They Don't Care About Quality

Kathy Iberle

Acknowledgements

My thanks to the participants of the Software Test Managers Roundtable (STMR), whose stories from a wide range of software development environments contributed substantially to this article. Facilities and other support for STMR are provided by Software Quality Engineering, which hosts these meetings in conjunction with the STAR conferences.

Special thanks to Esther Derby for her feedback on this article, and to James Bach for suggesting the "fear" angle.

Introduction

"These people just don't care about quality!" How many times have you muttered this to yourself while negotiating the rocky shoals of an impending release? I've certainly said this plenty of times, but never so frequently as in the year I moved from medical products to consumer printers. This is the story of how I came to recognize that quality takes different forms, and that sincerely caring about quality might not look the way I expect.

A Journey to a Different Land

Medical products is a field with a reputation for being quality-conscious, for obvious reasons. During my tenure in cardiology products, we used many of the classic practices described in software engineering textbooks: detailed written specifications, extensive use of inspections and reviews throughout the lifecycle, exhaustive requirements-based testing. Release criteria were established at the beginning of a project, and a product simply did not ship until its release criteria were met. We could (and did) miss release dates by weeks or even months, chasing down the last few serious problems. This was far from painless – we worked overtime to make up what we could, and there were heated debates over whether a particular defect should be classified as serious or medium – but in the end, correctness would win out over schedule.

After eight years in medical products, I moved into the inkjet printer business, which served small businesses and the average consumer. The practices in this organization were very different – the specifications were much shorter, release criteria were much less formal, and hitting the release date was all-important. Since I was working in testing, I particularly noticed the differences in testing practices. They didn't devote the bulk of testing effort to testing against the specifications, often ran the same tests repeatedly, and didn't attempt to exhaustively exercise every possible combination of input. There was much less test documentation - in fact, sometimes I would see testers testing with no written test procedures at all.

This created massive culture shock. At first, I walked around shaking my head and muttering, "These people just don't care about quality!" After a while, I began to realize that my definition of quality wasn't quite so infallible as I had thought. It was time to re-examine my beliefs about quality.

What is Quality?

In his book *Quality Software Management: Systems Thinking*, Jerry Weinberg defines quality as "value to some person" [Weinberg92]. When quality is defined this way, it is far from immutable – in fact, it's quite subjective. So, who expects to get value and what value do they expect? In the broadest sense, the customers expect value in exchange for their money. The company's board of directors and stockholders expect profit. The value expected by the customer takes different forms in different products. Profit can be achieved through different business strategies, which also place value on different aspects of the product. How can we find out what is most valued by the customer and by the seller for this particular business?

The most obvious approach is to ask the developers and marketing staff what they think the customer values, and what the business values. The answers are prone to some interesting biases. In both organizations, people said that the customer and the business valued "quality". In both organizations, people said that hitting the release date was very important to the business.[1] The language used was almost identical, yet the actions were quite different. Do the two groups value different things, or is one simply not walking its talk?

Asking about fears instead of values uncovers a more candid story. Workers are often told what their values should be, and reminded with posters and coffee mugs and motivational meetings, until they parrot the company line without thinking too hard about who values what. However, you don't see posters and coffee mugs trumpeting "The Top Ten Fears of Our Organization". People do think about their fears often, speak about their fears in meetings, explicitly attribute decisions to specific fears, and put up relevant cartoons on the walls. In contrast, people often assume that what they value is universally valued, and therefore too obvious to mention.

Observing what makes people worry, and more so, what justifies a major change in plans will often point directly to a business value, and often to a customer value as well.

Anxiety, Fear, and Terror

When I worked on defibrillators and cardiographs, we didn't consider a missed release date to be the worst possible thing that could happen. The possibilities that froze our blood were those that could lead to harm to a patient or operator – electrical shock, misdiagnosis, failure to operate when urgently needed. Any suggestion that a defect fell into one of these classes was automatic grounds to delay release – no argument, no discussion. Time-consuming and expensive efforts to find and conclusively eliminate the cause of the problem were routinely approved. The value to the business of avoiding legal liability or being shut down by the FDA certainly figured into this, but the driving consideration that I observed was simply the acknowledged responsibility for human lives.

[1] In fact, some of the marketing people in medical products argued that hitting the release date was part of "quality", since the customer couldn't use the product if it wasn't shipped. This made the developers deeply unhappy since it appeared to set on-time shipment ahead of their traditional definition of quality, which was more customer-centered. If you define quality as being both value to the business, and value to the customer, you can have your cake and eat it too.

Release dates, on the other hand, could be tweaked in several ways. The customer orders products rather than going to the store, so products could be listed as being on back-order for the first few weeks. Since the products were manufactured in a single location on a single shift, it was possible to double production for a couple of weeks simply by offering overtime, and fill those backlogged orders quickly. Often, the lag between a hospital buyer seeing a new product at a trade show and that buyer getting a purchase approved covered a week or two slip all by itself.

In consumer products, life was different. Fear of misdiagnosis doesn't apply at all, and the potential for injury is very small even under the worst imaginable conditions. True terror was engendered by missing the dates – when your software has to be packed into hundreds of thousands of boxes and those boxes shipped to stores in time for the day after Thanksgiving sale, there isn't much room to play catch-up. Confused users was another serious fear – you do **not** want thousands of people to start calling the help lines on the day after Christmas, unable to install their shiny new printer. Unexpected incompatibilities with popular software or hardware was yet another source of trepidation.

The definition of "quality" was substantially different in these two businesses. The customers valued different things – the medical customers valued correctness and reliability above all else, while the consumer-market customers valued usability and compatibility more than extremely high reliability. "Reliability" is another "motherhood and apple pie" word – everybody wants reliability. However, whether they realize it or not, people do value reliability in proportion to the pain inflicted on them by reliability problems. People are not happy when they have to reboot their PC occasionally, but their unhappiness pales in comparison to the anguish of people faced with a defibrillator problem. When someone goes into fibrillation, there's a five or six minute window in which the patient might be saved - there's no time to waste on equipment problems. The definition of "excellent reliability" is different in the two businesses.

The two businesses shared a common goal of profitability, but their business environments differed enough to make the businesses value different things. Consumer market windows and high-volume manufacturing realities combine to create a very high value in being on time, whereas in medical products, the liability risks often outweigh the cost of being late.

Once I had admitted to myself that nobody was going to die because of some mistake in a printer driver, I was able to relax and consider whether the practices I had learned in medical products were equally useful in the printer business. It took several more months to realize that some of what seemed to be silly or sloppy practices were actually sophisticated methods of addressing problems that didn't even occur in medical products.

Fears Drive Practices

As one would expect, people in both organizations chose actions that would reduce their fears. They employed the software engineering practices that promised to reduce the likelihood of their worst nightmares coming true. Since their nightmares were different, their practices also differed. In fact, viewed in light of the fears, the practices started to make sense. Fear of

misdiagnosis leads to detailed reviews and multiple layers of intensive testing of diagnosis routines. Fear of confusing the users leads to making an investment in formal usability testing.

Not unsurprisingly, people in the same market in similar businesses usually have similar fears and use similar practices. Common fears and values can also cut across businesses – for instance, aerospace and medical businesses bear a strong resemblance to each other. It is also possible for the same company to have different fears and values in different areas, as Hewlett-Packard did in my story (which took place prior to the spin-off of Agilent).

Businesses or software practitioners fall into groups who value similar things and have similar fears, based on similarities in both customer expectations and business environments. I've dubbed these groups *practice cultures* – groups of software practitioners who share common definitions of quality and tend to use similar practices.

Customer perception of value

Users of life-critical or business-critical software place a very high value on reliability and correctness, because the implications of incorrect or unreliable behavior are very serious. When the results of failure are merely annoyance, the users will place less value on reliability and correctness.

Customers who are early adopters frequently place a higher value on getting the newest, hottest features, and less value on whether those new features operate perfectly. Mainstream customers place a higher value on reliability.

Customers who receive training on the software don't expect as much in the way of on-line help, popup help tags, and so forth. The sophistication of the on-line help for our medical records management system lagged several years behind that of mass-market software applications, yet the customers found the software to be acceptably usable because they were trained on it as part of the on-site installation. [2]

Business perception of value

The business values the same aspects of quality that its customers value, since it wants the products to be popular. It also values aspects of quality that are irrelevant to the individual customer. These aspects have to do with the market (customers as an aggregate), and the factors other than popularity that affect profitability.

The aspects of quality valued by the business will often be communicated to the development staff in the form of product requirements, but they may also show up as business objectives or corporate guidelines and standards.

Certain aspects of quality are valued in order to sell into a particular market. A highly diverse target market will cause the business to value compatibility and usability very highly, whereas a

[2] We knew the customers were satisfied, because unsatisfied customers called Technical Support. If customers called repeatedly on the same issue, Tech Support let Development know in no uncertain terms.

business with a less diverse market can be happy with less sophistication in these aspects of quality. For instance, in-house IT applications typically don't need to achieve the same level of compatibility as that of consumer shrink-wrap software, because most companies provide their employees with relatively standardized workstations.

The business perception of value is also affected by the way in which profit is made. Being on time and within budget is more highly valued when working to a fixed-bid contract than when working on mass-market software. Seasonal market windows also play into the ability to make profit more strongly in some businesses than in others.

Profit is negatively affected when serious problems are shipped, but the potential for negative effects is higher in some businesses than in others. Regulated businesses face the possibility of being shut down as well as lawsuits from angry customers, and firmware manufacturers face extremely expensive recalls in the case of problems. When fixing bugs in the field is prohibitively expensive, a higher value will be placed on low defect escape rates.

Software Practice Cultures

Five important software practice cultures are:

- Custom systems written on contract
- Custom systems written in-house
- Commercial software
- Mass-market software
- Commercial and mass-market firmware

The inkjet printer software organization fits in the mass-market practice culture, and the medical products organization is part of the commercial software practice culture. In the description of the practice cultures that follows, it is important to remember that these are generalizations. Every testing method named is undoubtedly used by some people in every practice culture, not solely in the cultures where it is cited as being common or emphasized.

Custom Systems Written on Contract

In a fixed-bid contract, the customer specifies exactly what they want and promises a fixed sum of money to the vendor. The vendor's profit depends on staying within budget and delivering a product that works as specified. Large business applications and military software are frequently written on contract, so the typical product in this practice culture is either business-critical or life-critical. The cost of distributing post-release fixes is manageable because fixes are distributed to a known, accessible, and reasonably sized set of locations.

Vendors of custom systems usually fear:

- Incorrect results
- Running over budget
- Penalties for late delivery
- Not delivering what the buyer ordered (which may result in litigation)

In the early days, most software was part of a custom system written on contract, so much of the early writing about software testing comes from this practice culture. The sheer volume of finance and business systems ensures that plenty of authors continue to study and write about this culture. The U.S. Department of Defense funds research and publication in this practice culture, through the Software Engineering Institute as well as its own publications such as *Crosstalk*. The Capability Maturity Model (CMM) was written specifically for businesses in this practice culture, and so it recommends practices that address these fears. Unfortunately for testers, the CMM has very little to say about good testing practices.

Common testing practices in this practice culture include:

- Requirements-based testing, where detailed written requirements are turned into detailed tests.
- Automated testing, especially of transactions and batch processing.
- Extensive test documentation. There are several reasons for this. Sometimes the software is handed off to other companies for maintenance, and the tests naturally should accompany the software. Often documentation is explicitly required by the contract to demonstrate quality, or the contract requires audits that expect documentation. Documentation sufficient to demonstrate that due diligence was exercised can be helpful in lawsuits. Lastly, some of these projects involve so many people that a considerable amount of documentation is needed simply to communicate.

Custom Systems Written In-House

An alternative to contracting for finance and business systems is to write the software in-house, using the company's own employees. The economics are somewhat different from those of contracted software – the value of the work depends on improving efficiency or effectiveness in other aspects of the company's operations. The emphasis on meeting schedule is lessened, since projects are often paused and restarted depending on overall budgets. The systems may be business-critical or may be more experimental in nature. Again, fixes are distributed to a limited and identifiable set of locations.

The authors of in-house systems often fear:

- Incorrect results
- Impeding other employees' ability to do their jobs
- Having their projects cancelled

Unfortunately, it sometimes appears that IT developers don't fear impeding your ability to do your job, and may even relish making life difficult. This is often due to fear of cancellation overriding fear of dissatisfied users. The "buyer" (whoever funds IT) sometimes has goals that are at odds with the goals of the end users – for instance, a management mandate to be the first on the block with a web-enabled application may override the end users' need for usability, or hardware cost constraints may override the end users' desire for performance. This is not uncommon in situations where the end user doesn't make the purchase or funding decision.

In the in-house development practice culture, the types of software and the fears bear a strong resemblance to those of the contract software practice culture. As a result, common testing

practices in this practice culture are also similar to those of the contract software practice culture, with a couple of important exceptions:

- Test documentation is less emphasized.
- Agile and test-first development methods are more often used.

The latter especially presents some interesting testing methods, which have to be discussed in the context of the development method since they are intertwined. (See [Crispin2002] for more information).

Commercial Software

Commercial software is software sold to other businesses, rather than to an individual consumer. In this practice culture, profit depends on the more familiar economic model of selling many copies of the same item for more than it cost to design the item and make the copies. Instead of satisfying one customer's needs precisely, the profitable vendor satisfies many customers less exactly. The software is often business-critical or at least quite important to the operation of the customer's business. Since the software is in the hands of many customers in many locations, distributing fixes can be quite expensive. (Internet applications, where the customer subscribes to a centralized service, are an exception.) The customers also have an annoying habit of suing or causing other trouble if the software is seriously deficient, which drives up the cost of errors.

Vendors of commercial systems typically fear:

- Lawsuits
- Recalls
- Getting a bad reputation

Accordingly, their testing practices emphasize reliability and correctness:

- Reviews and inspections.
- "Key customers" who work with the vendor throughout the development and testing process as domain experts in the field in question.
- Alpha testing – early testing of key parts of the design under field conditions. A typical cardiograph alpha test involved the current model and a prototype model both Y-cabled to the patient. The alpha test consisted of comparing the prototype's results with the results from the current model, while the current model's results were used to treat the patient.

Mass-market Software

Mass-market software is sold to individual consumers, often at very high volumes. Profit depends on selling product above cost, frequently in market "windows" or buying seasons such as pre-Christmas. Software manufacturers don't set these windows – they predated the existence of consumer software by many decades. The potential effects of software failures on the customer are usually less serious than those in the preceding cultures, and the customers are less likely to demand or receive reparation for damages. Software failures can significantly affect the user's financial wellbeing, as in tax-preparation software, but many do fall into the category of mere annoyances.

As a result of all these factors, the fears of mass-market software vendors are significantly different than those of the older practice cultures, and their practices are correspondingly different.

Typical fears in the mass-market practice culture:

- Missing the market window
- A high rate of support calls
- Bad reviews in the press

Testing practices focus on finding problems that would result in support calls or poor reviews:

- Formal usability testing, using carefully designed studies with videotaping of participants.
- Extensive compatibility testing in a wide range of environments.
- Exploratory testing and other forms of non-prescriptive testing.[3]

Devoting most of the testing effort to requirements-based testing often doesn't make sense for mass-market software, for several reasons:

- The range of possible interactions with the operating system, other applications, and other hardware is so enormous that one cannot possibly cover more than a fraction of the space. Various sampling methods must be used, such as pair-wise combinations [Cohen1997, Kaner2002].
- The range of customer behavior is also extremely broad. "Acting like a customer" frequently reveals situations that the requirements never addressed at all.
- The development methods tend to be pretty light on written requirements – there are no contracts, regulations, or audits to require them, and everybody is in a hurry most of the time.

An explicitly risk-based approach to test planning [Bach1999b] is especially useful in this practice culture, where the range of behaviors is so enormously large.

Commercial and Mass-market Firmware

The business considerations for mass-market firmware differ enough from those for mass-market software that it makes sense to lump commercial and mass-market firmware together.

As in both commercial and mass-market software, profit depends on selling the product above cost. The cost of distributing fixes is extremely high, because the physical items have to be brought in and re-flashed or otherwise physically changed – fixes cannot be simply sent to the customer. The impact of failures in mass-market firmware is potentially more serious than the impact of software failures, since the firmware is controlling a device. While the destructive potential of small items such as digital watches is probably minimal, there are plenty of consumer devices that contain enough electronics to catch fire. Firmware failures in larger devices such as microwave ovens and automobiles can have fatal consequences.

[3] Remember those people running tests with no written test procedures? They were doing exploratory testing.

As a result, firmware developers fear:

- Incorrect behavior under any circumstances
- Recalls
- Lawsuits

The methods used by firmware developers in both commercial and mass-market firmware are very similar to those used in commercial software, especially high-reliability or regulated commercial software.

Testing practices include:

- Reviews and inspections
- Exhaustive requirements-based testing
- Formal methods

Key customers are, I think, less often used because the "right" behavior of firmware is easier to define than the "right" behavior of software.

"Best" Practices?

Some members of the various practice cultures act as if their culture was the only proper culture, and all the others are merely poor copies of theirs. Others are oblivious to the existence of other practice cultures. Much of the software engineering literature reflects these attitudes. This leads to the illusion that there are "best" practices which will work in all situations, rather than practices that are "very good" in certain situations.

When considering "best practices" in the literature, the first thing to realize is that the described practice almost certainly does work well in the author's situation. (Otherwise, they wouldn't write about it). Your task is to figure out whether your situation is sufficiently different to change the effectiveness of the practice. If you can identify the practice culture assumed by the author, you can more easily see the differences between your context and his or hers [Iberle2002]. Then you have to think about the effect of those differences on the effectiveness of the practice. James Bach offers a list of questions that may help. [Bach99a]

Summary

I believe that most people really do care about quality. Of the many people I've worked with over the years, only a very few didn't care passionately about their work. However, each business used a different definition of quality, based primarily on what mattered most to their users and secondarily on what mattered to the business itself. The practices used reflected those definitions of quality and successfully delivered products with the desired characteristics. When you start to feel that your colleagues don't care about quality, it's time to check on whether your definition of quality is the same as theirs.

References

[Bach99a]: Bach, James; “Good Practice Hunting”; Cutter IT Journal; February 1999.[Bach99b]: Bach, James; “Heuristic Risk-Based Testing”; Software Testing and Quality Engineering; November 1999.
[Cohen96]: Cohen, Daniel M., Siddhartha R. Dalal, M. L. Fredman, and Gardner C. Patton; “The AETG System: An Approach to Testing Based on Combinatorial Design”; IEEE Transactions on Software Engineering; Vol 23#7; July, 1997; Available at www.argreenhouse.com/papers/gcp/AETGieee97.shtml.
[Crispin2002]: Crispin, Lisa; House, Tip; *Testing Extreme Programming*; Addison Wesley Professional; 2002.
[Iberle2002]: Iberle, Kathy; “But Will It Work For Me? Software Engineering Practice Cultures”; Proceedings of the Pacific Northwest Software Quality Conference; 2002.
[Kaner2002]: Kaner, Cem; Bach, James; Pettichord, Bret; Lessons Learned in Software Testing; John Wiley & Sons; 2002; pp. 52-58.
[Weinberg92]: Weinberg, Gerald M.; *Quality Software Management, volume 1: Systems Thinking*; Dorset House; 1992; p. 7.