Commercial software products are often rushed to market with less regard for quality than for timeliness of release. The perversity of this approach is that tasks are allocated based on the time available instead of the time needed to do the job right. Is there an acceptable balance between such time-to-market pressures and the application of quality principles? A proper understanding of customer expectations and truly economical use of technical resources suggests such a tradeoff is possible.

Key words: customer satisfaction, productivity, project management, scheduling, tradeoffs

TALKING POINTS

Balancing Time to Market and Quality

STEVEN R. RAKITIN
Software Quality Consulting

Editor's note: "Talking Points" will feature short essays that are meant to facilitate discussion of quality principles and their application. This first "Talking Points" has been contributed by Steven Rakitin, a member of *SQP's* editorial board.

INTRODUCTION

"We are so used to the notion that quality must take a back seat to productivity that we continue to put up with practices that we know will produce software of lesser quality. Better practices, education, and tools are less important than a change in heart." (Hsia 1993)

During a typical day, most people encounter defective software. Some of the factors that affect software quality are:

- Customers often do not know what they want and frequently change the requirements.
- The work that software engineers do is technically challenging and highly complex.
- Universities spend too much time teaching programming skills and not enough time teaching the discipline of software engineering.
- Few software engineers have had formal training in essential skills such as software project management, software estimating and scheduling, risk management, software verification and validation, and software reliability growth modeling.
- Management invests little in training software engineers.
- Performance evaluations of software engineers are often based solely on productivity rather than quality and productivity.

In developing software, each line of code is hand crafted, frequently based on incomplete, ambiguous, and inconsistent requirements. Watts Humphrey (1997) has collected data that show that experienced programmers make one mistake for every 10 lines of code they write. At this rate, a product with

one million lines of code will have 100,000 defects! While 95 percent of these defects are typically found before the product is released, about 5000 defects would remain in the product, yielding a 5 percent defect rate. Clearly, people would not accept such poor quality in other types of products. Why then should they accept it for software products, which have become an indispensable part of today's society?

THE TIME-TO-MARKET VS. QUALITY TRADEOFF

Many software suppliers cannot keep pace with the demand for new products, so they frequently set overly aggressive schedules for new product development. As a result, software engineers are pressured to deliver based on the desire to meet perceived customer demand. Often, factors such as the complexity of the new product, the organization's capability and capacity, resources and the ability to add staff, past experiences, and the like are not fully considered when setting these schedules.

When the end date (time to market) is given, software engineers have to "schedule backward." This means that the project schedule is developed by starting with the end date and working backward. Scheduling backward often results in failure. Software engineers know it is wrong, yet they continue to do it.

When making the time-to-market vs. quality tradeoff, more often than not, quality suffers. From the customer's perspective, dealing with defective software is time consuming, frustrating, and inefficient. From the supplier's perspective, defective software is costly. Not only are defects expensive to fix, but they increase support costs and lower profits since expensive engineering resources are diverted from new product development (which generates revenue) to bug-fix releases (which typically do not generate revenue).

A Typical Scheduled-Backward Project

By focusing only on time to market, software engineers are forced to schedule backward. When they schedule backward, *task estimates are made based on the time available rather than the time required.* Things such as decomposing tasks to understand the scope of the problem and identifying intertask dependencies are usually not done.

With this type of schedule, consideration is seldom given to the unexpected occurrences that always happen during the course of every project. For example:

- The requirements will change.
- A key member of the team will leave the company, go on medical leave, or win the lottery.
- A key assumption about the product or technology will be wrong.
- Training needed to help people learn new tools or technology will not be planned.
- Important information one group needs to do its job will be incomplete or insufficient.
- Dependencies will arise that were previously unknown or ignored.
- Key resources will be pulled off the project to resolve problems with another product.

By ignoring the complexity, organizational capabilities, resource requirements, and the unexpected things that always happen, the schedule quickly becomes meaningless.

Sooner or later, a key task will be late or unable to be completed. For instance, changing requirements could cause the design specification to take twice as long to complete as originally planned—and a ripple effect begins. The test plans, documentation, and coding take longer than expected because the tasks were never fully understood in the first place, dependencies between tasks were never identified, contingency plans for staffing were never implemented, and so on.

At this point, the project manager begins to panic. The end date is fixed since customers have already been promised the product will be released on time. It seems the only choice is to take shortcuts. After all, the project team is being evaluated based on whether the product is released on time—not whether it does everything it is supposed to.

So what gets cut? First, the project manager abandons whatever process the team was following. The focus shifts to paring down features and cranking up coding. Activities such as regression testing, design reviews, and code inspections are eliminated. The amount of time the software quality assurance group needs for validation testing is drastically cut since it is always the last activity on the schedule. No design reviews, no code inspections, less testing, and more hurrying add up to a poor quality product. Sound familiar?

The most amazing thing about this scenario is that no matter how many times it happens, management is still:

- Appalled at the high support costs
- Upset with quality assurance for missing so many defects
- Quick to blame software engineers for doing shoddy work

Clearly, focusing *only* on time-to-market goals or only on quality goals to the exclusion of the other is not desirable. Having an acceptable product that is months late and, as a result, does not sell is just as bad as releasing an unacceptable product on time. The question is: "Can a reasonable balance between meeting *both* time-to-market *and* quality goals be achieved?"

WHAT IS ACCEPTABLE QUALITY?

For many organizations, exactly what customers expect is rarely defined. Lacking any quantitative definition, people often adopt an all-or-nothing approach, that is, either shipping the product at some preset time or spending an inordinate amount of time trying to achieve perfection. Neither extreme is good.

James Bach (1997) popularized the term "good enough quality." While some disagree with applying this notion to software, most organizations (developing software that is neither safety critical or life threatening) make the decision to ship based on some notion that the product is "good enough." They know that customers are not willing to wait for or pay for perfect software.

The objective then should be to find and fix those defects that customers are likely to find. Some mass-market software suppliers have managed to do this quite successfully. For instance, it is estimated that the original release of Windows 95 had more than 13,000 known defects that Microsoft chose to defer (Yourdon 1998). The product's success may be at least partly due to the good job Microsoft did of fixing the defects most users would encounter and find unacceptable, while deferring the rest.

Achieving Quality Software On Time

Time-to-market and quality goals are not mutually exclusive. Both goals can be achieved with the *proper motivation, a quality culture, and continuing support.*

The proper motivation

If one were to look at the performance objectives within an organization (that is, the objectives that salary increases are based on) for project managers and software engineers, he or she would probably not find the word "quality" anywhere. Yet, studies have shown that people will optimize their performance based on the objectives they are being measured against (Weinberg 1971).

Project teams should have performance objectives based on meeting both schedule and quality goals. To improve quality, management needs to make quality as important as time to market.

Establish a quality culture

"Quality is the most effective way to assure long-term profits. There is a strong relationship between 'sales' and 'product quality.' Quality is a base for mass production and mass sales. Quality contributes to a growth of total size of a market, as well as increase in market shares. Software producers have traditionally failed to understand this aspect of quality...." (Iizuka 1995)

By establishing a quality culture, management encourages people to do it right the first time. A quality culture has a number of attributes.

- *Make quality measurable.* To make quality measurable, people need to see the connection between what they do and the quality of the product or service they provide. While there are many objective measures that can be used to assess product quality, it is important that each organization establish quality measures that they then "own." Posting these measures in a central place helps reinforce the quality culture.
- *Schedule forward.* By using techniques such as yellow-sticky scheduling and the Wideband Delphi method (Weinberg 1971), organizations can develop realistic and accurate schedules—a key component of achieving quality software on time.

  What can one expect when scheduling forward? First, by having peers critique each part of the schedule, the resulting schedule is more comprehensive and more accurate. Second, project team members will be committed to meeting the schedule because the schedule is theirs. They created and therefore own the schedule—it was not forced upon them. Lastly,

the project team agrees to be held accountable for meeting the schedule. If someone misses a date for whatever reason, the team will recover without slipping the end date.

Management needs to learn to trust the schedule the project team develops. If customer expectations are set properly initially, customers will be delighted when the organization delivers quality software on time.

- *Get everyone involved in quality improvement.* The biggest improvements in software quality will come from the people doing the work. They know where the current process can be improved. Measurement data can be used to support their ideas. By getting everyone involved, employees will own the process and, as a result, be more inclined to follow it.

Provide continuing support

Management must provide continuing support to achieve quality software on time. Some examples are:

- *Learn to trust the process.* Do not allow project managers to abandon the process when the going gets tough—which is when the process is needed most.
- *Be willing to make the tough decisions.* If the project team says the product is not ready to be released, ask for the data, make an informed decision, and explain the decision to the team and to customers, if necessary.
- *Insist on having a well-defined set of product-release criteria.* Resist the temptation to relax the release criteria, especially during the final weeks of the project.

If through its actions, management demonstrates that it is serious about achieving quality software on time, the improvements in both quality and on-time delivery can be dramatic and will result in very satisfied customers.

SUMMARY

Balancing quality and time to market is difficult, but it can be done. With a strong commitment from management, the organization can benefit from lower support costs, fewer bug-fix releases, and increased profits. Customers can benefit by not having to deal with defective software. This makes customers more productive, and as a result, more satisfied. And satisfied customers are more likely to buy more of the company's products.

REFERENCES

Bach, J. 1997. Good enough quality: Beyond the buzzword. *IEEE Computer* (August): 96-98.

Hsia, P. 1993. Learning to put lessons into practice. *IEEE Software* (September): 14-17.

Humphrey, W. S. 1997. "What if your life depended on software?" Presentation to Boston Software Process Improvement Network meeting, Boston, Mass.

Iizuka, Y. 1995. A new paradigm for software quality: The turning point for the Japanese software industry. In *Software quality assurance and measurement: A worldwide perspective,* edited by N. Fenton, R. Whitty, and Y. Iizuka. London: Thompson Computer Press.

Weinberg, G. 1971. *The psychology of computer programming.* New York: Van Nostrand Rheinhold.

Yourdon, E. 1998. *The rise and resurrection of the American programmer.* Upper Saddle River, N. J.: Prentice Hall.

Author's note: Some ideas expressed in this article were based on an earlier unpublished paper "Quality On Time" by Akira Fujimura.

BIOGRAPHY

Steven R. Rakitin has more than 25 years' experience as a software engineer and software quality manager in a broad range of industries, including nuclear power, defense, computers, automated test equipment, telecommunications, medical instrumentation, and electronic design automation. He was one of the authors of the IEEE Standard for Software Quality Assurance Plans (IEEE-STD-730). He has written several papers on software quality and recently published a book titled *Software Verification & Validation: A Practitioner's Guide.*

Rakitin has a bachelor's degree in electrical engineering from Northeastern University and a master's degree in computer science from Rensselaer Polytechnic Institute. He has earned certifications from ASQ as a software quality engineer (CSQE) and a quality auditor (CQA). He is a member of the IEEE Computer Society and the ASQ Software Division, and is on the editorial board of *Software Quality Professional.* Rakitin is software quality chair for ASQ's Boston Section and is adjunct professor at Northeastern University where he teaches a course on software verification and validation. His company, Software Quality Consulting, works with companies interested in developing a predictable software development process. He can be contacted at Software Quality Consulting, 21 Whitney Ln., Upton, MA 01568, or by e-mail at srakitin@ma.ultranet.com .