1 Every Business Is a Software Business

While technology can change quickly, getting your people to change takes a great deal longer. That is why the people-intensive job of developing software has had essentially the same problems for over 40 years. It is also why, unless you do something, the situation won't improve by itself. In fact, current trends suggest that your future products will use more software and be more complex than those of today. This means that more of your people will work on software and that their work will be harder to track and more difficult to manage. Unless you make some changes in the way your software work is done, your current problems will likely get much worse.

Regardless of the industry you are in, you almost certainly use software in just about every part of the business. For example, your software people develop and maintain the administrative systems for payroll, billing, receivables, sales tracking, and customer records. Software controls production, manages inventories, directs warehousing, and runs the distribution systems that operate your business. In service industries, your people build software to analyze, optimize, model, and support your clients. In product development, your engineers find that software is the most economical and reliable way to implement almost any sophisticated function. Software is now a critical element of computers, television sets, cell phones, and automobiles.

The quality of the software, its usability, and its timely development are critical to just about everything businesses now do. This means that, to manage your business, you must manage the software parts of that business effectively. Many managers and executives have struggled with the problem of managing software and have essentially given up. Because nothing they have tried seemed to work, they have concluded that they cannot manage software work. A common reaction is to outsource or subcontract the software work to somebody else. As many of the examples in this book show, that is often the worst possible solution. The performance of the subcontractors is generally no better, and it is often much worse.

Software work is entirely manageable, but only if you know how to manage it. The Software Engineering Institute (SEI) at Carnegie Mellon University was established by the U.S. Department of Defense in 1984 to work on the software problem. Its people have been addressing this problem ever since. They have learned why software work is so troublesome and what you can do about it. They have packaged their findings in a family of methods that are designed to help businesses like yours. This chapter summarizes the principles of this work, and the rest of the book describes what you can do to apply these principles to your organization.

THE PRINCIPLES OF SOFTWARE MANAGEMENT

To manage a software-intensive business, you must observe three management principles.

Principle Number One: Recognize That You Are in the Software Business

Whether or not you know it, you are almost certainly in the software business. If yours is like most businesses, software plays a pivotal role in most of your operations. For example, software schedule delays affect product delivery dates, and product delivery dates drive cost, revenue, and profit. Unless you can manage revenue and profit, you cannot manage a business. If you do not treat software as a critical part of your future, you cannot manage software, and then you might not even be able to manage your business.

Principal Number Two: Quality Must Be *the* Top Priority

In software work, quality problems overwhelm everything else. Quality is critical, and when quality is not managed, entire software projects are unmanageable. There are known ways to manage software quality, but they require proper training and disciplined engineering methods. The key need is for you to make a commitment to quality. You must make software quality *the* top priority.

Principal Number Three: Quality Software Is Developed by Disciplined and Motivated People

You cannot run an effective software operation without disciplined and motivated people. Software development is intellectual work, and undisciplined or unmotivated people cannot do timely or predictable intellectual work. Your people must be personally committed to their work, and they must care about the quality of the products they produce. Quality work is not done by accident; it is done only by skilled and motivated people.

These are the basic principles for managing software work. While they may seem obvious, they are not simple. As the examples in the rest of this chapter show, if you fail to follow any one of these principles, you cannot have a productive or effective software operation.

WHY EVERY BUSINESS IS A SOFTWARE BUSINESS

A senior vice president of Citibank once told me that "we are a software business masquerading as a bank." He explained that they could not run the bank without software. I see this situation in business after business: software is now a critical part of running many businesses. Some executives recognize it, but many others do not.

One example of the growth of software is in weapon systems. Figure 1.1 shows the growth of software in military aircraft from 1960 to 2000. With the F-4 in 1960, software supported only 8% of the functions the pilot performed. With the F-16 in 1982, this proportion had reached 45% and, with the newest F-22 in 2000, software controls 80% of everything the pilot does [1]. As

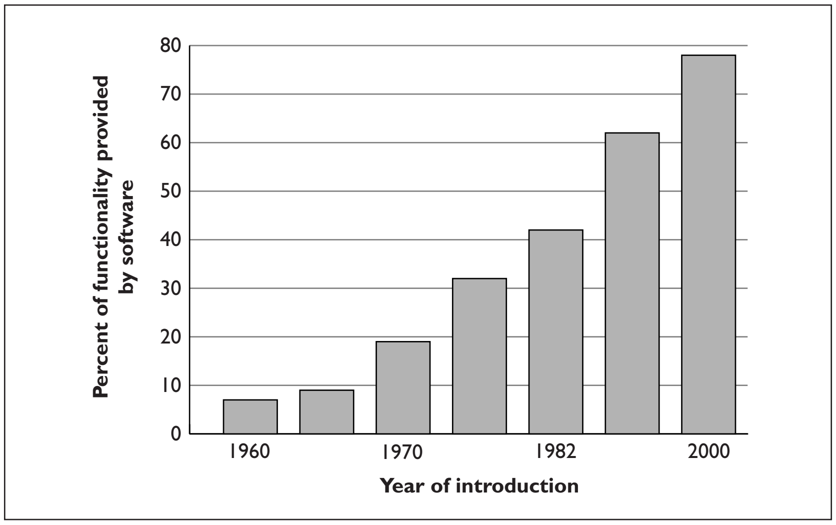

Figure 1.1 Software functionality in military aircraft

one general said, "The only thing you can do with an F-22 that does not require software is take a picture of it." That, of course, assumes that you are not using a digital camera.

The speed with which your people develop software can put you ahead of or behind your competitors. Software problems may have been frustrating in the past, but mismanaged software can now be fatal to a business. If your people do not produce quality software, testing times will be excessive, schedules will slip, and revenue will drop. You could soon be in serious trouble. The consequences of these problems are predictable.

1. If you extend schedules to make realistic customer commitments, you lose business.
2. If you make competitive commitments, your software is late and your customers are unhappy.
3. If you do this too often, you will be known as an unreliable supplier and have unhappy and disloyal customers.
4. If this condition continues for long, you will lose so many customers that you could well go out of business.

Even Fortune 500 businesses can fail, and often very quickly. In this fast-paced world, you rarely get a second chance—you must do the job right the first time. There is no second prize and no time to learn from mistakes. In fact, you rarely have time to catch up. Then, if you are not competitive, the business consequences will likely be severe.

WE'RE IN THE *HARDWARE* BUSINESS

IBM management did not understand the first principle of software management: they were in the software business. The consequences were severe. For many years, IBM thought of itself as a hardware company. It had started manufacturing and selling punched-card machines. It wasn't until the 1960s that IBM got into the computer business in a big way. Even then, IBM's senior management had grown up in the punched-card era and could not see the potential of software. They viewed software as an expense and thought they could make money only with hardware.

In 1976, the IBM CEO recognized that the personal computer (PC) business was important. He also found that engineering had no PC products under development or even planned. He pushed the development divisions for several years but finally lost his patience. Then he set up a special PC project that reported directly to him. He told this group to get a product on the market within a year.

The PC engineering group was formed quickly, and the product manager was told to break any IBM traditions that got in the way. IBM had recently disbanded its centralized programming group, and each hardware product team had software people of its own. Since the PC group was newly formed, it had no software resources and there was no centralized programming department from which to obtain help. The PC product manager had to get his programming support from a young entrepreneur named Bill Gates, who was just starting a company called Microsoft.

IBM announced the PC about a year later, and it was an enormous success. The company expanded production and extended marketing throughout the world. As PC demand grew, they added new models and enhanced performance and capacity. At the same time, Microsoft was enhancing the PC software.

When the PC was introduced, IBM was the third most profitable company in the world, and Microsoft was not even ranked among the top industrial organizations. Within ten years, IBM had the largest operating loss in history. Over the same period, Microsoft had grown spectacularly and had a market valuation more than twice that of IBM. It took another ten years, a complete change of management, and a downsizing of 200,000 employees for IBM to address the new software-intensive marketplace.

MAINTAIN CONTROL OF PRODUCT UNIQUENESS

IBM forgot a key business tenet: identify and protect that kernel that makes your products and services unique. Why did people want a PC? They did not want simply to have a pretty box. They wanted to do something, often something that no one had imagined. The power of computers is their flexibility, and the PC put this flexibility in the hands of the public. This flexibility was due entirely to the software.

When IBM realized that ceding control over the PC software to Microsoft was a mistake, it sank many millions into developing the OS/2 software system. OS/2 was an excellent system, but it was too late. Microsoft was too far ahead and the PC software business was moving too fast. IBM could not catch up.

IBM did not realize that the uniqueness of the PC product was in what it could do, and what it could do was almost entirely determined by the software. IBM got no software revenue from the PCs it sold, and the hardware soon became a low-profit commodity. The real tragedy is that IBM is no longer even a major force in the PC hardware business. That is now a commodity business, and whoever controls the PC software controls the PC business. Today, that is Microsoft, not IBM.

The Lesson of Principle Number One

With modern sophisticated products, the unique functions are increasingly embodied in the software, and a product's uniqueness is what makes it profitable. Therefore, if you do not recognize that you are in the software business and do not own the software in your products and services, you will lose control of your product's uniqueness. Then you will likely lose control of business revenue and profit.

QUALITY IS MORE IMPORTANT THAN SCHEDULE

Ashton Tate did not appreciate software management principle number two: that software quality must be the top priority. Their experiences illustrate the dangers of this mistake. The Ashton Tate business started in 1980 with the introduction of the Dbase database management program. This program was soon the market leader, and Ashton Tate was one of the software industry's big three. In 1987, Ashton Tate had sales of $215 million, only slightly behind Lotus at $283 million and Microsoft at $260 million. The Dbase product accounted for 65% of Ashton Tate's business.

When competitors started offering faster and easier-to-use database products, Ashton Tate developed an enhanced version called Dbase IV. In February 1988, it announced that Dbase IV would ship in May. In May, it announced a delay of two months, and in August it announced another two-month delay. In late September, Ashton Tate announced that the new Dbase product would ship by the end of October, when it was finally sent off to customers.

Unfortunately, Dbase IV had so many defects that, after it had been used for a few months, Ashton Tate had to withdraw it. In September 1989, when I met with Ashton Tate's CEO, the engineers were still testing and fixing Dbase IV. When the CEO asked for suggestions, I asked if he had any data on the product's quality problems. My suggestion was to look at these defect data and identify the most troublesome of the system's modules, or parts. Then they could focus on repairing the most defective modules. Since software defects typically cluster in a small percentage of the modules, Aston Tate's engineers could clean up most of the problems in about four months. However, because the CEO was committed to ship Dbase IV in two months, he did not follow my advice.

The Ashton Tate engineers continued testing Dbase IV, and they kept finding and fixing more defects. They did not ship in two months, and they were still testing and fixing problems a year later. By February 1991, Dbase IV was still in beta test and the CEO was replaced. Because of its quality problems, Ashton Tate reported a $5.6 million quarterly loss. It was soon bought by Borland. Ashton Tate, once the third largest company in the software industry, no longer exists.

The root cause of this problem was poor quality management. Ashton Tate had announced its Dbase IV product in February 1988 for delivery in May. The schedule kept slipping until finally, in October 1988, management said, "Ship it; we'll fix it later." This converted a schedule problem into a quality disaster. Instead of being late and inconveniencing their customers, they were now betting the company. Sadly, they lost the bet.

The Lesson of Principle Number Two

Ashton Tate's engineers and managers viewed software quality as a testing problem. As described later in this book, there are better ways to manage quality. However, these better ways all start with you. If you do not insist on quality from the very beginning, people will rush through their work, expecting somebody else to fix it later. Testing is enormously expensive, often taking half of the software development schedule. By using proven quality methods, these costs can be cut by ten or more times and schedules accelerated by many months or even years. Teradyne, in just two years, saved $5.3 million. That is why quality is important, and that is why you must make it the top priority. Everybody else can defer quality problems, but you must live with the consequences. Quality is an economic choice: pay a little now or a fortune later.

Another important lesson from the Ashton Tate experience concerns time-to-market. Most businesses quickly learn the importance of getting a product into the market at the right time. However, many make the assumption that time-to-market and quality are mutually exclusive. If you must get to market quickly, they reason, you will have to skimp on quality and ram products out fast as you can. As the Ashton Tate experience shows, this quick-and-dirty strategy is often slow and very expensive. To truly accelerate development work and optimize time-to-market, your people must do their jobs the right way the very first time. This reduces testing time and minimizes rework. Accomplishing this requires a corporate commitment to quality.

IN SOFTWARE, WHAT MUST HAPPEN OFTEN DOES NOT

The next example illustrates what can happen when managers do not follow software principle number three: that quality software is developed by disciplined and motivated people. I met Larry, the vice president of engineering, when his group was developing a large integrated production control system. Software delivery was committed for the following September, and it was only April, so Larry was optimistic that they would make it. He wanted my opinion.

When I asked about status, Larry explained that the coding was almost finished and that integration and system testing were under way. However, when I asked about product data, he did not know what I meant. I told him that, to understand where they stood, I had to know the size of the planned product, how much code had been written to date, how much had been released to integration and system test, and when it was released. He did not have this information.

Before giving Larry my opinion, I talked to the key managers and several of the engineers on the project. I also talked to the testers. Then I told Larry my conclusions. The system was just now starting testing and, based on the quality practices I had observed, the job was only about half done. Since the company had been developing this system for over a year, there was at least a year to go. While the product might be shipped in September, it would be in September of the following year.

Larry refused to believe me. They had to ship *this* September. While it was true that the business desperately needed an earlier shipment, they didn't even ship a year later. The company eventually ran out of money and was sold to a competitor. This is a classic case of poor software management.

The Lesson of Principle Number Three

Software managers and professionals who are not trained in quality methods will not believe that quality is important and will not follow the disciplined practices required to build quality products. Then testing will take at least half of the development schedule. Until you overcome this poor quality attitude, your people will not follow disciplined quality practices, and you cannot get quality software. Every aspect of software must start with quality, even the engineers' attitudes.

A QUALITY COMMITMENT

While few people talk about quality, and while software projects rarely have quality goals, most people would agree, "Of course, we must produce a quality product." Quality work is not an accident. People must believe that quality is important, and they must strive to produce quality products. Quality is like any other part of your business: if you don't measure it, you can't manage it, and if you don't manage it; it will not improve. Software quality can be measured, but until engineers measure and manage the quality of their work, the quality of their work will not improve.

No other modern technology rushes products through design and implementation and fixes them in test. Semiconductor engineers know that quality is critical. They cannot test and fix chips at the end of the line. When Toyota embraced the quality teachings of Dr. W. E. Deming, they demonstrated to Detroit that by managing quality, they could produce better cars and save money [2]. Finding and fixing problems in test is expensive for semiconductors, for automobiles, and for software.

Why don't software engineers focus on quality? It's not because they are lazy or unmotivated, but because of the way they have been trained and managed. Starting with their first programming courses, engineers learn that the most admired programmers produce code at lightning speed. Then they find and fix the defects in test. This fix-it-later attitude fosters poor practices throughout the software process. There are no quality standards, design standards, or even much in the way of implementation standards. To get quality work, you must change this culture.

If all engineers are poorly trained and if they all do undisciplined work, what can you do? Since you can't afford unpredictable schedules and poor-quality products, you must make changes in the way software is developed. The key questions are the following:

- Is there a better way to manage software?
- Is this better way economical?
- Is there a practical way to change organizations so they will consistently follow sound and high-quality engineering methods?

The answers are yes, yes, and yes. In the rest of this book I discuss ways to introduce effective software quality and management practices. The methods I describe—the Personal Software Process (PSP) and the Team Software Process (TSP)SM—are also attractive financially. As shown in Chapter 8 and Appendix F, an investment in these methods will yield a return of over 300%. Finally, a defined and available introduction program is available to help you and your people adopt these methods.

SUMMARY AND CONCLUSIONS

The following five principal points are made in this chapter:

1. Software is increasingly important to your business. If you can't effectively manage your software work, you will have trouble managing anything else.
2. The first principle of software management is to recognize that you are in the software business and to treat software management as critical.
3. The second principle of software management is that quality comes first, even before the schedule.
4. The third principle of software management is that, to consistently produce quality software, you must have disciplined and motivated professional teams.
5. There are known ways to manage software, but you must know them and you must use them. This book explains these methods, how to introduce them, and how to use them.

SM Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

94_HUMPHREY.ch01.qk 12/13/01 2:54 PM Page 14

Although sound software practices are not difficult, they are not obvious. The PSP and TSP provide an overall framework to guide engineers, managers, and executives through building and running an effective and productive software business.

REFERENCES

1. Jack Ferguson. “Crouching Dragon, Hidden Software: Software in DoD Weapon Systems.” *IEEE Software* (July/August 2001), pp. 105–107.

2. W. Edwards Deming, *Out of the Crisis*. Cambridge, MA: MIT Center for Advanced Engineering Study, 1982.