Chapter 1

Overview and Introduction

This is a book about managing the development of software-based applications and products. When we say "the development of software-based products," we mean the activities involved in deciding what the product needs to do, commonly called requirements engineering, designing the product at both high levels, sometimes called architecture, and low levels, called detailed design, coding the software, sometimes integrating the software with custom-designed hardware, testing the product, deploying it to users, and maintaining its evolution. So, we include the management of all phases of the development life cycle in our understanding of the phrase "development of software-based products." We also include all types and kinds of software. This covers the range from software that drives static web pages, interactive web pages, and scientific applications to large data management systems based on commercial database management systems to applications that monitor and control real-world distributed systems such as air flight control systems or telecommunication switches and transmission systems that are geographically disbursed. These software applications typically run in service on commercial computers. We also include embedded and real-time software that typically interacts intimately with electrical and mechanical devices and systems and that is frequently loaded permanently (or at least semipermanently) into the hardware devices or systems that it controls. When we develop the latter type of software, we will also need to be concerned about integrating both new software and the newly developed hardware on which and with which it operates.

With the inclusion of this broad array of software types and all phases of the life cycle, it quickly becomes obvious that the same rigidly defined management methods and techniques cannot be used to manage all software development work. In addition, some software applications will involve only a small number of people working together to do the work, sometimes only two or three people. Other projects will require hundreds, or occasionally thousands, of people to do the job. In the industrial environment of Bell Laboratories where I worked for several years, there were typically a few hundred software development projects under way at any one time. The number of people involved in each ranged from four or five up to approximately 1500 on the very largest undertaking. The average number of people involved

Managing the Development of Software-Intensive Systems, by James McDonald

on each project was about 20. So, as with the kinds of work and types of software being developed, the sizes of the teams doing the work also vary tremendously.

This extremely large variation among possible environments in which development managers could be working implies that any method that we will be discussing will need to be appropriately tailored to the environment in which it will be used. Tailoring is difficult and it is almost impossible to be prescriptive about which techniques should be applied under which conditions. However, by the time the readers finish with all of the materials in this book, they will have developed enough understanding and skill so that they will feel confident in their ability to tailor their use of the methods and techniques discussed to their particular environment.

I have found that one of the primary determinants of which techniques are appropriate is the size of the project, measured in terms of the product's size and functionality and the maximum number of people who will need to be employed to develop the product. For example, it should be quite obvious that a development team of four or five people does not require the same quantity or quality of management as would a team of several hundred people. It is likely that some readers of this book will be involved with small projects, involving only a handful of people. Others might be involved in projects that require a few tens of people to accomplish their objectives. A few may be lucky (or, perhaps, unlucky) enough to work with teams of a hundred or more people.

So, how should each reader review the content of this book? I would suggest that you start by temporarily backing away from the environment in which you are currently involved. Try to imagine that you are working with a team of 20 other developers on a project that will require that maximum number of people to accomplish the work. Also assume, for now, that you are starting at the very beginning of a brand new undertaking and that you will have an opportunity to influence not only what will be developed but also how that work will be done and, most importantly for this book, how it will be managed. Most of the examples discussed in this book will be drawn from projects like the one described here. For those projects, we will describe all of the things that need to be done to make them successful. For larger projects, it is safe to assume that all of the things that we discuss will need to be done (along, perhaps, with a few additional things) with great care and diligence. For smaller projects involving a few up to about 20 people, the methods and techniques that we will discuss will need to be selectively scaled back, with some not requiring as much care or thoroughness as we will imply and some that may need none at all. However, while reading, please try to concentrate on the project involving 20 people described above. Toward the end of the book, we will provide some guidelines for "backing off" from some of the more burdensome activities.

Throughout this book, you will read about the four major project management processes of planning, organizing, monitoring, and controlling and how those processes can be applied effectively by software project managers. We will also talk about two underlying processes that the manager needs to use to facilitate the process management processes. They are communicating and negotiating. These process are used somewhat sequentially as shown in Fig. 1.1 with significant amounts of feedback from the processes toward the right in the figure to the processes on the left.

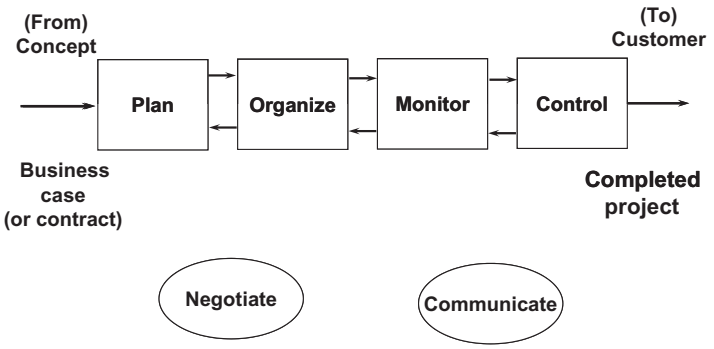

Figure 1.1 Project management processes.

Most product or system development undertakings will start with a concept and, perhaps, a business case or a contract that has been negotiated with a customer or with the management of the parent organization. The person responsible for managing the work needs to plan what will be done, organize the team that will do the work, monitor how the work is progressing, and, when necessary, take control to modify how the work is being done or by whom it is being done. These processes are intended to take the job from concept to delivery of a completed product to the customer. The two supporting activities labeled "communicate" and "negotiate" in Fig. 1.1 are used by the manager in support of the four major project management processes.

In this book, we will try to achieve several objectives. They are to develop

- a broad understanding of how project management can be applied to software development projects;
- an understanding of the estimation methods that are available and the pros and cons of each;
- the ability to properly apply project management methods to the requirements, architecture, design, testing, and delivery of software products;
- an understanding of common organizational structures, team building, and conflict resolution;
- the ability to monitor and to report on the status of software development work;
- the ability to use audits, reviews, and assessments effectively;
- the ability to minimize risk;
- methods for working with partners and vendors;
- the ability to manage projects in complicated but typical, multi-project, and multiple organization environments;
- an understanding of the complications and the costs involved in outsourcing portions of development projects;

- an ability to tailor the project management processes to appropriately match the project and its environment; and
- the ability to review successes and failures and to learn from our experiences.

We will try to achieve these objectives while simultaneously emphasizing the relationship between the management process outlined above and the underlying technical and business processes used by people doing the actual development work.

In addition to reading the text that is provided in each chapter, there are two significant activities in which readers might want to participate. These are, first, a series of short case studies, each of which is associated with the materials in the chapters where they appear, and, second, a major longer case study (Case Study 2 at the end of Chapter 2) in which readers can go through all of the steps involved in developing a project plan for a real project. Both the case studies and the project are best done by groups of four to eight participants, so they are best suited for use in classes and workshops in which all participants are going through the text together and working on the case studies and the project in groups. However, the case studies may also be of use to an individual reader who would like to get some real-world experience in thinking about how the content of the text might be related to situations faced by managers of software-based development work.

In the remainder of this chapter, we will address our first objective. That is "developing a broad understanding of how project management can be applied to software and software/hardware development projects." We will do this by discussing a model that will provide a context for the remainder of the book, some definitions of terms, characteristics of software projects, managers' roles, software life cycles, processes in the life cycle, and process management.

1.1 OUR MODEL

Throughout this book, we will assume that development of a new product or expansion of the capabilities of an older product follows the model shown in Fig. 1.2. In this model, we assume that product or system development work requires the successful completion of the business and technical tasks shown in the lower portion of the figure. They include the work required to develop requirements and to design and implement the necessary software and hardware, and the associated business processes such as development of a business case, a proposal, or a contract. This is work done by people using whatever business and technical processes the organization has decided to use to accomplish its work. Sitting above these business and technical processes is a box that contains the project management processes. These are also tasks that are done by people using a variety of management methods, some of which include the planning, organizing, monitoring, and controlling processes that make up a major portion of the material discussed in this book. These processes are also executed by people who can make use of a wide variety of management methods.

We will assume that most development work starts with a customer who is making use of a presently available product or system. That customer, whether they

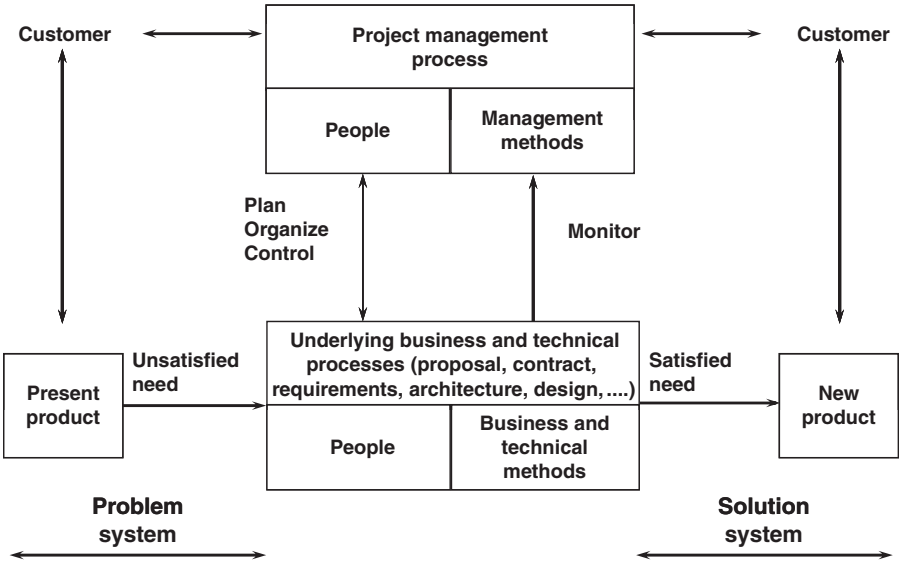

Figure 1.2 Model assumed throughout this book (from Crawford and Fallah 1985, reproduced with permission of IEEE © 1985 IEEE).

are inside or outside the organization in which the work will be done, is assumed to have a problem or an unmet need to which we will refer as the problem system. The customer interacts with some of the people who execute both the management and the underlying technical processes shown in the upper and lower paths shown in Fig. 1.2. The people in those blocks then execute their processes, sometimes over a short period of time, such as a few weeks or months, or over a longer period of time, sometimes extending to as much as several months or even several years. At the end of the overall development process, the development team delivers a new product or system to the customer that satisfies their needs. We will call that product the solution system.

Two important parts of this model are the lines connecting the project management process box with the underlying business and technical processes box. The reader should note that the line labeled "plan," "organize," and "control" is two headed, indicating that information flows both ways along this link. Those processes require that information flows both from the project management process to the underlying business and technical processes and that information flows from the underlying processes to the project management process. The one-headed arrow labeled "monitor" indicates that for monitoring, the primary flow of information is from the underlying business and technical processes to the project management process.

This model is a very simplified description of what usually takes place in practice. When we start talking about development models in more detail, we will discuss the waterfall model, the spiral model, and the agile model that bring out more of the gory details of real development processes.

1.2 DEFINITIONS

Most books like this one have titles like "Software Project Management." They contain many terms drawn from the traditional project management body of knowledge which had its early roots in the management of the development of large complex systems for the U.S. Department of Defense and for commercial and public works construction projects. In this book, we will make use of many standard project management terms. So, let us get started by defining a few of them. The first one we will define is the word "project." A project is an activity performed by people, sometimes with assistance from a variety of tools, like backhoes, cement trucks, or, in our case, computers, compilers, software development environments, test equipment, and so on. Most projects are faced with a variety of constraints like the availability of funds, the availability of people, or the need to have the new product or system available by a specified date. A project can be described by a set of tasks and processes where tasks are activities done by people and processes are the connections between those tasks that describe the order in which the tasks must be accomplished. Some of the tasks are typically dependent on the completion of other tasks, while some are completely independent of other tasks. In the case of software development, a task might be something like coding and compiling a program. Another task might be testing the program to determine if it works properly and, if not, what is wrong with it. Obviously, the testing task cannot begin until there is some code that has been written and compiled, so there is a process relating the testing task to the coding and compiling task. Projects need to be planned, organized, monitored, and controlled, and we will spend at least a chapter discussing both the meaning of those terms and how the manager can optimally manage those processes.

All of these tasks and processes are similar to other activities in which humans engage. However, there are two characteristics that make projects different from those other activities: (1) a project is a temporary endeavor, meaning that it has definite starting and ending points; and (2) a project creates a unique product or service. After a project ends, the world will be different than before the project was started.

These characteristics distinguish projects from repetitive or cyclic operations. Cyclic operations are processes that are repeated over and over again, as, for example, in a factory that produces pencils, refrigerators, automobiles, or even airplanes. Those cyclic processes lend themselves relatively easily to quantification and to statistical process control, making them susceptible to the use of a variety of metrics that can be used by the manager who is responsible for them. Software-based development work, while we continually try to make it a repetitive process, is not cyclic. It has intrinsic characteristics that make it very difficult to apply quantitative techniques to its management in the same ways that these techniques are used to help manage cyclic processes. We will talk about a few commonly used software metrics, but we will find that using more than a very few basic measurement methods is extremely difficult and unreliable.

Let us think about building a skyscraper. The underground portions of each skyscraper are somewhat unique, with their layout and structure being dependent

upon things like the ultimate height of the building, the soil and geologic conditions where it is being built, and the creativity and discipline of the architects and structural engineers who designed the building. The first and second floors usually contain some unique facilities like escalators, spaces for retail establishments, and elevator lobbies. However, after the third floor, all of the floors are quite similar, with the fourth floor being just like the third floor, the fifth floor being just like the fourth floor, and so on. Once construction reaches the third floor, the manager who is supervising construction can tell the work crew, "The fourth floor needs to be just like the third floor, the fifth like the fourth and just keep going until I say 'Stop.'" Then the roof is usually unique. So, while the construction of the overall skyscraper does, indeed, produce something unique, the rising of floor upon floor is more like a repetitive or cyclic process.

The development of software and its corresponding hardware is different from our skyscraper project in that, unless we are very creative, or sometimes very rigid, in the application of our development processes, there is usually minimal repetition from one part of the project to the next. Another characteristic of software-based development projects that makes them difficult is the fact that it is extremely difficult to look at a piece of software and to know the status of the work involved in developing it. We will discuss that issue in more depth when we talk about the monitoring process in Chapter 6.

The manager who is responsible for supervising development work is usually charged with meeting or exceeding the expectations of customers and other project stakeholders and with negotiating those expectations so that the work can be done within agreed upon budget, time, and personnel constraints. The manager is usually faced with the difficult task of balancing competing demands among customer needs and wants (usually called requirements and sometimes scope), the project schedule (time), the cost of the work, customers with competing requirements when a product is being developed for use by multiple customers, and quality of the design. Finally, the manager is sometimes faced with identified requirements as well as with unidentified requirements. Unidentified requirements are created when customers or stakeholders have made assumptions about what the product will do or how it will do it but have not communicated those assumptions to the development team.

In the preceding paragraph, we have mentioned "stakeholders." Stakeholders are the set of everyone who has an interest in the work of the development team. They include customers such as those who will ultimately be hands-on users of the product, a manufacturing organization that might need to load the software onto a commercial or custom-designed hardware configuration, sales and marketing people who will sell the product if it is being developed for sale to multiple customers, the higher management of the organization that is developing the software, and finally, and perhaps most important, members of the team that is doing the design and development work.

Throughout this book, we will be talking about both project managers as well as software organization managers. Sometimes these will be different people and sometimes they will be the same people. We will make that distinction more clearly when we discuss the organizing process in Chapter 5.

Let us first talk about what development project managers typically do on a day-to-day basis. Project managers have to understand the environment in which they and their teams are working. This includes the problem system that we discussed earlier and the stakeholders for the work that is being done, including customers, supporters, and detractors. They need to understand the business rationale for the work that is being done. And they need to understand the goals and rules that will be applied to the work. By the goals and rules, we mean explicit rules (such as the need for profit in an industrial environment or the need to meet the budget as closely as possible in a government environment), constraints (such as schedule and scope and the relationships between those constraints), and what information can and cannot be shared with customers. It is normal for some of these rules to be explicit and well known, but sometimes, the rules may be implicit and never openly discussed.

The project manager must take the lead in planning the work that will be done. We will discuss planning extensively in Chapter 2. However, it can be briefly described as identifying every task that must be done, clearly defining those tasks, allocating resources to tasks, scheduling the tasks, and assigning responsibility for completion of tasks for the overall project.

He or she needs to negotiate a commitment including the budget, the schedule, the features that will be delivered, and the people who will do the work. This is the most clearly defining role of a project manager. We frequently hear that a project manager has been assigned to a project for which the budget, the schedule, the scope, and the team membership have been predetermined and the project manager does not have the freedom or the ability to negotiate any of those constraints. If that is the situation, then the person we are discussing cannot be called the project manager. They can only serve the role of a clerk, monitoring the status of the project and reporting its status. Real project managers must have the ability to negotiate the budget, the schedule, the scope and team membership, and the freedom to negotiate changes in at least one, and possibly more, of those constraints when the need arises. If the project manager does not have this ability, they are not the project manager and they can be assured that they will not be successful in their job. If you ever find yourself in that position, you have two options. You can try to locate a member of the management team who, with your assistance, has the ability to negotiate these constraints and to pair up with that person to create the project management team. Together you could be called the "real project manager." Your second choice, if you cannot accomplish the first, is to leave the position and to look for another job. So, when we talk about a project manager in this book, we are referring to someone in the organization who has that ability to negotiate, someone who knows enough about the technical aspects of the work being done to understand it, and someone who also knows enough about project management to execute the more routine responsibilities of a project manager.

A project manager needs to organize the work including staffing the project and considering the skills, knowledge and abilities of the people who will be, or are, on the team, and the structure of the organization in which the work is being done. They need to clearly define and manage the definition of interfaces, both technical

interfaces, which will be implemented in the product being developed, and interpersonal interfaces through which intra-project communications will take place. And, finally, they need to make responsibilities clear.

They need to execute the plan by monitoring the status of the work, identifying and addressing issues as early as possible. This means tracking staffing, cost, progress of the work, and performance of the product being developed to identify discrepancies between the plan and the project status. Then, when discrepancies are identified, he or she must take control to resolve the underlying issues promptly.

They need to complete the project on time with appropriate quality, within budget and with features that satisfy the customers.

Throughout all of this work, the project manager needs to communicate goals, plans, project status, and changes to all of the stakeholders. We have been talking about things that the project manager does. You might be thinking what is the difference between what a project manager does on a day-to-day basis and the four major project management processes that we discussed earlier. The project manager is a person, an individual (or sometimes an individual acting with the assistance of others) who is responsible for executing the project management processes including planning, organizing, monitoring, and controlling. Ideally, the individual project manager would be a visionary who can formulate where the project is headed and can communicate that vision to stakeholders; a rebel who is willing, when necessary, to say "no" when one or more of the stakeholders is pressuring them to do something that is inappropriate or undoable; a planner who is able to get members of the team engaged in doing the difficult planning work that some team members invariably will resist; a team builder who can get team members and important stakeholders to work together creatively and efficiently; a leader who can help the team members stay motivated while heading as directly as possible toward the team goals; a communicator who can write to and talk with stakeholders on the team, in the organization, and outside the organization in which the work is being done; and a psychologist who sometimes needs to try to understand the reasons for behaviors of team members, bosses, and customers who may not appear to behave rationally.

That ideal project manager is often hard to find, but that is what we will be working toward throughout the remainder of this book.

1.3 LIFE CYCLE MODELS AND THE DEVELOPMENT PROCESSES

Almost any book or paper on general project management will start with, or at least assume, that a project timeline looks like the one shown in Fig. 1.3. The project starts with the development of an idea that states in very broad terms what will be done and what will be achieved. This takes place during the concept phase of the project. Then someone takes a first cut to determine whether the project is feasible. A team starts to define the product by developing requirements. The product is developed. Then it is deployed and goes into operation.

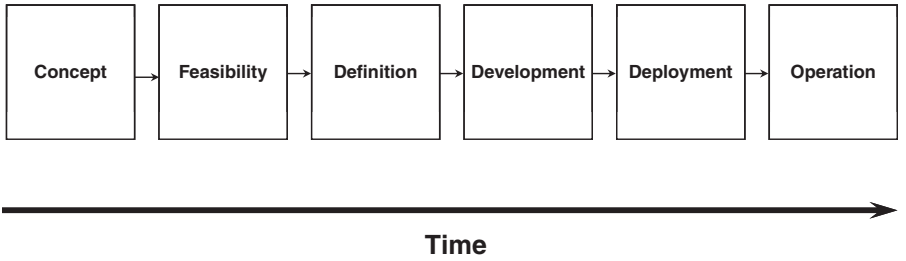

Figure 1.3 Project phases and the project life cycle.

The major outputs of each of the phases shown in Fig. 1.3 are the following:

- Concept phase
  1. A statement of the opportunity
  2. An estimate of the resources to complete the feasibility phase of the project
- Feasibility phase
  1. A preliminary business case or a proposal
  2. Formation of a project team
  3. A draft of an architecture document
  4. A preliminary project plan
- Definition phase
  1. Baselined requirements
  2. A baselined architecture
  3. A baselined project plan
  4. An approved final business case or a signed contract
- Development phase
  1. Hardware designs captured, software coded, and both baselined.
  2. Unit, integration, system, and beta testing completed
  3. Customer documentation and training materials completed
  4. Information for reproduction transmitted to production facility
- Deployment phase
  1. Customer plans for use completed
  2. Production and customer support in place
- Operation phase
  1. Product generally available
  2. Product in use by customers
  3. Development team may be adding features and capabilities
  4. Project closed out

Of course, anyone who is familiar with the development of software-based products or applications knows that software is never developed using the exact phases and sequential life cycle shown in Fig. 1.3. The first attempt to specify a development process that was more realistic for the development of software was the waterfall process shown in Fig. 1.4. That process was described by Winston Royce in 1970

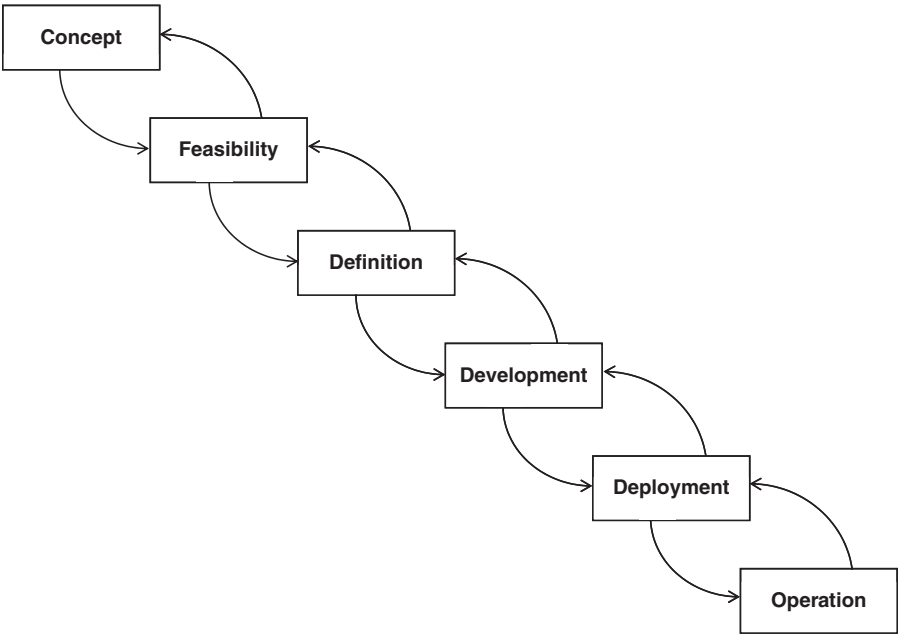

Figure 1.4 The waterfall model.

(Royce 1970). It is based on the traditional phased product development process of Fig. 1.3 but recognizes that it made more sense to iterate between adjacent phases to allow for some feedback from later phases to impact completion of earlier phases. It also allowed work on the succeeding phase to begin, to the degree possible, before the immediately prior phase is completely finished, thus shortening the interval from start to finish as compared to the linear depiction of Fig. 1.3.

Several years later, an alternative process, known as the spiral model, was introduced by Barry Boehm (1988). It is illustrated in Fig. 1.5.

The intent of the spiral model was to address some of the weaknesses of the waterfall model. It allows for the use of a waterfall model in that it calls for determination of objectives, specification and evaluation of alternatives, design and development of the product, and planning of future work. In this model, the distance from the origin is a measure of cumulative cost to date and the angular dimension represents progress in completing each cycle of the spiral. Notably, the spiral model explicitly specifies repetitive risk analysis steps and the use of prototypes, simulations, and modeling to help reduce risk. It also calls for frequent cycling during the early phases of the project to provide increased confidence that the concept, the requirements, the development plan, the prototypes, and the design are appropriate.

The third major contribution to the characterization of software development models was the development of the agile software development process. This method is based upon the content of a document that was developed and signed by a group of software consultants in 2001 (Beck et al. 2001). That document is called the Agile Software Development Manifesto. It includes four broad statements upon which the

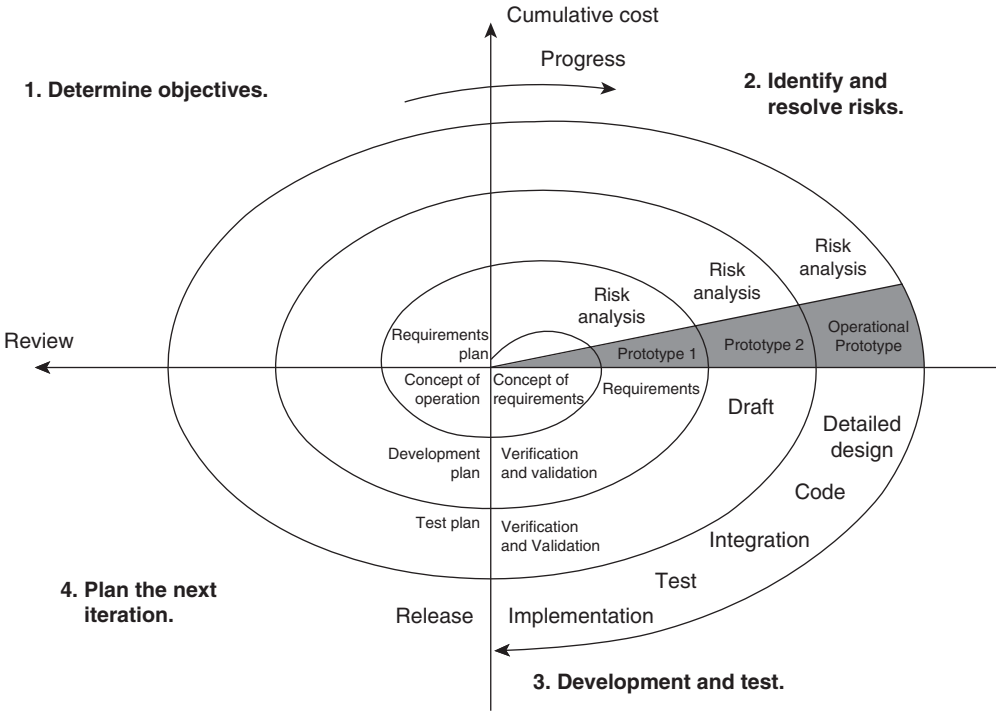

Figure 1.5 The spiral model (from Boehm 1988, reproduced with permission from IEEE © 1988 IEEE).

agile method is based. They are the following: (1) We value individuals and interactions over process and tools. (2) We value working software over comprehensive documentation. (3) We value customer collaboration over contract negotiation. (4) We value responding to change over following a plan.

In some software development circles, the availability of the manifesto and the agile model has given members of development teams license to throw away what they call heavyweight software development methods and to do their development work using whatever alternative methods suit their fancy. However, some advocates of the method have made a valiant attempt to bring its ideas to software development teams in a way that is both understandable and responsible. Probably the most commonly used software model is called Agile with SCRUM (Schwaber 2004). Figure 1.6 is one representation of how this model works.

Using this method, software development team members manage themselves and are responsible for figuring out how to turn a product backlog (list of capabilities desired by the customer) into increments of functionality in very short periods of time, typically of weeks or even days of duration. This is an incremental and iterative process. The loops in the figure are intended to show both the short iterations as well as the daily inspections of the work of the previous day that are done by members of the project team. The agile model can work for relatively small teams,

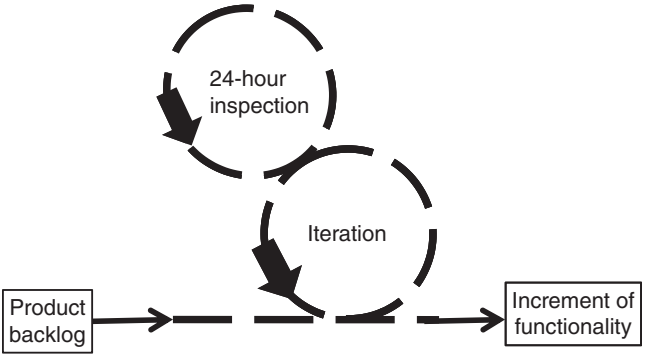

Figure 1.6 The agile model.

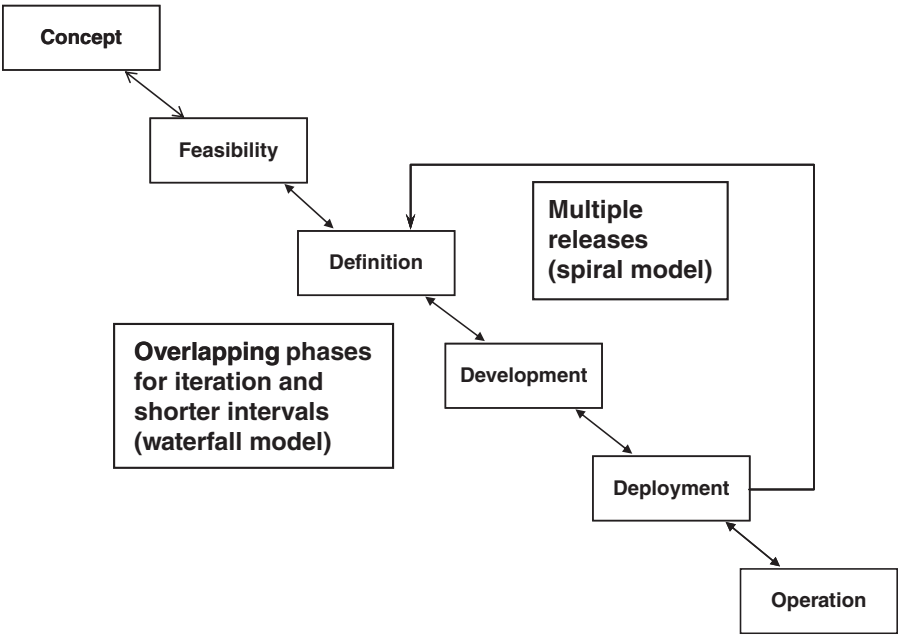

Figure 1.7 The better model.

and some of its advocates have attempted to scale it up for use on larger projects, but it has not been proven to be particularly useful for the kinds of projects that we will be discussing in this book. It is included here primarily for completeness.

To simplify our discussion, we will use a software development model that includes many of the characteristics of the waterfall model and the spiral model, and, perhaps, a bit of the agile model. It is shown in Fig. 1.7.

We will call it the "better model." It is better for several reasons. It is simpler than the spiral and agile models shown in Figs. 1.5 and 1.6 and contains

characteristics of both the waterfall and the spiral models. It provides a shorter interval to the first delivery than the waterfall model of Fig. 1.4. It allows for significant customer feedback and multiple iterations of the spiral and recognizes the possibility of both forward and backward iteration between phases and between repetitions of the spiral. It also allows for easier development because only some of the product's features are developed with each cycle. It also allows for both prototyping and risk analysis, which are vital parts of the spiral model, although those characteristics are not explicitly shown in Fig. 1.7. Using this model does introduce some risks. There is the risk of starting the succeeding phase before the preceding phase has been completed. There are risks that later iterations of the spiral might require significant changes to the architecture. Customers might not want a partially completed product. It requires early formation of a cross-functional team so that people who are involved in later phases are ready to begin shortly after the work starts. Therefore, project managers need to be sure they have the agreement of both their customer and their management when they use this model. All phases need to be managed, and frequent visible outputs are required to facilitate project monitoring and control when necessary.

So far, we have been discussing development models and processes at a relatively high level. None of these models tell an individual team member what he or she should do. In order to resolve that issue, we need to have a somewhat more detailed process to tell individual team members what to do and when to do it. The actual process that will be used on a specific project can be defined in several alternative places. It can be in the heads of team members, which is usually not a great alternative. It can be defined in a project plan, which is a better alternative, but if the organization takes on new projects, with new plans, regularly creating new processes for each project can consume significant effort and cost. The process might be defined in organization-wide documents, which might be good for overall business processes, but which tend to be too rigid for low-level product development work. It could be included in locally developed and managed process documents if they are available and could be used by everyone in the organization, or they can be contained in a quality manual, which is required if the organization intends to seek ISO (International Standards Organization) 9000 certification for its development operations or if it plans to seek a Capability Maturity Model Integration (CMMI®) assessment. While any of these alternatives can be useful, I have found that developing a quality manual and seeking ISO 9000 certification to be quite effective. Getting such a certification is expensive, but many organizations have found it to be valuable, particularly if the products being developed are intended for sale in global commercial markets. We will talk more about ISO certification and CMMI assessments in Chapter 9 when we discuss audits and reviews.

1.4 PROCESS AND PROJECT MANAGEMENT

If a software process is to be defined, someone needs to prepare and document that definition. It could be done by an individual, by an individual with some assistance from other members of the development team or organization, or by a

team charged with process management. Most observers would recommend that processes be managed by self-directed teams of people who are intimately familiar with the projects and the people in a product development organization. Self-directed teams can manage repeatable processes, and one of the purposes for developing a process specification is to drive software development in the direction of becoming a repeatable, or cyclic, process. The job of process management teams should be specification, management, and improvement of the process. They should not be charged with process enforcement. Their job is to define the process at an appropriate level that allows the process to be used repetitively by a group of people who may be working on one team or multiple development teams. It is sometimes desirable to provide the process management team with a management "coach" who can guide the team without acting as the supervisor of the team.

We have indicated earlier that software-based product development projects are not repeatable operations, so they cannot typically be self-managed. They can use the process that has been defined for the product they are working on or the organization they are working in, but someone must make the decision about whether the process can be used as is or whether it needs to be tailored for the current project. Software development projects frequently require prompt decision making and teams cannot usually make quick decisions. Therefore, while it is possible to have the process managed by a self-directed team, the project must be managed by an individual, sometimes with a staff, who has clear responsibility and authority for the project. That is the project manager.

So, while well-defined processes are necessary and helpful, they will not ensure successful projects. Both processes and projects must be planned, organized, monitored, and controlled.

1.5 SOME THINGS TO REMEMBER

The most important things we have discussed in this chapter are the following:

1. A product development *project* is a one-time, limited life endeavor that produces a unique output.
2. A *project manager* is responsible for producing the output on time, within budget, and meeting stakeholders' expectations.
3. A *project manager does* many things: plans, organizes, monitors, controls, leads, communicates, and motivates.
4. The *typical* project process (the "better" process) includes overlapped activities and multiple releases to achieve *speed*.
5. *Process management* is required to ensure good processes are in place and are working as planned.
6. A *development project* needs to be managed by an individual who has the ability to negotiate, scope, schedule, and budget.

CASE STUDY 1 *Implementing a Project Management System*

Purpose

1. Understand a situation faced by some newly trained development project managers.
2. Consider the criticality of several problems, including the lack of a project management process, competing for the manager's time and attention.
3. Practice planning the implementation of a project management system in a difficult environment.

Background

A few months after completing his study of this book, Stan Pasterchech was asked to take on his first job that involved software development. The move was not a promotion for Stan, but it gave him an opportunity to broaden his background, which had been in systems analysis and business planning. Stan was offered the opportunity to take on this new assignment because he had experience in two areas that the manager of the organization wanted to strengthen. They were in system requirements development and inventory management. Stan's previous assignment gave him the systems analysis background. His formal education had been in operations research and, in one of his early career assignments, he had become involved in several applications of inventory management theory to real-world problems.

When Stan arrived in his new assignment, he found himself managing an organization of 83 people. There were five supervisory groups in the organization, all of which reported to him. Three of the supervisory groups were responsible for software development. The fourth was responsible for customer support, and the fifth was a documentation and training group. Stan had been authorized to create and staff a sixth group that was to be responsible for requirements development. System test was the responsibility of a group in an organization managed by one of Stan's peers. His primary customers, who funded the work, were in other parts of his own corporation.

The people in Stan's new organization had spent several years developing and maintaining a large system based on a commercial database management application that tracked both physical inventory at a large number of geographically disbursed locations and the detailed financial records associated with the inventory investment. The system was in use at several locations throughout the world and was likely to require some enhancement and maintenance for several years into the future. However, the customers for the system were looking forward to a reduction in the needs for enhancements and a reduction in the cost and the need for staff on the project.

Fortunately for the staff, a new customer came along and asked if the organization could develop an inventory management system for a completely different application. The new application was not well defined, but there were several customer representatives who were looking forward to participating in the detailed definition of the new system.

The technology being used by the organization was very old. The existing application ran on IBM mainframes that were running the MVS operating system and the DB2 database management system. Most report-generating software and online transaction software were written in PL/I, and some of the online application softwares were even written in assembly language to assure adequate response time. Some of the older pieces of the inventory control software were written in FORTRAN. A few experiments had been carried out in which C++ had been used for report generation and online transactions. Formal detailed requirements were not usually written. A representative of the customers normally discussed their needs with one of the development group managers. One or more software developers would be

assigned to the feature and would proceed to mock up a transaction or a report and then review it with the customer representative before it was implemented. There was no formal development methodology being used. There was no project management process and there was no inspection program in place.

Stan knew that the methods that had been used for development for more than 20 years could not be used for the new application. He wanted to avoid the use of PL/I, FORTRAN, and assembly language at all costs. When he suggested that they might consider moving completely to a modern high-level development language and a distributed architecture with graphical user interfaces, two of the software development managers explained that it would be very difficult to do that given the underlying architecture of the existing system, and that they would be risking unpredictable transaction response times. While getting to know some of the nonmanagement members of the organization in one-on-one meetings, he found that there was a small cadre of some very good software developers who wanted to implement a code inspection program but that they had not gotten support from their peers or their management. Stan also knew that unless he put his newly found project management knowledge into action, he would quickly forget what he learned and that the new undertaking would be a disaster. Finally, within a few months, his team would have to develop an estimate of the funding that would be required during the following year for both the old and the new applications and would have to negotiate with the funding organizations on the deliverables that they could expect.

Activity

Try to put yourself in Stan's position. Think, from his point of view, of the problems that he is facing. Develop and describe in writing your evaluation of the relative seriousness of the lack of a project management methodology versus his other problems. Finally, develop a written recommendation to Stan about how he should address the lack of a project management process by briefly describing the most important steps that he should take to get one implemented. In preparing this recommendation, you should consider the following: What can practically be done under these circumstances? Who might he go to for help? How can he start moving the organization in the right direction? In particular, prepare a prioritized list of the issues Stan was facing and a second list indicating what steps he could take to implement a project management process.

Output

If you are using this text in a class, at your next class meeting, you should be prepared to discuss your conclusions and recommendations. During that discussion, you should be prepared to answer the following questions:

1. What is the relative importance of project management compared to other issues that are normally faced by organizational managers?
2. Is it practical to impose project management process requirements on an organization like the one described above?
3. How are the activities associated with implementing a project management system the same as, or different from, the activities associated with the general management of an organization?
4. In order to have a practical project management system in an organization like the one described here, what part of the implementation needs to be done personally by the manager of the organization and what part can be delegated to the group managers or to the people in the managers' groups?

REFERENCES

BECK K. et al. (2001) *Manifesto for Agile Software Development*, February 2001. http://www.agilemanifesto.org (accessed October 31, 2008).

BOEHM B. (1988) A spiral model of software development. *IEEE Computer* **21** (5), 61–72.

CRAWFORD S. G. and M. H. FALLAH (1985) Software development audits—A general procedure. *Proceedings of the 8th International Conference on Software Engineering*, pp. 137–141, IEEE, Piscataway, NJ, August.

ROYCE W. W. (1970) Managing the development of large software systems. In *Proceedings of IEEE WESCOM*, IEEE Computer Society Presss, Los Alamitos, CA, August, 1970, pp 1–9.

SCHWABER K. (2004) *Agile Project Management with SCRUM*. Microsoft Press, Redmond, WA.