

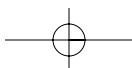
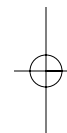
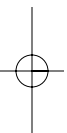
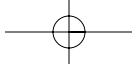
## PART 1

---

# ELEMENTS OF EFFECTIVE SOFTWARE MANAGEMENT

---

Effective software management requires a good manager, a good project management approach, and disciplined, directed management applied daily. Chapter 1 describes what it takes to be a good software manager. It can help you decide if you have or want the required qualities. Chapter 2 introduces the four ingredients of a successful software project, which are the main themes of this book: (1) the right balance of people, process, and product; (2) the diligent application of visibility techniques; (3) the correct use of configuration management; and (4) a reliance on standards. Chapter 3 looks at what happens when project managers violate these four principles. Chapter 4 illustrates how a project can succeed a day at a time—a concept that seems obvious but is often not realized in practice.



---

# *What Makes a Good Software Manager?*

---

Developing and maintaining software has become one of the most challenging and rewarding tasks a person can do. I've been privileged to attempt it in various settings for the last 15 years, and I've seen good software managers and bad ones. I've also *been* both at one time or another. This rather humbling experience has taught me what to value in a project manager. It has taught me that many managers approach software somewhat like the model in Figure 1.1. The project starts, everyone works hard, and software comes out some time later. If someone asks about progress, the project is always "almost finished." (If the project is headed by a more experienced project manager, it is always "93.7% finished.")

After much thought and observation, I believe this cloudy view of project management stems from a lack of three key perspectives. Of course, just having these perspectives does not guarantee a successful project, but it will go a long way toward making success possible.

## 1.1 PEOPLE PERSPECTIVE

Before I was a manager, I was a programmer, happily banging out solutions on the keyboard. I did well working alone. But software project management is not a loner's world. Loners who want to become software project managers must ask themselves hard questions: Do I like working closely with people? Do I want to understand what they mean when they talk to me? Do I basically *like* people? If the answer to these questions is a resounding "no," then you must ask another set of questions: Am I willing to change? Am I willing to



Figure 1.1 Typical software project management

study, take courses, read books, apply myself to people as diligently as I have ever applied myself to anything? If the answer is still “no,” stop reading right now, and give this book to a friend. Stay in coding and have a challenging, rewarding life.

If you are still reading, you must have answered “yes” to at least one set of questions. Now take a hard look at how you feel about yourself. Gerald Weinberg (Weinberg, 1994), an excellent source of advice on people, has pointed out that we need a good sense of self-worth, self-esteem, self-confidence, and self-examination to work closely with people. If we don’t have these, we will be hurt and hurt others often and deeply.

I have seen the truth of this on more than one occasion when a manager would attempt to change the way people do their jobs. Improvement, a goal for all, requires change and most people don’t like to change. If I as a manager walk into a gathering of programmers and announce that the group would be better off changing the way they do things, I can almost always expect unanimous rejection. If I do not have confidence and self-esteem, I will drag my bruised ego out of the room and never implement the needed improvement. If, however, I do like myself and believe in my knowledge and abilities, I am more likely to realize that the group’s negative reaction is a normal response to potential change, not a personal attack. A self-confident manager works through the fear of change person by person over time and implements the improvement.

I was able to move a group of programmers from coding cowboys to SEI CMM level 2. (Chapter 6 describes the Software Engineering Institute’s Capability Maturity Model in more detail.) It was neither easy nor quick, but it happened. Patience, knowledge, and self-esteem are required, and most people with a little determination can do the same.

Unfortunately, most of us are operating with a distinct handicap. We weren’t taught people or management skills. When we were in school, the instructor described the problem, and students individually coded a solution. Problems at work, however, require interaction with people, which can be fraught with difficulties. As Weinberg says, “the three causes of [software project] failure are people, people, and people.” (Weinberg, 1994) The lack of a people perspective has helped litter the software industry with failed projects. Rob Thomsett (Thomsett, 1995) states that “most projects fail because of people and project management concerns rather than technical issues.” I agree. I’ve seen new managers tell a group of talented programmers exactly how to

solve a problem. The resulting revolt is quick, decisive, and puzzling. As a new manager, I asked myself countless times, "How could these people not embrace my carefully thought-through technical solution and management plan?" Besides, wasn't I the manager (translation, "the boss")? Weren't they supposed to do what I said? And wasn't I appointed manager because I was the smartest person in the group? If they didn't agree with me soon, we would all fail.

As long as attitudes like these continue, people skills will be in demand. Over the years, through much trial and error, I have identified some core people skills and personal qualities anyone who attempts to manage will need. The list below is by no means inclusive.

- *Be flexible.* Let your people perform. The same people will react differently to a new project with a new product. Managers cannot manage each project just like the last one. The basic repeatable principles remain the same; only the particulars change (Constantine, 1995).
- *Have compassion.* You must also learn how to deal compassionately with difficult people. In Chapter 4 on managing a project day to day, I call for removing toxic people from projects. Some people habitually hurt others; some lie and steal. However, removing them does not always mean firing them. Some of the people who act this way can be helped. An organization should never condone theft, dishonesty, or hatred, but it should also strive to act with kindness, patience, and compassion in helping to correct such actions. Try to help reform problem employees and help good employees when they need help. Be careful, however, about attempting to perform miracles. Rescuing can become a destructive addiction. Do what you can, recognize your limitations, and call in professional help when needed.
- *Know when to lead and when to manage.* Lead people; manage the process and product. Leading means that others are following, so set an example. If you act honestly and courteously all the time, people are apt to do the same. Examples also extend to behavior with other groups. You can be sure that your people will be watching as you talk to upper management, so tell the truth.
- *Accept the role of meetings.* You will perpetually be in meetings (Maguire, 1994; Gause and Weinberg, 1989). They will be held in halls, offices, parking lots, and even in rooms. These meetings are not just bureaucracy; they are communication. If they are bureaucracy, it is your fault and you should empower yourself to correct it. Chapter 5 gives you some techniques for doing this.

Perhaps the best advice comes from the sports and recreation world (Hawkins, 1994). Choose the best people, keep the team small, minimize distractions, train them, meet together as a team regularly, know them, and

set an example. This is common sense, but all too uncommon in practice. As managers, we know what we should do and we have the means to do it. The last step is to get in the habit of doing it regularly.

## 12 BUSINESS PERSPECTIVE

The software industry is plagued with problems. Projects fall behind schedule, have cost overruns, or fail outright. And the failures are sometimes spectacular (who can forget the Denver International Airport baggage handling system or the Federal Aviation Administration's traffic control system?).

I'm convinced that a good part of the reason for these failures is a lack of business perspective. Most of us were introduced to software through programming. The problems were easy enough and we were smart enough to sit down and code the solutions without much thought. The problems we face now are more difficult, and we can't solve them alone. This requires different techniques and delving into what seems like mindless bureaucracy. Tasks like writing requirements, selecting a design alternative, and having others review our work are some techniques that can make us smart enough to solve these larger, more complex problems.

But these tasks encompass more than an immediate perspective. They require some idea of the big picture. They mean that we must look hard at management problems in the context of business requirements.

A well-known study by the Standish Group (Glass, 2002) paints an embarrassing picture. In the survey, 23% of the projects were canceled before completion (projects couldn't be saved); 49% of the projects were finished, but were over budget, late, and did not have the required functionality. Only 28% were on budget with all the desired functions. These numbers were improved from 1995 (Johnson, 1995), but still have much room for improvement.

I suspect that projects fail to deliver as frequently as they do because practitioners don't see the needs of the business enterprise. Building software at work is done for the benefit of the business. This means asking questions like "Who will use the software? Who wants it and who doesn't? What will the software do for the user and our business? Where will people use it? When do the users need (not want) it? Why do the users want it? Why does our business need it? Why are we developing it? What can we do differently that will bring greater benefit to our business and consequently to us as employees?"

I agree with Howard Rubin (Rubin, 1996), who said "a world-class (software) organization's primary distinguishing aspect is a common understanding of how its technical performance is transferred into *value for the enterprise*."

One of the best pieces of advice a software project manager can heed is to build only the software people want (Szmyt, 1994). This seems obvious, but

it is ignored all too frequently. Programmers build software; they don't look for reasons not to. When programmers learn a new technique, they quickly create a new solution (whether or not anyone else wants it) and invent a problem for it.

Unfortunately, if people don't want the software, they won't use it. All the money spent building it and trying to have people use it is wasted. Companies that waste money go out of business, and people lose paying jobs. Build interesting new software at home; at work, build software that people want and the business needs.

The key is to add value to the customer's endeavors. I close with a comment from Jerry Weinberg's SHAPE forum (Weinberg, 1997). "We all think we add value, but it's not value if the customer doesn't see it that way. If what you value is not what the customer values, it doesn't mean either one of you is "bad." It just means there is not a very good fit." One simple way to determine if I am adding value to the customers is to ask them. In my experience, they answer honestly. I place these comments here because they span the concepts of the people and business perspectives. Business, as most things, comes down to people. If the people aren't satisfied, the business isn't either.

### 13 PROCESS PERSPECTIVE

Software project managers must do the right things in software projects. This is known as using the right process or applying best practices. Table 1.1 takes a look at current level of practice and what is recommended for a successful project. The left column shows that the build phase or coding always occurs (McConnell, 1993). If there is no code, there is no software. The second column shows the tasks some projects may complete, although often unintentionally. Software managers should state the problem clearly (define requirements), decide among alternative solutions (design), bring together the elements of software into a product, and test the result systematically. The third column (when combined with the first two) shows what should be done. If you are a software manager who does not do each one of these on every project, this book is for you.

**Table 1.1** Always, maybe, and should do on a software project

Always do this	May do this (without knowing it)	Should do this
Build	Requirements Design Integration Test	Risk management Configuration management Inspections and reviews Quality assurance Project management Process improvement

### 13.1 Successful Process Techniques

A major influence in the process movement has been the Capability Maturity Model of the Software Engineering Institute (Humphrey, 1989; CMU/SEI, 1995). The CMM is a progression of levels, each of which contains key processes. An organization starts at the Initial level (see Chapter 6) and step by step adopts processes that lead to a greater ability to produce software. The SEI produced several more capability maturity models (e.g., one for software acquisition, one for people management, one for system engineering, etc.). They consolidated and replaced these with the Capability Maturity Model Integrated (CMMI). The CMMI contains proven engineering and management basics. Some people have maligned these CMMs as being too bureaucratic (government and defense oriented) and instead pushed best practices and, more recently, agile methods. If you look closely at the two, however, most best practices lists contain the same items as the CMMI.

I believe the CMMI works, and I've seen it work. I prefer not to debate whether it can be improved. A couple dozen experts in the United States are qualified to argue the finer points of the CMMI; the rest of us would be much better off if we simply followed it.

Humphrey created a CMM for the individual called the Personal Software Process (PSP) (Humphrey, 1995). The PSP (see Chapter 6) guides an individual programmer through a similar progression of processes. I worked through Humphrey's method and found that it also works at work. The exercise proved, much to my annoyance, that I was not smart enough to sit down and code off the top of my head. The PSP makes you write out a design, review it, write code, print the code, and review the code before compiling it the first time. My own metrics proved that with this mindful bureaucracy, I could produce more error-free software in less time.

These techniques (the CMMI, PSP, and others like them) apply basic engineering and management principles to software. Over the years, we software practitioners convinced ourselves that software was different. The basics did not apply to us; we needed to break the mold. We were wrong. Software is different in some ways, but it has more in common with other fields than we want to admit. We must use what others have proven works at work.

Some people feel they cannot do the right thing in their current job. I've heard it all: "Our management won't let us write requirements and hold a requirements inspection." "Our programmers would revolt if we asked them to let others inspect their code."

To these objections, I respond with two statements, which are easier said than done, but are not impossible to either say or do. You can probably fill in the blanks better than I can.

- Life is too short to \_\_\_\_\_ ("work for indecisive management," "chase out-of-control projects," "argue about proven practices," "fix errors late instead of early").

- We have never done \_\_\_\_\_ here (“code inspections,” “configuration management,” “planning,” “quality assurance,” “metrics”).

On my 35th birthday, I woke up and realized I was halfway to my biblical three score and ten years. Being on my second half, I decided that I would no longer run out-of-control projects. I would do the things I knew would work. If other people did not like that, I decided life is too short to put up with people who will not do the things proven to work at work.

### 13.2 Best Practices

The idea behind best practices (Glass, 1995) is to examine organizations that succeed consistently to discover what they do. If enough of these best-performing organizations practice similar tasks, these become a best practice for the industry. This is similar to, but not the same as, the emphasis on process. The best practices also include elements of people and product.

Below is a best practices list compiled from the best practices lists of many well-known authors: (Bach, 1995; Comaford, 1995; Jones, 1996a; Jones, 1996b; Johnson, 1995; Parnas, 1996; Racko, 1995; Reid, 1995; Sharp, 1995; Wirth, 1995). Note that the newest of these sources dates from 1996. Although I have read several dozen papers and texts since then that discuss best practices, none of them offer anything beyond this list. All the latest best practices come from one form or another of those listed here. I will repeat this list and discuss it again in Chapter 6.

- Risk management
- User manuals (as system specifications)
- Reviews, inspections, and walkthroughs
- Metrics (measurement data)
- Quality gates (binary quality decision gates)
- Milestones (requirements, specifications, design, code, test, manuals)
- Visibility of plans and progress
- Defect tracking
- Clear management accountability
- Technical performance related to value for the business
- Testing early and often
- Fewer, better people (project managers and technical people)
- Use of specialists
- Opposition of featuritis and creeping requirements
- Documentation for everything
- Design before implementing

- Planning (and use of planning tools)
- Cost estimation (using tools, realistic versus optimistic)
- Quality control
- Change management
- Reusable items
- Project tracking
- Users—understand them
- Buy-in and ownership of the project by all participants
- Executive sponsor
- Requirements

### 1.3.3 Management "Secrets"

There is no one big secret to developing software. If there were, more people would succeed at it. Fortunately, there are several little secrets, which can make a big difference to a project.

**Avoid having team members work in isolation.** Users, designers, programmers, testers, and managers, should be able to talk face to face in a matter of minutes. These days, that does not necessarily mean they are in the same building. It does mean, however, that you avoid saving up issues for the monthly flight to the coast. People who interact daily come to know one another, and the group of individuals more easily becomes a team. If possible, physically collocate everyone; if not possible, virtually collocate them. Inexpensive video conferencing provides a means for quick, easy discussion and is the closest thing to a physical meeting. Daily e-mail is a step down, but still permits inexpensive information transfer. With ftp sites and the like, people can work on the same virtual computer as if they were sitting down the hall. Think and use these technologies to provide fast, frequent contact at work.

**Stay with your project team.** Most of us started in front of the computer, programming solutions. We feel comfortable there, but the computer will not complete the project; the people will. Spend most of your time with the people working on your project. Most people drift to what is comfortable when under stress. For most of us, this means we go back to the computer terminal. Watch out for that tendency and spend your time with your people.

**Concentrate on tasks, not tools.** Learn how to perform a task first, and perform it until it becomes second nature. The tasks must become an established part of your culture. It takes time and money to learn how to use a tool. That learning cannot occur at the same time you learn how to perform a task. For example, learn how to plan projects using cards, sticky notes, and other common objects. Once you have mastered that task, learn how to use Microsoft Project or another schedule tracking tool. Don't try to learn both at the same time.

**Do your homework.** An informed manager is more likely to succeed. People and organizations worldwide are documenting their successes. Ideas are published in books, taught in seminars, and appear on Web sites. Our industry is often guilty of "ignorant originality" (Parnas, 1996), inventing techniques that have been proven to fail and creating techniques that someone else described years ago. Anything more than five years old is dismissed as irrelevant. Read books and magazines, attend seminars, and search the Internet. Keep a folder containing notes on everything you learn (Buzan, 1991).

#### 14 KEY THOUGHTS IN THIS CHAPTER

Successful project managers acknowledge a project from three main perspectives: people, business, and process.

The lack of a people perspective has helped litter the software industry with failed projects. You need a good sense of self-worth, self-esteem, self-confidence, and self-examination to work closely with people. If you don't have them, you will be hurt and hurt others often and deeply. Be flexible, have compassion, and know when to lead and when to manage. Choose the best people, keep the team small, minimize distractions, train them, meet together as a team regularly, know them, and set an example.

Project tasks like writing requirements, selecting a design alternative, and having others review our work mean looking hard at management problems in the context of business requirements. Build only the software people want and the business needs.

Use processes that are proven to work at work. The process should include stating the problem clearly (define requirements), deciding among alternative solutions (design), bringing together the elements of software into a product, and testing the result systematically. Follow models like the Capability Maturity Model and Personal Software Process, which apply sound engineering and management principles to software. Best practices also reflect proven processes.

There is no one secret to successful project management. It is a mix of common sense, people skills, planning, and awareness. Don't let people become isolated; be a presence; focus on tasks, not tools; and keep informed.

#### REFERENCES

- J. Bach, "The Challenge of Good Enough Software," *American Programmer*, October 1995, pp. 2-11.
- T. Buzan, *Use Both Sides of Your Brain*, 3rd ed., Penguin Books, New York, 1991.
- The Capability Maturity Model: Guidelines for Improving the Software Process*, Carnegie Mellon University/Software Engineering Institute, Addison-Wesley, Reading, MA, 1995.

- C. Comaford, "What's Wrong with Software Development?" *Software Development*, November 1995, pp. 27-28.
- L. Constantine, "Software by Teamwork: Working Smarter," *Software Development*, July 1995, pp. 36-45.
- D. Gause and G. M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, New York, 1989.
- R. Glass, "In Search of Self-Belief: The 'BOP' Phenomenon," *Computer*, January 1995, pp. 55-57.
- R. Glass, "Failure is Looking More Like Success These Days," *IEEE Software*, January February 2002, p. 102.
- "Coach Your Team to Success," *Software Development*, July 1994, pp. 36-43.
- W. Humphrey, *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989.
- W. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, MA, 1995.
- J. Johnson, "Creating Chaos," *American Programmer*, July 1995, pp. 3-7.
- C. Jones, "Our Worst Current Development Practices," *IEEE Software*, March 1996a, pp. 102-104.
- C. Jones, *Patterns of Software Systems Failure and Success*, International Thomson Computer Press, Boston, 1996b.
- S. Maguire, *Debugging the Development Process*, Microsoft Press, Redmond, WA, 1994.
- S. McConnell, *Code Complete*, Microsoft Press, Redmond, Washington, 1993.
- D. Parnas, "Why Software Jewels Are Rare," *Computer*, February 1996, pp. 57-60.
- R. Racko, "Hedging Your Bets: Part 2," *Software Development*, August 1995, pp. 73-76.
- W. Reid, "You Call That A System? Part 2," *American Programmer*, August 1995, pp. 25-33.
- H. Rubin, "World-Class Information Technology Organizations," *IT Metrics Strategies*, January 1996, pp. 1-2.
- O. Sharp, "How to Build Reliable Code," *Byte*, December 1995, pp. 50-51.
- P. Szmyt, "User Centered Development," *Software Development*, November 1994, pp. 49-59.
- R. Thomsett, "Project Pathology: A Study of Project Failures," *American Programmer*, July 1995, pp. 8-16.
- G. M. Weinberg, *Quality Software Management: Vol. 3, Congruent Action*, Dorset House, New York, 1994.
- G. M. Weinberg, The SHAPE Forum, [www.geraldmweinberg.com](http://www.geraldmweinberg.com), September 1997.
- N. Wirth, "A Plea for Lean Software," *Computer*, February 1995, pp. 64-68.