CHAPTER 1

GETTING STARTED

Management is doing things right; leadership is doing the right things.

—Peter Drucker

GOALS AND SCOPE

This book is about managing software development efforts. By its very nature, management deals with people. It is a difficult job because you have to rely on others to perform the work. To accomplish the task at hand, you have to motivate your staff to do the right job right the first time. The job involves planning, organizing, staffing, directing, and controlling the delivery of complex software products. It requires leadership and teamwork. It requires discipline and organizational skills, as there is a lot to do and not a lot of time to do it. It requires being able to smell out the problems and address them before they have a chance to harm you. Many times, your destiny is shaped by others within enterprises who sometimes have trouble understanding what resources it takes to generate a quality product on budget and schedule. Other times, it involves working within enterprises where software workforces are underappreciated and viewed with distrust. Under such conditions, it is no wonder that such organizations have had a less than stellar success record.

By reading this book, I hope that you will learn what others have done to succeed in developing software even when the environment that they work in is not totally supportive. It is my contention that secrets of software success are known and can be communicated. Seasoned software managers know how to deliver quality products on time and within budget. Many have done it for years. The reason for software's seemingly high failure rate is that many neophyte managers do not put these secrets to work on the job. Sometimes, they cannot because they are just not in control of the factors that drive the development. For instance, if the schedule is driven by some hard deadline, such as a trade show, there is no way to postpone delivery. You either meet the schedule or have to wait for the next year when the promotional opportunity presents itself again. Other times, the people who are assigned to do the job just do not know what to do, when, and under what circumstances. They do not have the experience and have not developed the skills and knowledge needed to succeed at the job. Lastly, the task is difficult in its own right.

Software War Stories: Case Studies in Software Management, First Edition. Donald J. Reifer.

Building software to aggressive schedules under severe budget constraints is neither for the weak nor the weary. Vigilance, self-control, initiative, and attention to detail are required to ensure that the software product delivered satisfies all the parameters that govern whether or not the development is viewed as successful.

To address these issues, I have written this book. It contains 12 case studies that provide new and experienced managers with pointers on what to do when placed in commonplace situations. It also provides exercises to stimulate the reader to think about what they would do when facing obstacles that can cause the effort to go astray. I deliberately chose the cases to address the most common issues that have plagued software projects in the past. For example, managers may spend a disproportionate amount of time at the beginning of the effort trying to get the requirements right. When this is the case, they often play catch up throughout the project because there just is not enough time and budget left to finish the job right. Based on experience, they might have been better served by developing requirements iteratively using a build-a-little and test-a-little philosophy. This book also focuses attention on management indicators and metrics. By analyzing the data gathered and using these indicators to identify trouble signs early, my hope is that those reading this book will be able to take the corrective actions needed to get back on track in a timely and responsive manner. I also believe that metrics do not lie, and when they indicate that something is amiss, it most surely is because the numbers are truthful.

UNDERSTANDING THE ENTERPRISE

Software permeates just about every organization and enterprise engaged in commerce worldwide. For some firms, packaged software represents their bread and butter. They generate it for sale and it represents their business. For others, software represents an integral part of the information technology (IT) infrastructure that these industrial, government, and academic institutions use to conduct their business. According to Garner Research,* such enterprises will spend an estimated $3.8 trillion in 2012 on such IT services worldwide. These firms use software to help perform their administrative, customer support, human resources, management, marketing and sales, and manufacturing tasks. Most could not operate without it. Take an automobile manufacturing plant as an example. Their production and quality control systems are software driven, as are their inventory control, supplier management, distribution, and customer support systems. Software applications packages handle their payroll and accounting, human resources, travel, and benefits. Additional web-based systems facilitate access to these systems by their customers and employees. It should be noted that not a single aspect of their production process is unaffected by software, including the numerical control machinery, robotic controllers, and automated assembly units used for manufacturing. In addition, the resulting product, the automobile itself, often contains dozens of computers networked together to provide such functions as engine control, entertainment, driver controls and displays,

* See http://www.infoworld.com/d/the-industry-standard/gartner-lowers-2012-it-spending-forecast-37-percent-growth-183295.

parking assistance, active suspension, diagnostics, and collision avoidance. The world of the automobile and other products has become electronic based, computer controlled, net centric, and software driven.

Yet few automobile executive and plant managers come from a technical background. When you look at the corporate rosters of automobile companies worldwide, most senior managers in the business have marketing and sales, finance and accounting, and/or legal backgrounds. While most of these executives view software expenditures as a just and necessary expense, few understand these outlays well enough to make decisions that can have a profound effect on how either their IT or engineering organizations perform these tasks for the enterprise. One reason for this is that the enterprise's organizational architecture, which governs interactions and information flow, is often neither well designed nor understood. Instead, these executives rely on their chief information officer (CIO) to provide them with counsel and advice. While these executives may delegate the necessary authority to the CIO to run the IT shop, these same managers seem to constantly complain about the expenses and problems that occur when software is late, over budget, does not work, and/or generates incorrect results. They care because for premium automobiles, the cost of software and electronics can represent anywhere from 35% to 40% of the cost of the car.* Luckily for the software staff, the CIO often buffers the software teams from the pressures from above and provides them with room to operate in most industries, such as the automotive field, where electronics and software dominate.

You are probably thinking, "Why should I care whether senior management understands?" You need to be concerned because these people establish the policy infrastructure in which software groups must operate. This infrastructure by its very design creates guidelines for action for software and other groups within these firms. For example, you might have to go up the management chain to get signature after signature to hire a skilled software engineer without a college degree because company policies dictate that a degree is a prerequisite for employment. Yet this new hire may be one of the few people available who can program applications in some archaic language that you are still using in one of your legacy systems. A better approach would have been to delegate such hiring decisions (including the discretion for hiring waivers with justification) to lower tiers of management, especially when they can directly impact the performance of a critical task in the organization.

REVIEW OF SOFTWARE MANAGEMENT FUNDAMENTALS

I will not spend any more time on management basics. If you are interested in learning more about management theory, there are lots of good books and articles on the topic, some of which are referenced at the end of this chapter. For those needing more of a refresher, I have included Appendix B, For Your Bookshelf, where I provide some additional recommended readings.

* See http://news.discovery.com/autos/toyota-recall-software-code.html.

Software management is the act of motivating people to accomplish desired work-related goals and objectives using the resources available in the most efficient and effective manner. Management in such a context involves planning, organizing, staffing, leading or directing, and controlling activities within organizations comprised of groups of people charged with accomplishing a goal. Such goals can be product or service related. Product- or service-related goals can be accomplished via dedicated support teams or project organizations. Services can be rendered on-site or off-site, in person, by telephone, and via the Internet (blogs, websites, etc.). Resources available include people, equipment, facilities, licenses, and operating budgets, including funds set aside for capital expenditures and expenses. Subcontractors and suppliers can be used to augment internal resources so long as they are integrated into the management framework and viewed as part of the team.

Based on this definition, you are probably asking, "How does software management differ from management in general?" The answer is that it does not. It is not the principles that vary, it is the practice. Because software is intangible and hard to quantify, many have difficulty in applying proven concepts in practice as they go about their daily activities.

Most software organizations operate within the policy infrastructure established by their parent firm. They have to in order to survive and prosper. They follow its administrative practices and adhere to the processes and procedures that have been set up to guide accomplishment of just about every task from hiring and firing personnel to procurement and purchasing to fire safety. They take advantage of staff from their centralized accounting, legal, and human resources departments to handle things such as intellectual property concerns, bookkeeping and financial records, and privacy requirements. They guide their interaction with other engineering and support organizations within the firm as each group fights for attention, resources, and support from senior management. Even within firms that sell software products, the software group tends to be the minority, as the combined staff of the marketing, administrative, customer support, and other groups frequently exceeds their numbers by factors of 2 and 3 to 1. Success comes to those software managers who know how to play "the system" and get results.

Many of these firms have set up some form of engineering process group to develop, support, and institutionalize the disciplined use of common organizational software processes, such as requirements and supply chain management across their software and/or engineering organizations. Others have established independent quality assurance organizations whose job is to assess the quality of the software and other engineering products. Some of the quality groups perform independent testing as well. Others coordinate alpha and beta testing with third parties, when applicable. Still others are charged with performing configuration management and change-control duties as well. A variety of options exist as firms seek to combine functions to take advantage of the economies available through centralizing such activities.

Most software developments are managed using traditional project management techniques. Projects in this sense are temporary endeavors with defined start and end points and established objectives. Projects obey a life cycle (e.g., planning, requirements specification, architectural design, product development and test, delivery, distribution, maintenance and enhancement, and retirement), and their development may be ordered using one of many popular development paradigms

(waterfall, agile, incremental, iterative, etc.). Projects consume resources (people, equipment, facilities, etc.), and their success and failure can be measured in terms of whether or not they deliver an acceptable product (i.e., one that works and satisfies requirements) that provides value to the user on schedule and within budget.

Some firms perform all of their software development work as projects, while others use functional organizations or centers of excellence to get the job done in a more piecemeal manner. Yet others may use some hybrid approach that employs functional groups to perform projects. The functional organizations pass work between groups, each of which completes a different part of the technical job (applications development, platform readiness, test and evaluation, customer support, etc.). Many firms use a mix of both approaches using matrix management concepts.[1] Matrix management employs functional organizations to provide needed technical competencies and perform the work tasks. Functional managers are charged with acquiring and developing the skilled personnel needed to engineer the products and ensure their quality upon delivery. As part of the matrix concept, projects are superimposed on top of these functional groups to focus management attention on achievement of product-, service-, and/or customer-related goals (agreed-to budgets, delivery schedules, requirements, etc.). The matrix approach often causes discord within the organization as projects fight for the limited resources they need to meet their commitments. Friction often occurs between groups as alliances are made and broken.

The typical tasks that are performed by software development teams that use one of these three organizational approaches (i.e., project, center of excellence, and matrix management) include the following:

- Focus on achieving both near- and long-term business goals established by the enterprise.
- Attract, develop, and retain the talent necessary to accomplish these business goals.
- Create a work environment in which employees feel compelled to do their best and take pride in their work.
- Build and manage collaborative, cross-functional, and/or multidisciplinary teams. It should be noted that such teams can be either local or global and can include teammates, suppliers, subcontractors, vendors, offshore resources, and customers.
- Foster a culture that stimulates generation of innovative ideas, products, and services.
- Capture customer requirements and expectations and validate them with the presumed users.
- Ensure that projects are executed so that they conform to the enterprise's requirements and expectations.
- Develop prototypes to serve as proof-of-concept versions of new products and services.
- Develop versions of existing products for system integration and architect, as well as manage their implementation.

- Architect, design, and build comparable solutions innovating as needed to satisfy customer requirements and expectations.
- Document work products, both interim and final, in a manner that makes them useful to their intended audiences.
- Maintain the quality and integrity of the work products as they go through development and are distributed to the field for operational use.
- Direct teams so that they achieve product development milestones, timelines, and budgets.
- Mentor, train, and guide software engineers and managers in their assignments and evaluate their performance and work products.
- Participate in new business development supplying the ideas and talent needed to win contracts and attract new customers.
- Play a leadership role in improving how products and services are implemented by product and services teams.
- Serve as a technical resource for executives and managers with nonsoftware backgrounds to use to determine what it will take to get projects and products completed and delivered on time and within budget.
- Try to build better products more quickly by using new and more lean and agile developmental approaches.

As a departure to traditional techniques, many agile software organizations are embracing self-organizing teams. They are also utilizing product owners instead of project managers to facilitate development of software and are trying to minimize the need for excessive oversight. These unconventional techniques are sometimes hard to swallow for more conventional firms. However, results to date show promise especially when viewed in light of the software organization's ability to deliver.

THEORY VERSUS PRACTICE: WHICH IS IT?

Most good software managers that I know seem to understand the theory. Where most have had problems in the past has been trying to put these fundamentals into practice on-the-job. The primary difference between theory and practice is action and knowing when to take it. For example, practitioners have had to resist the temptation to take shortcuts in order to succeed with a software development effort. They also have to believe the indicators and take action in a timely manner when the warning signs indicate difficulties. Most software development efforts get into trouble a little bit at a time, not all at once. Acting upon the warning signs enables the practitioners to address these troubles with action before they can get out of hand. In some cases, no action may be an appropriate action. This is especially true when you believe the parties impacted by the issue will resolve it by themselves. In other cases, some sort of direct intervention may be needed because decisive action may be required to make corrections (placate the customer, replace nonperforming personnel, provide added resources, etc.).

For the most part, the problems that plague software projects tend to be persistent and do not fix themselves. Take as an example the issue of determining when a task is complete. Theory suggests that as you plan out the work, you break the job done into discrete tasks. For each task, you then define entry and exit conditions and the intermediate work products that will be generated. As the effort unfolds, these entry and exit conditions dictate whether or not tasks can be checked off as completed. As expected, these gating criteria should be as measurable as possible (all defects found fixed, all task products required delivered as promised, etc.). Else, all you would have to do is declare you are ready to start or complete a task. As an example, let us take a design task. Closure criteria for the task involve completing the design specifications, responding to design walkthrough comments, delivering, and placing the completed design document under configuration control in project repository. But these criteria are not enough. Experienced practitioners know modular designs lead to superior code (i.e., the theory) and, in response, will continuously refine their products and seemingly never reach closure. This update process tends to go on forever, and it appears that the designers will never finish their tasks. Experienced software managers know when they have reached that point where the gains made in improved design quality are worth the added costs. In other words, they know when it is time to tell the designers to "stop" and "move" on with the implementation. This knowledge reflects what it takes to put the theory into practice. To be truly effective, those managing the design effort need to comprehend when the design is "good enough." Theory is not sufficient. Practice requires a deep understanding of the work involved, the desired end game, and what it normally takes to finish it so that it is good enough. "Good enough" is conditional in the sense that it is a function of many time-sensitive governing factors that help you make an informed decision as to "go" or "no-go" with a proposed plan of action and milestones.

EMPHASIZING PRACTITIONER ROLES

When developing software, there are many roles for software practitioners and nonpractitioner alike. As you would expect, these practitioners are charged with performing the technical tasks associated with requirement elicitation, architecture, design, implementation, test, qualification, delivery, and distribution. Practitioners are asked to perform as members of teams and to provide peer review comments on intermediate work products, such as designs, code development, and unit tests. Depending on the software development paradigm selected for use, these practitioners will work with customers as coaches to translate their needs into use cases, stories, operational scenarios, and/or requirements specifications. Others may be tasked to architect the system and perhaps develop tests in parallel as they evolve the design and implement the system (i.e., test-first programming). Practitioners also may act as technical leaders especially if they are asked to mentor, train, and grow new supervisory talent. All of these tasks require specific skills, knowledge, and abilities to perform. The more proficient and experienced the practitioner, the better the results will be as competency wins out in just about all cases.

There are also many roles for nonpractitioners, both technical and support. For example, I have worked with many talented mathematicians whose job on software jobs has been to develop new and novel algorithms. I have also worked with technical specialists who are steeped in a subject such as network protocols, and others who come from the end-user community who understood intimately the operational aspects of the job. As an example, I worked with one insurance company where the IT staff had more people certified in insurance practices than the agents that they built applications for. Looking around, it is not uncommon within the larger software organizations to find business analysts hard at work handling accounting and finance issues and acting as the liaison with contracts, suppliers, and commercial off-the-shelf (COTS) packaged software vendors. Members of the staff frequently include measurement/metrics analysts and other specialists skilled in Six Sigma and lean manufacturing concepts. These professionals help the organization take and make sense of the software metrics and measurement data. Some software shops have business managers who handle the financial and accounting tasks, including record-keeping, within the shop. Others retain a separate staff to prepare and manage documentation. Yet others deploy support teams to do things such as keep the network operational and refreshed, address security and privacy, provide education and training, and perform configuration, database, and distribution management.

SETTING REALISTIC EXPECTATIONS

As the previous paragraphs emphasized, when managing a software development, we have a lot of work to do and a wide variety of resources at our disposal to do it with. Why, then, do we get in so much trouble with the job? Because software development involves so many different people and technologies and is approached in so many ways, this is not an easy question to answer. One bounding concept that I have used in the past[2] to answer this question is what I call the operating environment pentagon. The pentagon, which is shown in Figure 1.1, views the environment in which a software development operates from five separable viewpoints: organization, project, process, product, and people.

To understand each of our cases, I will review these five views so that readers can grasp the context in which software development takes place. Taken singly, each view paints a different picture of the case. Taken together, they provide a comprehensive framework that allows us to study any number of factors at a given time that can influence the development's outcome.

Organization

When managing a software development, it is unrealistic to assume that you will be exempt from the many policies and procedures that govern the organization of which you are part of. To be successful, those managing the work have to understand how to take advantage of the organizational infrastructure as they try to get their tasks done. For example, they might consult finance and accounting to determine the best ways to record, analyze, and report current software expenditures when they are

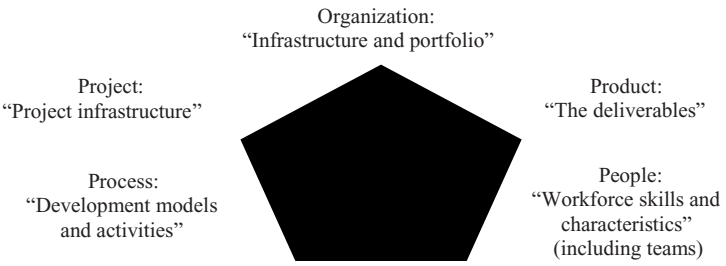

Figure 1.1 The operating environment pentagon.

trying to get insight into where projects are spending their money. You will be surprised how favorably people in other departments will respond when you treat them with respect, show them that you value their contributions, and ask them for their help. You also need to avoid using software jargon. Instead, talk to others in their language. If you are seeking finance and accounting help, talk business. If you are trying to communicate to senior management, discuss how software can contribute to achieving business goals in terms that they can understand (e.g., if seniors are from operations, talk in operational terms).

When I set the stage for the case studies that follow in later chapters, I will address the following aspects of the organizational infrastructure that most enterprises establish to manage their product portfolios. I will do this because each of the cases is impacted to some degree based on these factors, most of which are outside of the software manager's immediate control. In response, software managers have to learn to manipulate these factors so that they assist rather than impede them as their development efforts unfold.

- Senior management's focus on accomplishment of the enterprise's business goals
- Organizational alignment, with an emphasis on
  - Clear lines of communications and accountability
  - Authority commensurate with responsibilities
  - Ability to track problems to solutions to responsible parties
  - Use of interdisciplinary teams and fostering of teamwork
- Portfolio management and enterprise-wide policies and processes, especially those dealing with the following:
  - Capital equipment and software purchases
  - Contracting and subcontracting (including subcontract management)
  - Intellectual property protection
  - Release and approval for product launches
  - Supplier management
- Products and product line management practices, especially those emphasizing reuse
- Governance policies (especially those impacting software)

- Finance and accounting policies, including those involving reporting progress to seniors
- Human resources policies and practices (hiring, firing, promotion, etc.).

Project

Projects are established in many organizations to provide dedicated management for software product development and maintenance efforts. Project management practices are used in such organizations to plan, organize, staff, direct, and control accomplishment of these efforts. As reflected in many of the cases, expectations set by management for these projects are often unrealistic, especially when it comes to providing the resources needed to get the job done. For whatever reason, budgets and schedules established in these cases for achieving the project's goals are inadequate. When you search for the underlying cause, you often find that the software part of the project is poorly planned, understaffed, lacks resources, and is running out of control. Successful software managers adopt the following project management practices because they understand that this situation can exist and they want to reduce the chances of its likelihood.

- Plan the project by breaking the work done to tasks (i.e., each with their entry, exit, and delivery milestones defined) that can be achieved with available resources.
- Fight for the time, budget, staff, and other resources needed to succeed with the job.
- Organize the project, staff it with the best people available, and foster teamwork and collaboration with participating work units.
- Direct, motivate, and focus the team's energy on achieving the project's end goals.
- Lead the effort by example, showing subordinates how to successfully achieve their objectives within the organization's infrastructure framework.
- Measure progress and control variations by providing subordinates with meaningful feedback, mentoring, training, and guidance.
- Take pride in your work and deliver products that you can brag about.

Process

Processes are used by organizations at all levels of the enterprise to structure the activities pursued and provide policies, procedures, guidelines, and instructions for their disciplined conduct. At the enterprise level, such structure provides guidelines for action that are aligned with organizational business and growth objectives. For software, such structure is established to help facilitate sharing of best practices across organizations, both project and functional centers of excellence. Process frameworks, such as ISO/IEC 12207[3] and ISO/IEC 15288[4] and the Software Engineering Institute's Capability Maturity Model® Integration (CMMI),[5] provide a well-thought-out structure for pursuit of a responsive system/software process framework and improvement program, and so do the guidelines set forth by those advocating the use of agile and

test-first methods, such as Scrum.[6] Successful software managers take advantage of such process frameworks because they understand that they provide them with the following resources. However, they also recognize that the frameworks may have differences and embrace the unconventional when it makes sense to do so.

Effective systems and software engineering processes and best practices that work within the context of the overall process framework established by the enterprise.

- Examples of exemplary work products and training in their development
- Best-of-class project management practices that leverage existing reference works such as the Project Management Body of Knowledge (PMBOK®).[7]
- Responsive product management practices, such as configuration management, distribution management, and quality assurance
- Superior acquisition or supply chain management and licenses practices
- As we will see in our case studies, process issues can dominate, especially when they impede rather than expedite how development teams approach and accomplish their tasks.

Product

Software development activities are directed toward developing products and/or services that satisfy user requirements and, for some organizations, make a profit. Some of these products are stand-alone such as packaged software, while others may be embedded in another product such as an automobile, fighter aircraft, washing/drying machine, or cell phone. Some take advantage of libraries of reusable parts, while others focus on developing the features they need to differentiate their wares from their competitors. To build such products requires that they be architected with certain properties in mind (portability, reliability, maintainability, etc.) and that they can be developed and delivered within reasonable time frames. Successful software managers understand the intricacies of the product and put the processes and project management practices that they have available to work within the organization in which they operate using the following principles:

- Emphasis on nonfunctional as well as functional aspects of requirements (performance, reliability, etc.) as they are elicited
- Sound and modular service-oriented architecture that establishes the basis for product design
- Evolutionary paths through the architecture that facilitate planned future releases
- Focus on the reuse of qualified components and parts (algorithms, designs and design patterns, code libraries, etc.) from both vendors and past projects
- Traceability between user requirements and product features that promotes accountability
- Guidelines that foster superior product craftsmanship and construction (standards, etc.)

- Demonstrable quality for such properties as usability, reliability, and maintainability (i.e., through the use of quantitative measures instead of qualitative analogies).

The cases will show that when software developments get behind, management often eliminates either requirements or features because there just are not enough resources available to get all of the work done in the remaining time. However, limiting functionality beats taking shortcuts that can jeopardize product quality because these are the features that your users and customers really care about.

People

When you think about a development effort, you develop project, process, and product standards and guidelines to arm people with the principles, tools, guidelines, and standards they need to do their jobs in the most effective and efficient manner possible. However, when push comes to shove, it is the people you assign to the project that make the difference. If the people are motivated and try their best, the job will get done in spite of any obstacles that are placed in the way, including inadequate guidelines and standards. When they are down and out, project activity will slow down, and your prospect of meeting deadlines will be nil. Good software managers know that a few good people can make the difference between success and failure. They know which 20% of the workforce is responsible for 80% of the productivity. They also know that they can count on teams and teamwork to stimulate people to meet the challenges that they face in difficult situations. They also understand the impacts available personnel can make on a project especially in light of the following factors:

- Availability of a highly skilled, experienced, and talented workforce
- Experience with and understanding of the applications domain
- Interdisciplinary and integrated product team capabilities and possibilities
- Ability for advancement and opportunities to perform exciting technical work
- Capability to learn and use newer and more lean and agile methods (resume builders)
- Educational and training opportunities and ability to excel on the job
- Personal ownership of the work and a commitment to get the task done no matter what
- Clear sense of ethics, accountability, and personal responsibility
- Commitment to use organizational processes, practices, and the enterprise's system of management to do the job
- Clear and open communications channels across the enterprise.

To provide insight into the team's view of the current personnel landscape, each case tries to shed light on how the workforce views the project situation. This includes an assessment of the capabilities and experience of the staff, their morale, their skills, their view of management, and whether or not the team is functioning well together.

HOW DO YOU KNOW WHETHER YOU WILL BE SUCCESSFUL?

When asked, how do you know when a software project is going to be successful, I often ponder how to respond. My immediate answer is "it depends." While sounding evasive, no one answer to this question suffices as individual circumstances differ, and there are so many factors that can potentially influence the outcome. When looking a project over, I trust my blink[8] and know when something smells bad. I also know what factors software managers can and cannot do something about. As a consultant, I find that I am often brought in to influence factors outside of a software manager's immediate control (i.e., the noncontrollables) because I can get away with telling senior management that to correct performance issues changes are needed in the management infrastructure and environment where software operates.

In my opinion, there is no magic elixir for good software management. Sound management principles and concepts work for software projects just as they do for other technical disciplines. The trick is learning how to put them to work in an area where the mystique surrounding software sometimes makes understanding difficult for those uninitiated in the practice. As already noted, this book provides case studies that serve as examples of how to put these management principles to work in different operational settings. These cases emphasize how to use the available management tools and techniques to get the software job done in a timely manner while managing budgets and motivating workforces to perform what are often heroic tasks. They tell you to trust your blink, move ahead, and gain control of those factors that you can influence, both directly and indirectly, in order to get the job done.

RECOGNIZING BAD SMELLS AND TRUSTING YOUR BLINK

When you get involved in projects like I have as an independent observer, you can almost immediately figure out whether or not the project is going to succeed. It only takes a little digging to uncover the bad vibes, trouble signs, and bad smells.[9] I have found that your first impression or blink is almost always right. To trust your blink means that you must learn to trust your intuition. You must recognize that blink focuses for the most part on "how" and not "what and why." Seasoned-software managers can walk into a development shop and feel its heartbeat, pulse, and other vital signs. By paying attention to these signs, they can judge the level of activity being pursued, gauge the workforce's morale, and determine whether progress is being made. It is not hard when you have seen the good, the bad, and the ugly in the past. Good managers trust their blink. They use blink to know where the problems are and where focus is needed in order to put a project on the right track.

Management practitioners know not to equate the level of activity with that of progress. That is why they ask penetrating questions and investigate the indicators. They know when trouble is beckoning because they understand how to read the warning signs and when people are not giving them all of the facts. They look at measures of progress and trust the metrics/indicators to paint a more truthful

picture. I am often amazed about how new managers react when placed in a situation where their boss asks them the one question that unnerves them during a progress review. "How did he or she know to ask that?" the meeting attendees wonder. As I became that boss, I learned that you can often pinpoint the problem when you smell the fear or when your blink tells you that something is wonder. Blink and bad smells in my opinion are just cause to dig deeper especially when presenters are either suddenly nervous or seem to be moving through an important topic too quickly. It is time to dig deeper because something just is not right.

I will talk about bad smells and blink as part of each of the cases. Some of the common trouble signs that you probably can recognize include the following:

- People are milling around, arriving at work late or leaving early, and there seems to be no sense of urgency. This is a sign of low morale.
- Just the opposite of the above. Lot of frantic activity is occurring, but with little apparent progress. This warning signal suggests that people are not properly focused on the plan at hand either because it is no longer valid or due to a lack of direction.
- Software leaders are afraid to look you in the eye. Something smells wrong when this occurs. Either they are covering for someone or they failed to accomplish an assignment.
- Lots of finger-pointing occurs when you ask about why tasks are late. Lack of organization and/or poor teamwork may be the problem.
- Milestones are consistently occurring late even though the staff is working 40–60 hour weeks. This is a sign that there just may be too much work and too few staff assigned to get it done. A new, more realistic plan may be in order.
- Defect rates occurring during test are three times the norm. This metric indicates that quality may be suspect. The trick, as we will see here, is to search for the root cause.
- The rate of progress on the project is slower than expected. Perhaps, the staff needs more training? But sometimes adding people to a late project makes it later. This can occur when you have to take people off their tasks to train others. Sometimes, you can never catch up.

SEPARATING THE CONTROLLABLES FROM THE NONCONTROLLABLES

In many cases, software projects sometimes get in trouble for reasons beyond their control. For example, although the software workload is increasing, senior management is cutting staff by 10% because sales are suffering and they need to economize. They want to spread the pain fairly and the software group will have to cut like the rest of the departments. As another example, the tax laws change and all efforts seem to stop as the IT department reprioritizes all of its development work to ensure that the payroll can be met. As a third example, program management dictates that additional testing needs to be conducted to ensure that quality goals are satisfied as part

of its risk mitigation program. Of course, the tasking comes without additional budget, and other development work has to be cut back to pay for it. As a final example, the systems engineering group prematurely releases requirements specifications to the software group. They did this to meet a milestone, but the product they delivered was not up to par. The specifications that they published were incomplete, poorly written, and full of errors. The software group is in a fix. Because they need the requirements specification to do their design work, they need the problems fixed right away. However, they do not want to make too big a stink because they might damage their relationship with the systems group and they have to work with them closely in the future to get the product delivered on schedule.

There are many more examples that I could cite to show how software groups are not the masters of their own destinies. For software managers to succeed in practice, they have to recognize what factors they can control and what they cannot. Else, they will waste a lot of their time and energy needlessly trying to fix the unfixable. This does not mean that they should disregard things that are presently outside of their immediate control, especially those items that they can influence. For example, they might have been able to deal with the tax example above by addressing tax changes in their design using tables that can be easily updated, and help the systems engineering group develop better requirements specifications earlier by fostering the use of either interdisciplinary or integrated product team to generate the product specification.

I will discuss the controllables and the noncontrollables in each our case studies as well. My goal will be to try to help the reader understand what can be done when faced with situations that you may have the ability to influence, but not control directly. The reason for this is that most management texts do not consider these control issues. However, such issues are real and exposure to what works and what does not is useful. These situations occur most often in organizations that are easily influenced by external factors, such as those in changing and regulated marketplaces, that is, such as those involving new tax laws in our example. They also occur in organizations that task different groups with performing parts of the job, such as the systems engineering groups specifying requirements for others in our example, or in global development shops that parcel out the work overseas to keep their costs low.

SURVEYING THE TOOLS OF THE TRADE

As part of our case studies, I will refer to a lot of tools that software managers use to perform their jobs. For simplicity, I will organize these discussions around the line and project management functions that software managers perform. Line managers assume a wide range of responsibilities within functional and product-oriented organizations. Besides handling human resource functions (hiring, firing, appraising, promoting, and handling personnel issues), line managers help establish policies and processes, administer workplace practices, equip and tool the work environment, and establish a culture that enables achievement of enterprise goals, including those aimed at developing products and others directed toward realizing a profit. In contrast, project management plans, organizes, staffs, and controls available resources

from the line manager's functional organizations and directs them so that they can achieve customer-directed goals aimed at delivering acceptable products on time and within allocated budgets.

LINE MANAGEMENT TOOLS AND TECHNIQUES

Software line managers have a wide range of tools and techniques available to help them perform their jobs. The ones that I have used most frequently are summarized by function in Table 1.1. While most of these techniques are not unique, all can be tailored for use within the software domain. In most organizations, management of the functions listed in Table 1.1 is not viewed as unique across technical disciplines, and all organizations use the approaches that the enterprise selects to perform these standard functions. The reason for this is simple. This allows everyone to use the same standards and procedures adopted by the firm in a unified and systematic fashion. For example, time card systems employed across organizations for financial management can be the same. All you have to do is use a charge code or number to delineate the task that the group and/or individual uses to accumulate costs and assess budgetary performance. As another example, a single configuration

TABLE 1.1 Popular Software Line Management Tools and Techniques

Management function	Tool/technique
Portfolio analysis	• Kanban[10] and other lean techniques that use visualization and metrics to make selection and management of portfolio easy • Asset allocation and optimization tools • Sales targets and rate-of-progress visualization tools
Organizational planning and development	• Recommended organizational planning framework and life cycle models[11]
Financial resource management	• Software cost models[12,13] and spreadsheets (to set targets) • Financial management and time card accounting systems
Human resources management	• Career paths, job descriptions, and related education and training opportunities • Mentoring and on-the-job training opportunities • Staffing targets and rate-of-progress visualization tools
Process management	• Process management and improvement frameworks like those established by ISO and CMMI (appropriate for project and enterprise management as well) • Process management tools that support process modeling, compliance of processes to standards (e.g., CMMI), process tailoring and process execution for projects, visual process descriptions, and automatic generation of measurement reports and objective evidence for process appraisals • Agile frameworks that rely more on people than process and embrace self-organizing teams and other nontraditional concepts.
Product management	Product standards, development instructions, and guidelines
Decision support	Decision support system with variety of quantitative and informational tools used to help managers make timely decisions

management system can be used to manage versions using a common set of configuration identification, change control, configuration accounting, and verification practices to ensure the integrity of both hardware and software products. Such systems have an advantage especially when being used to track changes that have both a hardware and software component from initiation through implementation.

PROJECT MANAGEMENT TOOLS AND TECHNIQUES

Software project managers use a wide array of traditional project management tools and techniques to perform their jobs. Some of the more popular of these aides are summarized by function in Table 1.2. As already mentioned, software project managers use these tools and techniques to do their jobs just as do their counterparts in other technical disciplines. The trick to their effective use is to tailor these tools for

TABLE 1.2 Frequently Used Software Project Management Tools and Techniques

Management function	Tool/technique
Project planning	• Project charter establishing goals and triple constraint (cost, schedule, and scope of project) • WBS[14] that breaks work into tasks via architectural partitioning so that work packages can be allocated to teams and their members • Agile planning game used to prioritize work and develop time boxes
Project scheduling	• Gantt charts with measurable milestones and quantitative gating criteria (sometimes called quality gates) • PERT • Sprint planning such as that used by agile methods[15] • Critical chain and/or event chain management techniques[16] • Project management toolkit such as Microsoft Project
Project estimating/ budgeting	• Software cost models[12,16] and spreadsheets (to set targets) • Financial management and time card systems (to track actuals)
Project team building	• Interdisciplinary and/or integrated product teams[17] • Daily stand-up meetings such as those used by agile methods
Motivational tools	• Variety of financial (bonuses, etc.) and nonfinancial (recognition in project newsletter, time off, etc.) incentives and rewards
Progress assessment	• Rate of progress charts (target-actual comparisons) • Earned value reporting[18] • Velocity or burn rate like that used by agile methods
Product management	• Configuration management and version control system[19] • Design and coding guidelines • Collective code ownership
Quality management	• Metrics and measures[20] • Six Sigma techniques[21] • Peer reviews and walkthroughs
Risk management	Top 10 list with risk listed by impact on program along with recommended mitigation actions[22]
Decision support	Decision support system with variety of quantitative and informational tools used to help managers make timely decisions

WBS, work breakdown structure; PERT, program evaluation and review technique.

use for the particular needs of the organization (based on project size, type of product being built, stage of development, etc.) and provide realistic examples of what is expected relative to outputs (code snippets, documentation, etc.) and outcomes. Besides enabling software tasks to fit into a product framework, this enables software managers to share a common language and unifying framework for managing their work with their counterparts from other technical disciplines.

DIGGING DEEP TO FIND THE ROOT CAUSE

Many software managers view their work as unique and argue for use of separate systems for managing their work. However, I recommend resisting this urge and joining the mainstream. Because software is just one of the many organizations involved in product development in most enterprises, many organizational- and project-related problems that occur are caused by others whose work interfaces with and impacts software development. For example, software testing is often delayed because the hardware needed for testing is late. Because the software group needs the real hardware to demonstrate that the software works across the actual interfaces, they get blamed for missing deadlines as they wait for others to complete their tasks. Because such delays are a common occurrence, good software leaders should anticipate them, setup a contingency plan, and follow it to keep on schedule when problems occur. These managers need to use information from an integrated performance management system to track others' progress and put this Plan B into operation at the right time.

Software managers, like their counterparts in other organizations, often treat the symptoms of a problem instead of its root cause. The search for the truth is never easy and is always convoluted, especially when the leading indicators/metrics point in several directions at once. However, successful software managers know that the effort they spend digging for the truth is worth its weight in gold. They also understand the trouble signs and know when and when not to take action.

QUESTIONS TO BE ANSWERED

This book takes the view that management can be taught through example. To do this, I will set the stage for each case using the five views that are part of the operating environment pentagon illustrated in Figure 1.1. My aim is to show how factors, such as organizational constraints, process discipline, personnel motivation, and product standards, influence success in realistic cases within both industrial and government settings, both large and small. Issues will be raised and actions will be taken as each of the cases answers three to five important questions. Some of the many questions that we will address in the cases include the following:

- What functions does a software manager typically perform when managing line and project organizations (Chapter 1)?
- How do you increase your chances of being successful as a software manager (all chapters)?

- What are the tools and techniques of the software management trade, and how do they differ from the norm (Chapter 1)?
- How do you best organize to facilitate accomplishment of corporate goals in a large IT shop (Case Study 1—Chapter 2)?
- What is a reasonable cost and schedule for a development being done by multiple players (Case Study 2—Chapter 3)?
- When trying to get a software project back on track, what do you focus on (Case Study 3—Chapter 4)?
- What nonfinancial incentives would you use to reduce software staff turnover (Case Study 4—Chapter 5)?
- When managing requirements, what are the traps to watch out for and how do you identify them (Case Study 5—Chapter 6)?
- How do you ready an organization for transition to a new way of doing business (Case Study 6—Chapter 7)?
- When addressing software cost and schedule problems, how do you determine their root causes (Case Study 7—Chapter 8)?
- How much oversight is enough within a constrained but competitive contractual environment (Case Study 8—Chapter 9)?
- How do you mechanize the notion that software requirements are a learning exercise rather than a specification process (Case Study 9—Chapter 10)?
- How do you quickly change a software product and keep customers happy at the same time (Case Study 10—Chapter 11)?
- What education and training opportunities do you provide for new software hires (Case Study 11—Chapter 12)?
- How do you stimulate pursuit of software research in academia that has a near rather than far-term impact (Case Study 12—Chapter 13)?
- What are the most prized secrets that you can rely on to succeed in software management (Chapter 14)?

SUMMARY OF KEY POINTS

This chapter has introduced you to the book, its goals, and how it is organized. Using case studies, it tries to arm you with the core knowledge software managers need to succeed when pursuing difficult ventures. Using exercises, it will develop rudimentary skills for use on the job by having you answer relevant questions. This chapter has also introduced you to the framework that I will use to handle each of the case studies that will follow and what key questions I hope to answer. References and web pointers are provided because my goal is to illustrate through example how to address issues that you might face when trying to put proven management approaches, tools, and techniques into practice in your organization. The underlying message of the book is that there is no magic associated with good software management. In my opinion, hard work, competence, attention to detail, perseverance, and discipline when combined with sound management practice will result in success.

REFERENCES

1 Jay R. Galbraith, *Designing Matrix Organizations that Actually Work: How IBM, Procter & Gamble and Others Design for Success*, Jossey-Bass, 2008.

2 Donald J. Reifer, *Software Change Management: Case Studies and Practical Advice*, Microsoft Press, 2011.

3 International Organization for Standardization (ISO)/International Electro-technical Commission (IEC) 12207, *Software Life Cycle Processes*. Information Technology, 1995.

4 International Organization for Standardization (ISO)/International Electro-technical Commission (IEC) 15288, *Systems and Software Engineering, Life Cycle Processes*, 2008.

5 Mary Beth Chrissis, Mike Konrad, and Sandra Strum, *CMMI for Development: Guidelines for Process Integration and Product Improvement*, 3rd ed., Addison-Wesley, 2011. See CMMI for Development, Version 1.3 at http://www.sei.cmu.edu/reports/10tr033.pdf.

6 Mike Cohn, *Succeeding with Agile: Software Development using Scrum*, Addison-Wesley, 2009.

7 Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK®) Guide*, 5th ed., Newtown Square, PA: Project Management Institute.

8 Malcolm Gladwell, *Blink: The Power of Thinking Without Thinking*, Back Bay Books, 2007.

9 Hui Lui, Zhiyi Ma, Weizhong Shao, and Zhendong Niu, "Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort," *IEEE Transactions on Software Engineering*," January 2012, 220–235.

10 David J. Anderson and Donald G. Reinertsen, *Kanban: Successful Evolutionary Change for your Technology Business*, Blue Hole Press, 2010.

11 Chief Information Officer Council, *A Practical Guide to Federal Enterprise Architecture*, available at http://www.gao.gov/bestpractices/bpeaguide.pdf, 2001.

12 Steve McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, 2006.

13 Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece, *Software Cost Estimation with COCOMO-II*, Prentice-Hall, 2000.

14 Liliana Buchtik, *Secrets of Mastering the WBS in Real-World Projects: The Most Practical Approach to Work Breakdown Structures (WBS)*, Project Management Institute, 2010.

15 Chris Sims and Hillary Louise Johnson, *The Elements of Scrum*, Dymaxicon, 2011.

16 Lawrence P. Leach, *Critical Chain Project Management*, 2nd ed., Artech House, 2004.

17 Edward B. Magrab, Satyandra K. Gupta, F. Patrick McCluskey, and Peter Sandborn, *Integrated Product and Process Design and Development: The Product Realization Process*, 2nd ed., CRC Press, 2009.

18 Quentin W. Fleming and Joel M. Koppleman, *Earned Value Project Management*, 4th ed., Project Management Institute, 2010.

19 Robert Aiello and Leslie Sachs, *Configuration Management Best Practices: Practical Methods that Work in the Real World*, Addison Wesley, 2010.

20 Capers Jones, *Applied Software Measurement: Global Analysis of Productivity and Quality*, 3rd ed., McGraw-Hill, 2008.

21 Thomas Pyzdek and Paul Keller, *The Six Sigma Handbook*, 3rd ed., McGraw-Hill, 2009.

22 John McManus, *Risk Management in Software Development Projects*, Butterworth-Heinemann, 2003.

WEB POINTERS

Applicable web resources that amplify points made in this chapter can be found at the following:

- Amazon has lots of offerings that provide more details on the fundamentals. Go to http://www.amazon.com and search under books for "software management," "software project management," and "agile project management."

- The Project Management Institute is the one source that I count on for lots of papers, guides, and pointers to information about project management principles, tools, and techniques. You can access this information at http://www.pmi.org.
- The U.S. government provides lots of resources to help federal and sometimes state organizations striving to manage their software projects better. Some of the more useful of these sites include the following:
  - The White House has published several versions of Circular A-130 to provide overall policy and guidance for management of information resources in terms of consistent federal department and agency compliance with the requirements of various governance legislation, including the Budget and Accounting, Chief Information Officers, Clinger–Cohen (also known as the Information Technology Reform Management Act), Computer Security, e-Government, Federal Property and Administrative Services, Government Performance and Results, Paperwork Reduction, Privacy, and Procurement Acts, and their current revisions. These Acts provide insight into how large organizations move to improve their overall management of their IT resources at the enterprise level based on current critical reviews of their ongoing operations.
  - The U.S. General Services Administration (GSA) has issued many useful IT documents, such as its *Capital Planning and Investment Control* (*CPIC*) *Policy Guide* (see http://www.gsa.gov/graphics/staffoffices/capplan.doc).
  - The U.S. National Institute for Standards and Technology (NIST) provides numerous helpful guidelines for improved IT management, such as their *Guide for Developing Security Plans for Federal Information Systems* (see http://csrc.nist.gov/publications/nistpubs/800-18-Rev1/sp800-18-Rev1-final.pdf), *Risk Management Guide* (see http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf), and their *Contingency Planning Guide for Federal Information Systems* (see http://csrc.nist.gov/publications/nistpubs/800-34-rev1/sp800-34-rev1_errata-Nov11-2010.pdf).
- Many federal departments and agencies issue useful policies and guidelines for IT, including the following:
  - U.S. Department of Homeland Security's most recent *Management Directive on Information Technology Integration and Management* provides a good example (see http://www.dhs.gov/xlibrary/assets/foia/mgmt_directive_0007_1_information_technology_integration_and_management.pdf).
  - Department of Agriculture's *Information Technology System Development Life Cycle Guide* (see http://www.docstoc.com/docs/3455995/Information-Technology-System-Development-Life-Cycle-Guide).
  - Office of Personnel Management's *Guide to Telework in the Federal Government* (see http://www.telework.gov/guidance_and_legislation/telework_guide/telework_guide.pdf).
- The CIO Council addresses lots of subjects and publishes an assortment of guidelines and working papers for its members, many of which like those in

the *Federal CIO Roadmap* (see http://ourpublicservice.org/OPS/publications/viewcontentdetails.php?id=145) address current issues, such as use of social media in the work environment (see their *Guide for Secure Use of Social Media by Federal Departments and Agencies*, which can be found at https://cio.gov/wp-content/uploads/downloads/2012/09/Guidelines_for_Secure_Use_Social_Media_v01-0.pdf).

- A hidden source for excellent management materials for enterprises is located at university sites. These sites by design are open because they have to provide guidance to students in their use of IT. Some examples of the more useful of these sites are located at EDUCAUSE (see http://www.educause.edu/studentguide) and the University of Wisconsin—Madison (see http://www.doit.wisc.edu/facultystaff).
- Lots of excellent reports, papers, presentations, books, and other resources are available on facilitating software process improvement from the Software Engineering Institute's website, which is located at http://www.sei.cmu.edu/library. These resources by design revolve around use of the Institute's framework for process improvement called the CMMI.
- For papers and books that focus on the use of agile methods, you can visit http://agilealliance.org.
- There are several useful commercial sites that provide software project management resources. I particularly like Bright Hub's and the Air Force's Software Technology Support Center's for IT and embedded systems management (no endorsement offered for either of these sites as your personal tastes may differ from mine), which are located at http://www.brighthub.com/guides/project-management-software.aspx and http://www.stsc.hill.af.mil/resources/tech_docs.
- Process management tools can provide significant business benefits and facilitate process appraisals and tailoring. For example, Method Park's web-based process management system "Stages" enables process modeling, compliance of processes to standards (e.g., DO-178B, CMMI), process tailoring and process execution for projects, visual process descriptions, and automatic generation of measurement reports and objective evidence for process appraisals. See http://www.methodpark.com/en/product/stages.html, for details.