



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

September 2009 , Vol. 6 No. 5
[[Text-only Version](#)]

Welcome to **Food for Thought**TM, 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 clicking this **Forward Email** link. If you've received this newsletter from a colleague and would like to subscribe, please click this **Enter New Subscription** link. If you don't wish to receive this newsletter, click the **SafeUnSubscribe**TM link at the bottom of this newsletter, and you won't be bothered again.

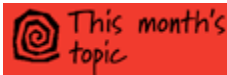
Your continued feedback on this newsletter is most welcome. Please send your comments and suggestions to info@swqual.com.

In this issue

In **This Month's Topic**, I begin a discussion on the state of the software quality assurance profession...

Regular features to look for each month are:

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



Software Quality Assurance turns 50 A critical look at the state of the profession

Part 2 - Present State of the Profession

Software Quality Assurance (SQA) as we know it was first applied to software development projects about 50 years ago. To recognize this important milestone, the state of the SQA profession is the topic for this issue of my newsletter. In my **June newsletter**, I discussed the history and evolution of SQA. In this month's installment, I discuss some SQA successes and failures. The last installment will focus on the future of SQA.

Present Situation

Today the best most highly skilled software developers inject an average of one defect for every 8 lines of code they write. [2] The single most common reason software engineers inject defects has been and is still poorly written requirements.

We also have anecdotal information suggesting that through verification activities such as peer reviews and static analysis and validation testing, we typically find about 95% of these injected defects. The end result is:

released software has, on average, a defect density in the range of 5-6 defects per thousand lines of code (KLOC).

So if we look at a typical software-based system that has one million lines of code, here's what we'd expect to find:

Defects injected: using 1 defect /8 lines of code =
~120,000

Defects removed: assuming 95% found = 114,000

Defects remaining: (defects injected - defects removed) = **6,000**

To put this in perspective, 2010 model-year cars will have about 100 million LOC. Given the above, there could be as many as **600,000** defects that remain in the software that controls your car.

- **Read more about safety-critical software...**

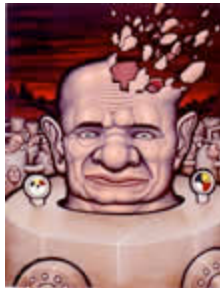
Recently, a prestigious group of researchers from the **National Research Council** published a book on the issues related to developing software for dependable systems. Here's their sobering assessment of the present situation:

"Society is increasingly dependent on software. Software failures can cause or contribute to serious accidents that result in death, injury, significant environmental damage, or major financial loss. Such accidents have already occurred and without intervention, the increasingly pervasive use of software - especially in arenas such as transportation, health care, and the broader infrastructure - may make them more frequent and more serious." [1]

The problem is exacerbated by a pervasive **lack of evidence** about both the incidence and severity of software failures. This lack of evidence has led the National Research Council researchers to the following conclusions: [1]

- Better **evidence** is needed so that approaches aimed at improving the dependability of software can be objectively assessed.
- For now, the pursuit of dependability in software systems should focus on the **construction and evaluation of evidence**.

What we must recognize is that software engineering and SQA best practices are **necessary but not sufficient** to ensure that software



Complexity is increasing so rapidly that software engineers cannot possibly understand everything about how software

works...

systems are dependable and safe. This has become painfully clear as most agree that current software development methods have failed to keep pace with the exponential increase in software complexity.

“An awareness of the need for simplicity comes only with bitter experience and humility gained from years of practice. There is no alternative to simplicity. Advances in technology or development methods will not make simplicity redundant; on the contrary, they will give it greater leverage.” [1]

While there have been some successes...

While the outlook is generally bleak, there are a few positive results that are noteworthy:

- **Testing**

As observed by Tony Hoare:

“The real value of tests is not that they detect bugs in the code but that they detect inadequacies in the methods, concentration, and skills of those who design and produce the code.” [4]

The software industry has finally recognized that testing is an indispensable part of the software development process. Today, most every software development organization has some form of testing group. We have also recognized that in some cases, testing can find defects more easily than other methods.

The fact that anecdotal evidence indicates that we are capable of finding up to 95% of injected defects is certainly encouraging.

- **Test Automation**

Test automation is another bright spot. Today, the array

of test automation tools available is staggering. More and more software development organizations are turning to test automation as a way to perform more rigorous testing more often and as a result, enable testers to focus on testing the more challenging aspects of complex applications. More and more organizations are using test automation in conjunction with nightly builds as a way to catch problems at the point they are introduced.

- **Review a list of test automation tools...**

- **Independent Verification and Validation (IV&V)**

Large government organizations have recognized that for their software development programs to succeed, they need to include an IV&V function. Independence is one of the reasons IV&V has been so successful. As a result, organizations like NASA now require IV&V on all programs that involve software.

- **Static Analysis and Formal Methods**

A new crop of static analyzers has emerged in the past few years to help catch problems before testing begins. These tools can find things like memory leaks and improper use of pointers that often can result in bugs that are difficult to find.

In addition, there has been renewed interest in applying **formal methods** to complex, safety-critical applications. These methods can also be very effective in finding problems that would otherwise be very difficult to find.

- **Review a list of static analysis tools...**

- **Training**

Florida Institute of Technology has been offering advanced courses on **Software Testing** developed under

the guidance of **Cem Kaner...**

The IEEE has published a **Software Engineering Body of Knowledge (SWEBOK)** that includes both software quality and software testing skills.

And lastly, amazon.com currently lists almost 14,000 books with the words **software** and **quality** in the title.

- **Review a list of additional training courses...**

There have been many more failures...

Software complexity has increased exponentially over the last five decades. The engineering discipline we use to develop and test software hasn't kept pace with this increase in complexity. As a result, we still have problems developing large, complex systems.

- **Read about some high profile software failures**

The **Standish Group's CHAOS 2009 Report** shows a marked **decrease** in project success rates, with

- **32%** of all projects surveyed were **successful** - defined as projects delivered on time, on budget, with required features and functions.
- **44%** of all projects surveyed were **challenged** - defined as late, over budget, and/or with less than the required features and functions
- **24%** of all projects surveyed **failed** - defined as cancelled prior to completion or delivered and never used.

The trend over the past 15 years is also very revealing:

Year	'09	'06	'04	'02	'00	'98	'96	'94
-------------	------------	------------	------------	------------	------------	------------	------------	------------

Successful Projects	32%	35%	29%	34%	28%	26%	27%	16%
Challenged Projects	44%	19%	53%	15%	23%	28%	40%	31%
Failed Projects	24%	46%	18%	51%	49%	46%	33%	53%

Note that some people have **questioned** the methods used by the Standish Group in their surveys.

Let's look at other areas where we still have work to do...

- **Software Quality Assurance**

Many software development organizations have yet to recognize that SQA is much more than testing. As Roger Pressman [5] has stated, SQA is an umbrella for many value-added activities that can significantly improve product quality.



SQA is still not recognized as a legitimate and important discipline. There are no universities that offer any kind of undergraduate degree in Software Quality. As a result, most people who are in an SQA role have few choices when seeking training in their field. This is a sad situation and needs to be rectified...

As observed by Watts Humphrey:

"SQA is a valid discipline in its own right and people can be SQA experts without being software design experts. This SQA expertise is what is required to establish a strong quality program. It includes knowledge of statistical methods, quality control principles, the software process, and an ability to

deal effectively with people in contentious situations." [3]

- **Testing and Test Tools**

While there have been positive improvements in these key areas, there have still been many failures attributable to ineffective testing. This can be partly due to the:

- lack of education and training resources available to testers
- increasing complexity of software systems
- lack of domain knowledge

Test automation tools also have a way to go as well. While there have been many significant improvements in tools, many test automation tools require programming experience and are far too complicated and expensive for many organizations.

- **Requirements**

After 50 years of experience, you'd think that we would realize just how important clear, unambiguous requirements are. Sadly, many organizations just don't get it. They rush into writing code without understanding what it is they are supposed to develop. Developers guess - and as you could imagine - they only guess right half of the time.

- **Metrics**

We still have difficulty defining metrics in ways that can lead to meaningful comparisons across projects - even within the same organization. We need to agree on a small number of basic measures that can be applied across a wide variety of applications and organizations so that we can determine which projects are more effective at meeting project quality goals.

- **Evidence of Dependability**

One of the key findings of the **National Research Council's** study is that we need to focus on providing evidence that software is dependable. This evidence needs to reflect specific claims for dependability made by the developing organization.

The list of areas for improvement is very long - the items I've described here are ones I believe are most important...

The Bottom Line....

There have certainly been some notable improvements in **software quality assurance** over the past 50 years. Unfortunately, these improvements have been overshadowed by several high-profile failures. To me, this indicates that the improvements in process and practice have not kept pace with the increases in complexity.

Next month, we'll discuss the future of SQA...

'Til next time...



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

References

1. Jackson, D., *et. al.*, *Software for Dependable Systems - Sufficient Evidence?* National Research Council, National Academies Press, 2007.
2. Humphrey, W., "The Quality Attitude", *news@seinewsletter*, Number 3, 2004.
3. Humphrey, W. S., *Managing the Software Process*, Addison-Wesley, 1989.

4. Hoare, C. A. R., "How did software get so reliable without proof?", *Lecture Notes in Computer Science*, 1051:1-17, 1996.
5. Pressman, R., *Software Engineering – A Practitioners Approach*, 4th edition, McGraw-Hill, 1997.



Calendar

Every month you'll find news here about local and national events that are of interest to the software community...

- **Software Quality Calendar**

There are many organizations that sponsor monthly meetings, workshops, and conferences of interest to software professionals. **Find out what's happening...**

- **Workshops Offered by Software Quality Consulting**

Software Quality Consulting offers workshops in many topics related to software process improvement. **Get more info...**



About SQC

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**.

I hope this newsletter has been informative and helpful. Your comments and feedback are most welcome. **Send me your feedback...**

Thanks,



Steve Rakitin

info@swqual.com

Software Quality Consulting Inc.	
Steven R. Rakitin President	<ul style="list-style-type: none">• Consulting• Training• Auditing
Phone: 508.529.4282	www.swqual.com
Fax: 508.529.7799	info@swqual.com

Food for Thought, Predictable Software Development, Act Like a Customer,
and ALAC are trademarks of Software Quality Consulting, Inc.

Copyright 2009. Software Quality Consulting, Inc. All rights reserved.

Graphic design by **Sarah Cole Design**.