefly described and analyzed two reported cases of failed software development projects in organizations to illustrate the organizational and managerial issues at the heart of most failed projects. We ended with an explanation of why software project failures in Internet- and Web-based companies cannot be blamed on the technologies, but rather must be blamed on flaws in the underlying business models.