Food for Thought™

An e-newsletter published by
Software Quality Consulting, Inc.

January 2007
Vol. 4 No. 1

Welcome to **Food for Thought™**, an e-newsletter from Software Quality Consulting. I've created free subscriptions for my valued business contacts. If you find this newsletter informative, I encourage you to continue reading. Feel free to pass this newsletter along to colleagues by using this Forward Email link. If you've received this newsletter from a colleague and would like to subscribe, please use this Enter New Subscription link. If you don't wish to receive this newsletter, use the SafeUnSubscribe link at the bottom of this newsletter, and you won't be bothered again.

Your continued feedback on this newsletter is most welcome. Please send your comments to **steve@swqual.com**

In This Issue

In This Months' Topic I discuss barriers to quality…

Regular features to look for each month are:

- Monthly Morsels
  Hints, tips, techniques and references related to this month's topic
- Calendar
  Conferences, workshops, and meetings of interest to software engineers, QA engineers and anyone interested in software development

This Month's Topic

Barriers to Quality

It was about a year ago that my e-newsletter **All Software is Defective** appeared. Since that time, I have been talking on this topic to groups across the US. The message I've been trying to get out is that defective software is causing more problems in more products and at a frequency that appears to be increasing. And as an industry, we need to do something about this…

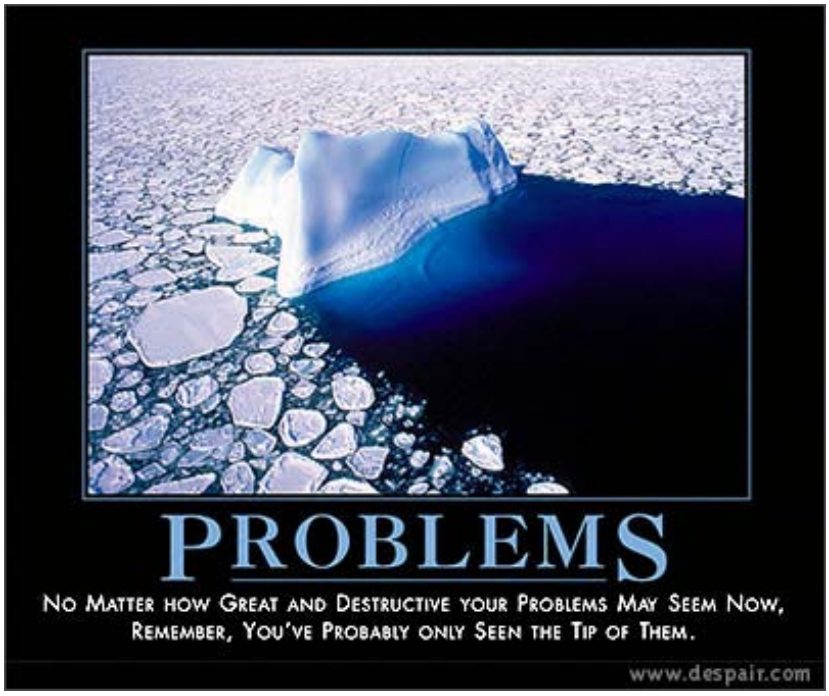
I've raised this concern in a paper published in IEEE Computer with regard to software embedded within medical devices. The problems we face are far-reaching and affect all products that include software. It seems that on an almost daily basis, we hear of new software problems. Here's a recent sampling:

- "[In Nov 2005] automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly. But unlike the large-scale auto recalls of years past, the root [cause] of the Prius issue wasn't a hardware problem -- it was a programming error in the smart car's embedded code. The Prius had a software bug." [1] An automotive software specialist indicated that the automobile industry spends **$2 billion to $3 billion** per year fixing software problems.
- Millions of bank accounts were impacted by errors due to installation of inadequately tested software code in the transaction processing system of a major North American bank, according to mid-2004 news reports. Articles about the incident stated that it took two weeks to fix all the resulting errors, that additional problems resulted when the incident drew a large number of e-mail phishing attacks against the bank's customers, and that the total cost of the incident could exceed **$100 million**.

- According to news reports in April of 2004, a software bug was determined to be a major contributor to the 2003 Northeast blackout, the worst power system failure in North American history. The failure involved loss of electrical power to 50 million customers, forced shutdown of 100 power plants, and economic losses estimated at **$6 billion**. The bug was reportedly in one utility company's vendor-supplied power monitoring and management system, which was unable to correctly handle and report on an unusual confluence of initially localized events. The error was found and corrected after examining millions of lines of code.

- "In January 2004 a special election was held in Broward County, Florida. Only one contest was included on the ballot. Yet, of the 10,844 votes cast on ES&S (Election Systems & Software) paperless touch screen voting machines, 134 were … for no one at all. Since the winning candidate won by only 12 votes, people understandably wondered what had become of those 134 votes; there was no way of telling if some had been lost by the computer. The mayor of Broward is now calling for paper ballots." [2]

As more and more complex applications are developed and put into use, there will be more highly visible software disasters and these disasters will come with staggering price tags.



The question is - What are we doing about this?

In my opinion, not nearly enough. This is clearly a multifaceted problem. One aspect of this problem I've observed is that many organizations have inadvertently created barriers that prevent people from doing quality work. Often, these barriers are not obvious...

What are these barriers? Here are a few examples – there are many more:

- short term thinking
- incentive systems that reward wrong behaviors
- lack of effective and timely training
- ineffective infrastructure
- focus on individual rather than team performance

We need to identify and remove barriers like these whenever possible. This month, we'll look at one of these barriers – short-term thinking…

Short-term Thinking as a Barrier to Quality

Short-term thinking occurs when decisions are made that satisfy an immediate concern but turn out to be detrimental to the business and/or customer in the long run. Many software companies are burdened with quality problems that are the result of short-term thinking. For example:

- **Short-term thinking and Requirements**

  Here's an area where short-term thinking can have a huge affect on quality. Many organizations view requirements as "optional" and they're written down only "if we have time." But the old adage still holds true:

  We never seem to have time to do it right,
  but always have time to do it over.

  Further, in far too many organizations, the prevailing management mentality is that developers should spend 100% of their time coding.

  In fact, we should have learned by now that coding but one of many tasks developers should be involved in. If developers spend more than 40% of their time coding, that's cause for concern. What tasks are they **not** doing that they should be doing?

  Short-term thinking also prevents teams from fully understanding the implications of requirements changes once work is underway. This obviously leads to more problems and delays.

- **Short-term thinking and Architecture / Design**

  An area of software development that is often overlooked is architecture and design. This often results in poorly architected applications that lack important attributes like conceptual integrity…

  Again, when developers are focused solely on coding, this is an example of one of the areas that suffers.

- **Short-term thinking and Coding**

  Coding standards are an effective tool for preventing quality problems. By identifying common coding mistakes and recommending good practices, many subtle coding problems can be avoided.

  Does you organization have coding standards? Are they enforced? Here again, we often allow short-term thinking to short circuit good engineering practices by either not having or having but not enforcing reasonable coding standards.

  To be effective, coding standards need to be developed by developers, not managers. To reinforce the importance of coding standards, they need to be enforced by developers – usually as part of peer reviews.

- **Short-term thinking and Peer Reviews**

  All the evidence from over three decades of experience shows that effective peer reviews are a powerful tool for identifying defects when they are cheaper and less time-consuming to remove.

  Yet, in spite of this overwhelming data, many organizations ignore choose to not include peer reviews in project plans. These short-term thinkers seem to believe that the fewer tasks between here and the delivery date, the more likely it will be to meet the date. What they don't understand is that it takes

more time to find and fix show stoppers when you find them in test as compared to finding the same defect earlier…

Managers seem to understand that finding defects sooner rather than later is more cost-effective but when it comes time to make decisions, they often ignore this fact. Why?

- **Short-term thinking and Testing**

  Test results are often viewed as the **only** indicator of quality when in fact, test results are but one of many indicators of quality. Short-term thinkers tend to harp on test results and often will cut what little time is set aside for testing in order to meet arbitrary release dates.

  When we allow short-term thinking to impact testing, the result is releasing products with far too many defects. This means more unplanned and resource-draining bug fix releases. Not to mention the disruptions to your customers and the extra calls to support…

What can we do about short-term thinking?

Breaking people from the habit of short-term thinking is clearly not an easy task. Especially in today's business climate where "immediate gratification" is the mantra of many 30-something managers. Changing the culture requires firm commitment and leadership from the most senior levels of management.

And it isn't just managers who are guilty of short-term thinking. Young software engineers are guilty of this and more. As Ed Yourdon [3] observed:

> "Another reason the software industry continues to have this specific problem is that the third and fourth generations of software developers do not care to learn anything from their elders. They probably assume that anybody who is old enough to have programmed in COBOL cannot possibly have anything valuable to offer from their experiences. The result is that younger engineers in their 20's and 30's wind up learning these lessons the hard way; it's déjà vu all over again."

Here are some steps that can be taken to help identify potential quality problems that may result from short-term thinking:

- **Recognize the difference between importance and urgency**

  If something is important to the organization in the long-term, it should also be important in the short-term.

  Many "urgent" things really aren't urgent, and if they aren't important to the organization in the long-term, they may not need to be addressed at all.

- **Explain the pain**

  Remember in the movie **The Karate Kid** how Daniel was frustrated doing all that work around Mr. Miyagi's house? Once he saw the final outcome, he realized how valuable doing that work was.

  Explain how long-term thinking will pay bigger dividends in the form of fewer unplanned bug fix releases, higher customer satisfaction, lower call volume to support, etc.

- **Understand that software products are more than just code**

  Many software organizations have a laser-like focus on code as if nothing else mattered. While developers like to think that's the case, it isn't. There's user documentation, training, support and other services that are part of your product offering. Short-term thinking often excludes these other valuable aspects of your offerings. These other parts of the organization need to push back when

necessary to ensure that the quality of these services is not compromised by code-focused short-term thinking…

Summary

The problems that the software industry is facing will only be getting more difficult. As Capers Jones [4] recently reported, software development organizations should expect:

- Most delays are due to poor quality and requirements creep

  Short-term thinking doesn't acknowledge this fact…

- Requirements will change/grow at >2% per month

  Short-term thinking often prevents the impact of changes to requirements from being fully understood, resulting in more defects and delayed releases.

- Defects and error rates will often exceed 5 defects per function point

  In terms of lines of code, Watts Humphrey measured the average defect injection rate for a sample of experienced developers to be 1 defect for every 8 lines of code. Defect injection rates are higher for less experienced developers.

  Short-term thinking often precludes actions that could lead to fewer defects being injected.

- Cumulative defect removal rates (peer reviews and testing combined) are often less than 85% and testing defect removal rates are often less than 70%

  This means that customers are finding a significant percentage of defects. Testing teams need training to help them learn the most effective techniques for testing. Peer reviews can be very effective in defect removal – if they are done properly.

  Short-term thinking often prevents effective testing and effective peer reviews from happening.

And lastly, this insight from a really smart guy:

> We can easily forgive a child who is afraid of the dark;
> the real tragedy of life is when adults are afraid of the light.
>
> -- Plato --

Until next time…

Every month in this space you'll find additional information related to this month's topic.

- **References:**

[1] History's Worst Software Bugs, Wired News, Nov 8, 2005

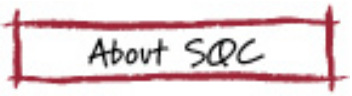
[2] B. Simons, "Electronic Voting Systems: the Good, the Bad, and the Stupid," paper submitted to the National Academies.

[3] Focus on Ed Yourdon, A CAI State of the Practice Interview, September, 2006

[4] Jones, C., "The Economics of Software Process Improvement", Boston SPIN Presentation, Dec 2005.

- **On-line Resources:**

Business Pundits.com Short Term Thinking – It Kills Startups Too



Software Quality Consulting provides consulting, training, and auditing services tailored to meet the specific needs of clients. We help clients fine-tune their software development processes and improve the quality of their software products. The overall goal is to help clients achieve Predictable Software Development™ – so that organizations can consistently deliver quality software with promised features in the promised timeframe.

To learn more about how we can help your organization, **visit our web site** or **send us an email**.