Tsuneo Yamaura

Soapb

Why Johnny Can't Test

EDITOR: Tomoo Matsubara • Matsubara Consulting • tmatsu@xa.so-net.or.jp

This is an important observation by a Japanese software engineer from a culture of rigorous practices who works with American software engineers from a markedly different culture. I believe his pessimistic and fatalistic view is just a starting point for debate that could lead to optimistic and constructive propositions. I welcome your comments.

—Tomoo Matsubara

The question "Can US programmers be great testers?" is one side of a coin whose other side is, "Can Japanese programmers be creative software designers?" I compare American and Japanese programmers because I believe they represent the extremes in the range of programmers' behavior with regard to the "value of software." We could probably plot other countries' programmers in between.

Based on my 15-year experience of working with American software engineers, my pessimistic, fatalistic answer to the question I've posed is this: American programmers will never excel at developing high-quality software. My answer to the counterpart question is that Japanese software engineers will never be creative programmers. Why? Because a creative programmer cannot develop quality software, and a person who can make flawless software cannot innovate.

American and Japanese programmers have exemplified this for several decades. The reason is quite simple: they were brought up differently. Such cultural differences affect programmers' basic activities—they cannot be trained to be good testers or creators unless their social and engineering environment supports these traits. State-of-the-art tools and methodologies will not help Americans and Japanese compensate for their weaknesses—to improve, they must change corporate environments and long-standing traditions. If such change is even possible, it would take an enormously long time.

People—whether organized into societies, project teams, or companies—both exist within and create culture. This means that if people (and organizations) belong, for example, to a culture that does not foster sound quality assurance practices they cannot create quality software—even if they bring in state-of-the-art test strategies, methodologies, or tools and organize a quality assurance team. Their culture does not permit it, and therefore change cannot come from outside but must come from within.

0740-7459/98/$10.00 © 1998 IEEE

Industrial Environment in the US and Japan

American programmers aren't great testers, but they are exceptionally creative: The key concepts that determine the entire software industry's future have always come from the US. Although Japan's software industry, including quality assurance, has matured, it almost always builds on innovations from the US. The basic difference between the US and Japanese software industries is that the Japanese seek "software industrialization" while US corporations reward "novelty." Such idiosyncratic characteristics exist in the business environment as canonical standards to which employees must conform. The corporate culture (sometimes documented as company regulations, but often simply indicated by rewards for "appropriate" behavior) becomes deeply etched in the programmer's mind, and following generations inherit it without doubting its validity or suitability.

If people (and companies) work in a culture that does not foster quality assurance, they can't create quality software.

Definition of success

The Japanese value project team success far more than individual accomplishment. Thus programmers honestly report the bugs they produced during all phases of software development. Such defect data serves to create and validate software quality assurance theories, which are intrinsically statistics-based and empirical.

On the other hand, US culture values personal achievement over team success. Since reporting "I caused this many bugs" automatically implies "I am this stupid," a programmer will likely fudge his personal bug statistics when submitting them to a manager. Needless to say, this makes the application of QA theory difficult, and virtually guarantees that US companies will produce flawed software.

Business strategy

The software company's strategy of quality versus marketing also varies between the US and Japan. In the US, companies try to dominate the software market by shipping a new product early. Although a product's quality must satisfy certain minimum criteria, corporate culture emphasizes being the first out the door no matter what.

In Japan, companies demand high-quality products even at the expense of cost overruns, schedule delays, and lost market share. They believe that this approach, while slower, is the surest way in the long run to save money and recover from delays.

Cultural background

The significant differences between US and Japanese cultures uphold my belief. US culture encourages people to emphasize their uniqueness, their differences from each other, and to develop a strong "I am what I am" mentality. This makes novelty and challenge two core cultural values that, in turn, encourage software engineers to design ambitious products. Most of these may not meet market demands, but a few pieces of exceptional software will become the backbone of the next generation of technology. A generous social atmosphere fuels the budget for such challenges, and investors can usually expect a handsome return for their high risk.

Japanese culture emphasizes "safe play" and gives the risk-free plan the highest priority even though it brings only a minuscule return. Japanese companies feel safest observing what US software companies do first, then importing their most promising ideas. Great ideas such as the Internet and Java cannot emerge in such an environment.

Human Characteristics of the US and Japan

Shepard of the University of Illinois at Urbana Champaign has collected decades' worth of data from many countries to analyze human characteristics. He classified people into four types, as shown in Figure 1.

- A P-type person likes forward-flowing work and can think flexibly. He or she can solve problems intuitively but often skips stepwise logical confirmation.
- An A-type person does not easily come up with novel ideas, but likes to run things step by step.
- An F-type person seeks togetherness and appreciates family, friends, collaboration, and harmony.
- A C-type person is a typical eagle-eyed manager who may puzzle you with a barrage of tough questions and demand quick responses.

While this classification profiles individual characteristics, we can apply it to organizations and

countries, too. Typically, Japanese programmers and organizations embody type A characteristics, while Americans as a whole fall under type P. Shepard's classification gives us another way to look at why US programmers can invent new technologies but are not good at quality assurance, whereas the Japanese eagerly seek software quality but don't often come up with unique ideas.

Type dynamics and interactions

Shepard's work uncovered important rules and dynamics among the four types. For example, diagonally situated types (that is, P and A, and F and C) do not get along well. We can easily see this in one area of Japanese–US interaction: American programmers find the Japanese QA approach[2] cumbersome and time-consuming even though it guarantees that only 0.02 percent of all bugs a project generates ultimately emerge at the user's site.

According to Shepard, a person's characteristics can span two or more types. He called someone with both A and P characteristics the Genius type because that person can come up with a great idea and be persistent enough to prove it logically. Einstein epitomizes this A–P type. A type C–F person is a born manager who knows how to effectively use carrots and sticks.

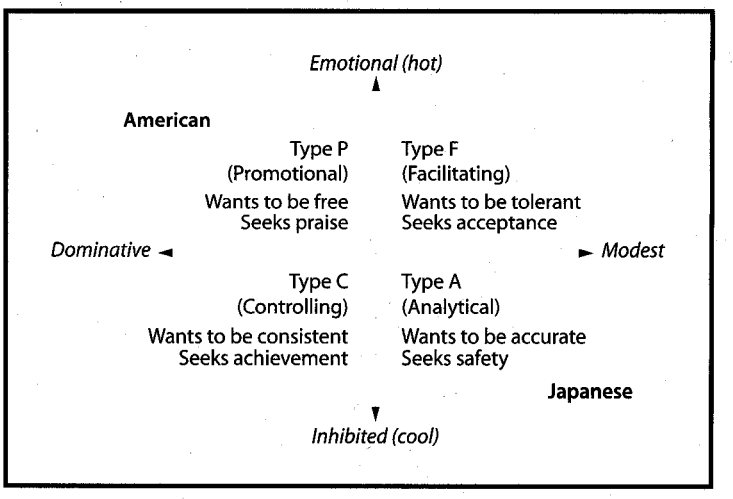

FIGURE 1. Shepard's classification of human characteristics.

Fixing the Imbalance

My analysis of cultural differences, while quite pessimistic for software engineering, also suggests two possible remedies. The first, as Raymond Paul suggested to me at COMPSAC '97 in Washington, D.C., is to pick A-type people to form a test team in the US, and P types in Japan to organize a creative project. This approach is time-consuming, risky, and expensive, however, because few candidates will be found that can meet these qualifications and they will likely be in great demand.

The other remedy entails the collaboration of US and Japanese corporations in an Einstein-type project. For example, an American company may develop a core idea and build a prototype that embodies just the basic functions and does not require serious testing, documentation, error handling, or GUI work. Using the prototype as an executable specification, a Japanese company could then industrialize the software.

I do not deny the usefulness of test tools, methodologies, and theories. Rather, I insist they will never work unless programmers and projects focus on quality. It is hopeless to believe that bringing in fancy tools will solve the poor quality problem instantly—a broccoli hater can never be a great broccoli cook, even with the best cookware. ❖

References

1. T. Yamaura, "Standing Naked in the Snow," *American Programmer*, Vol. 5, No. 1, Jan. 1992.
2. A. Onoma and T. Yamaura, "Practical Steps toward Quality Development," *IEEE Software*, Sept. 1995, pp. 68-77.

Tsuneo Yamaura is a senior engineer at Hitachi Software Engineering. His research interests include testing methodologies, software metrics, development paradigms, software modeling, and CASE.

Yamaura received a BS in electrical engineering from Himeji Institute of Technology and was a visiting scholar at the University of California, Berkeley from 1984 to 1986. He is a member of the IEEE Computer Society and ACM.

Address questions about this article to Yamaura at yamaur_t@soft.hitachi.co.jp.