The changing world of software

Contents

When the marketplace changes, companies must adapt; those that don't, suffer serious consequences. The once invulnerable IBM, for example, had to cut its work force from 406,000 to 220,000 employees when they lost customers. One critical requirement for lasting business success is meeting customers' needs and doing it as well or better than the competition.

Since the early days of software development, our industry has been the outstanding example of poor performance. Even today, few expect software to be delivered on time and, when the products finally arrive, they often have lots of defects.

How long will customers to tolerate such performance? There are two parts to the answer. First, can anyone do better? And second, will the defects cause damage? Clearly, if no one does better work and if the customers do not suffer intolerable pain, the software marketplace could continue much as today. A closer look, however, shows that our world must change.

How does the software industry measure up?

It is not easy to produce quality software on time and on schedule. And, as tough as this work is today, it will not get any easier. That, however, is our problem. From the customers' perspective, our dates are no better than guesses. Even leading software organizations regularly adjust delivery dates several times before first product shipment. For other products or services, a deal is a deal. If the supplier raised the price, you would not be happy; you might even refuse to pay. Except for software, we expect firm dates and prices.

In other industries, suppliers commonly stand behind their products. We do not tolerate defective cars or appliances, and when we get them, we expect our money back or a replacement product. When we are seriously damaged, we demand compensation. In software, warranties are not warranties of quality; they are more like conditions for use. It is assumed, for example, that users will find defects. In fact, defects aren't even called defects; they are called bugs. The implication is that they are pesky annoyances that creep into products and must be tolerated.

Some developers argue that software development is an art form and that planning and cost management will damage creativity. Software work can be estimated, planned, and tracked, and when it is, these plans are generally met. From my experience, the least creative projects are unplanned, over budget, and behind schedule.

Some engineers argue that defect-free software can't be produced. While producing defect-free software is a non-trivial job, there is no evidence that it can't be done. In fact, in most large software systems, the defects are concentrated in only a few of the modules. In one large communications system, for example, 50% of the defects were in the modules for only 3% of the code. All the defects were in only 16% of the code. In three years of system maintenance, no defects were found in 84% of the code. If we can build products that are 84% defect free, shouldn't we try for 90%, or 95%, or even 99%? Who knows how far we can go until we try? Asserting that defect-free software is impossible merely breeds sloppy habits and puts off an orderly attack on the problem.

The new application environment

Until recently, software has been a peripheral business. That is, it has not run the production lines, controlled the money, flown the airplanes, or been the highest-cost or longest-lead-time product component. Until recently, the computer center shielded the general public from software problems.

This, however, has changed. Software now brakes automobiles, flies airplanes, controls railroads, and manages financial transactions. Daily, software moves over one trillion dollars in the world's financial markets. Defective software is increasingly expensive, and it can even be dangerous. In the threatening world of terrorists and hackers, software defects increase our exposure.

Software is now the critical component in many products. It is cheaper to use microprocessor chips and software than even simple hardware logic. My Army friends tell me that the trigger mechanisms in advanced side arms fire the guns through software. Artillery shells now have several thousand bytes of code and new military radios are microprocessors with antennae.

Once their defects are found, software-based systems are more reliable than hardware. In the nuclear power industry, for example, electro-mechanical instruments are being replaced by computerized controls. The old instruments are deteriorating and the new models are longer-lasting and potentially more reliable. If you lived next to a nuclear power plant, however, would you be happy with the traditional defect levels or a "buyer beware" warranty?

Why is this situation tolerated?

The software industry has a remarkably tolerant set of customers. Now, however, more users deal directly with the software. They don't like what they find. They don't understand our problems and they don't want to understand them. When a software defect damages or inconveniences them, they will be even less tolerant. If the damage is severe enough, you won't get calls on the hot line, your lawyers will hear from their lawyers.

Software is now or soon will be at the heart of most business systems. When the software is down, the production line stops. When the software fails, trains and airplanes stop moving. And when the software encounters a defect, the communications systems, banks, and financial markets will close.

An increasing number of industries are approaching this point. The trend can only accelerate. Unless we do something, software problems will soon become intolerable. Then, software managers will have only three choices. They can fix the software processes in their own organizations or they can look for outside suppliers. If neither of these alternatives works and if the software problems persist, they can get out of the software business. Unless they opt to improve their in-house software process, you could be out of a job.

This is not a hypothetical possibility

Highly visible software failures are increasingly common. American Airlines killed a big in-house project because of software problems. The A16 aircraft was canceled by the Navy, again partly because of software troubles. Ashton Tate had such severe quality problems with a new DBase version they recalled it. Before they could fix the problems, they went out of business.

There are now a growing number of alternatives. While no one consistently delivers defect-free software, some organizations measure software quality and work aggressively to improve it. A few organizations have learned to consistently deliver their products early. Hughes Electronics, for example, delivered the Peace Shield air defense system to Saudi Arabia six months ahead of schedule. Their $20,000,000 bonus fee is being divided among the 900 hardware and software development engineers. That averages $22,000 per worker. Don't tell me that process improvement doesn't pay!

What can one engineer do?

If your organization is not performing responsibly, and if your management is not working to fix the problems, your job could be exposed. This might be a prudent time to investigate software process improvement. While you probably can't fix the problems yourself, most successful process improvement efforts started with a single engineer. When you see needed improvements and when you aggressively pursue them, you can have an impact. Think about the problems in your organization and start an improvement ground swell. You could be the catalyst for major change.

An invitation to readers

In future columns, I will talk about process improvement, the process bureaucracy, and the impact of processes on engineers. For this column to be successful, however, I need your feedback. This will tell me your interests and help me to address the issues that are most important to you. Drop me a note, but please recognize that process is an enormous subject, and that each column can only address a small topic. When some facet is not mentioned, it does not mean it is unimportant. I probably didn't have space or didn't think of it.

Also, please keep your e-mail notes brief and only send mail on your key concerns. If they somehow relate to the software process, let me know. Do not expect an answer but be assured that I will read your mail and consider it when I plan future columns.

Thanks for your attention and stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu