Food for Thought™

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

June 2009 , Vol. 6 No. 4
[Text-only Version]

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 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**™ 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 Months' 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

This month's topic

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

Part 1 - History and Evolution

Software Quality Assurance (SQA) was used for the first time on a software development project about 50 years ago. Over the next several months, I plan to take a critical look at the state of the SQA profession as a way to recognize the significance of this milestone. In this month's installment, I discuss the history and evolution of SQA.

In subsequent e-newsletters, I will discuss some successes and failures, and the future of SQA.

During the past half-century, the software industry has gone through dramatic changes. Today, software is an integral part of daily life. Many software-based products that were beyond one's wildest imagination five decades ago are now commonplace. The explosion of the Internet, digital gadgetry, and cheap hardware has resulted in software finding its way into millions of products and services, many of which are **safety-critical or mission-critical**. Today **software plays an integral role in most every major segment of the global economy**. The following are but a few examples:

- **Energy:**
  - Electric power generation - nuclear and conventional power plants
  - National and regional power grids
  - Oil and natural gas distribution management systems
- **Transportation:**
  - All kinds of vehicles
  - Urban mass transit systems
  - Railway signaling systems
  - Avionics and air traffic management
- **Healthcare:**
  - Hospital patient monitoring systems
  - Life-supporting and life-sustaining medical devices
  - Electronic patient medical records
  - Research into diseases and development of new drugs
- **Banking and Finance:**
  - On-line banking systems and ATM machines
  - International currency trading
  - Stock exchanges and brokerages
- **Defense:**
  - Weapon systems
  - Command, Control, and Communications Systems
  - Satellite communications and imaging
- **Space exploration:**
  - Hubbell Telescope
  - International Space Station
  - Space Shuttle

Everyday we become more and more dependent on software. As I have stated many times, **all software is inherently defective**. Unless you live in a third world country, a typical day involves using a

significant amount of software - either implicitly (as in software embedded in a product) or explicitly (as in software applications). For people living in third world countries, there are **on-going efforts** (some controversial) aimed at providing children with inexpensive laptops so they can connect to global virtual communities.

We begin this critical review by starting at the beginning...

In the beginning...

In the late 1950's, software first began to find its way into systems procured by US government agencies such as the Census Bureau and the Dept. of Defense (DoD). Not surprisingly, these projects were always behind schedule, over budget, and suffered from both technical and management problems. Frequently, software did not work as intended and many projects were cancelled before anything was delivered.

Software development contractors often gave overly optimistic assessments of the software development status to managers - the origin of "We're 90% done!" Managers were frequently unaware of schedule, budget, and technical problems until very late into the program – when they were often unable to understand them, assess their impact or do anything to change the situation.

The Atlas Missile was the first operational intercontinental ballistic missile in America's nuclear arsenal and marked the beginning of the US space program. The **Atlas Missile Program** was one of the first software projects to try to address these problems. The program manager hired an "independent software tester" to "perform additional, unbiased testing of the software". [1] The program manager hoped to get a timelier, accurate and objective technical assessment of the project's status by employing someone **independent** of the software development contractor.

Around the same time, the first independent test team on a large software project was formed and led by **Jerry Weinberg** on **Project Mercury** - the first US manned space flight program.

During the 1960's, the role of the independent software test team evolved from just focusing on testing to focusing on the entire software development life cycle. This role became known as **Independent Verification & Validation (IV&V).**

Today, IV&V is a critical function contractually required on most large, mission-critical projects for US government agencies including DoD, NASA, FAA, HUD, EPA and DEA. The set of tasks performed by IV&V contractors is comprehensive and spans procurement, development and deployment.

- To learn more about IV&V see the **Monthly Morsels** section below...

Much data has been collected to support the assertion that projects with IV&V perform much better than similar projects without IV&V. [2], [3] As a result of this data, NASA now requires IV&V to be applied on applicable NASA projects. [4]

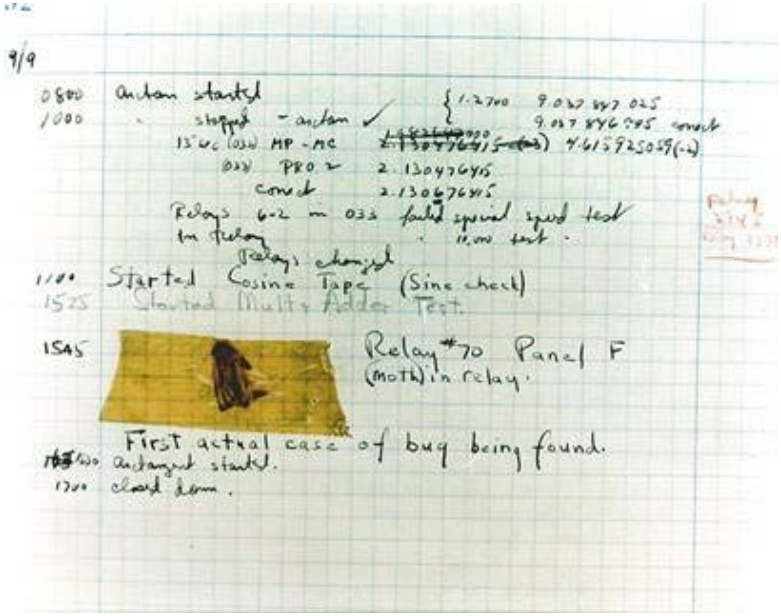
Much of the success of IV&V is attributable to the fact that IV&V contractors are completely **independent** of the software development organization. Working for and reporting to the procuring entity, IV&V contractors **provide an unbiased, objective technical and managerial assessment** of a project. As a result, the procuring entity is in a much better position to identify and resolve issues that could otherwise easily be overlooked (intentionally or unintentionally) by the software development contractor. Raising these issues in a timely manner ensures that they are more likely to be resolved and not affect the project.

When did it become Software Engineering?

In the really old days (1940s-50s), people who worked with computers were usually mathematicians and were called **programmers** or **data processing specialists**. Computers of the time, such as the **ENIAC**, were very clunky and were usually "programmed" by re-wiring patch panels that changed the sequence of operations the computer's electromechanical relays performed.

Back then, designing hardware was much more prestigious and as a result, hardware engineers were mostly male. Women, most of whom were mathematicians like **Admiral Grace Hopper**, often performed the "lowly" job of programming the computers.

> Grace Hopper was a brilliant mathematician who worked at Harvard on the **Mark II Aiken Relay Calculator** – an early analog computer built from hundreds of electromechanical relays. She liked to tell a story about an event that occurred in late summer of 1947. It was before the advent of air conditioning so the windows in the computer lab were open most of the time. A technician solved a problem with the Mark II machine by pulling an actual insect (a moth) out from between the contacts of one of its relays. Admiral Hopper taped the moth to her lab notebook on September 9, 1947 and made the entry shown below:



In the fall of 1968 and again in 1969, the **NATO Science Committee** sponsored technical conferences to bring together experts to discuss problems with the development of software. The conference organizers used the term **software engineering** as a way to provoke interest and discussion. The term wasn't used prior to the conference and the birth of the software engineering profession is generally recognized to coincide with these conferences.

As observed by one of the conference attendees:

> "Although the term was not in general use at that time, its adoption for the titles of these conferences was deliberately provocative. As a result, the conferences played a major role in gaining general acceptance, perhaps even premature, for the term. The motivation for these conferences was that the computer industry at large was having a great deal of trouble in producing large and complex software systems. (Does that sound like dêja vu all over again?)" [5]

The participants at the conference represented computer hardware manufacturers, computer users, representatives from a few small software development companies, and academia. Most of the people attending the NATO Conference acknowledged that there were many problems associated with producing large, complex software systems. (Note that systems called "large" and "complex" in 1968 terms would be considered "small" and "trivial" by today's standards). In the summary of the NATO Science Committee report [6], it states:

> "... the report also contains sections reporting on discussions, which will be of interest to a much wider audience. This holds for subjects like the:

- problems of achieving sufficient reliability in the data systems which are becoming increasingly integrated into the central activities of modern society

- difficulties of meeting schedules and specifications on large software projects

- education of software (or data systems) engineers”

The problems observed in 1968 are striking in how similar they are to problems we have today.

The emergence of SQA

The 1968 NATO report also used the term **Software Quality Assurance**. During the conference, the participants discussed the issue of SQA and raised several very interesting questions: [6]

- Is software quality assurance done by an independently reporting agency representing the interests of the eventual user?

- Is the product tested to ensure that it is the most useful for the customer in addition to matching functional specifications?

- Do software quality assurance test programs undergo the same production cycle and method (except Q/A) as the software they test? Are they defined and constructed concurrently with the software?

- Is at least one person engaged in software quality assurance for every ten engaged in its fabrication?

- Are there tests for overall system performance as well as for components?

- Are software quality assurance tests a part of the general hardware acceptance test on the customer’s machine before it leaves the factory’?

- Can software field release be held up if these tests are not passed?

- Do the tests include a system logic exerciser?

- Are tests provided to ensure matching of computational results with those of other equipment?

- Is this test library applied upon issuance of each modification of the software system?

- Is each customer's system tape tested on the software production machine for a sufficient period of time, where feasible?

These issues, raised more than 40 years ago, still resonate today.

During the 1970's, software development activity expanded to commercial companies. These companies experienced the same poor results that US government agencies had seen a decade earlier. These companies had difficulty delivering software within the constraints of schedule, budget, and quality. Many projects undertaken in the 1980's and 90's were disasters. Several projects failed to deliver anything. The few projects that did deliver something were significantly over budget and years behind original schedules and delivered software of such poor quality that it was often unusable.

In the 1980's, the software industry experienced what became known as the "software crisis" – the point in time when spending on software maintenance exceeded spending on creating new software products. The advent of the "software crisis" brought with it a host of changes - not the least of which was the emergence of SQA as a critical function to be performed on software development projects. Initially, SQA was viewed as sort of an internal IV&V function.

Drawing on its roots in IV&V, SQA evolved into an effective tool that software development companies have used to help identify quality problems earlier in the development process. While SQA was viewed as the "poor stepchild" of software development, many enlightened managers of the day saw measurable benefit from integrating SQA into the software development process.

By the 1990's, many software companies had SQA functions within their organizations. Yet, high profile software failures continued to occur. (see [7, 8, 9]) Was SQA not living up to expectations? Hard to say. But there were several differences in the nature of software being developed during this time that are worth noting:

- Complexity of software developed during the 90's increased significantly.
- Competitive business pressures also increased significantly.
- Software was being used in many new areas – especially areas that were life threatening.
- Many people working in SQA received little formal training in SQA. SQA engineers were expected to learn their craft primarily from on-the-job training.
- Universities failed to recognize that SQA is a legitimate discipline unto itself and that it requires specialized training.

The Bottom Line....

Edsger W. Dijkstra was one of the distinguished participants at the 1968 NATO Conference. He made an incredibly insightful remark at this conference that, while important in 1968, is even more important today:

> "The dissemination of knowledge is of obvious value — the massive dissemination of error-loaded software is frightening." [6]

Til next time...

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

References

1. Nelson, J. G., "Software Testing in Computer-Driven Systems," in *Software Quality Management*, ed. Fisher, Matthew J., and Cooper, John D., Petrocelli Books, 1979.

2. Arthur, J. D. and Nance, R. E., "Verification and Validation Without Independence: A Recipe for Failure", Proc. 2000 Winter Simulation Conference, Orlando FL, 2000.

3. Wallace, D. R., and Fuji, R. U., "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards", National Institute of Standards and Technology, Special Publication 500-165, May 1989.

4. NASA Policy Directive NPD 8730.4A, Effective August 1, 2001.

5. Robert M. McClure, **Introduction to the 1968 NATO Software Engineering Conference**.

6. **Software Engineering, Report on a conference sponsored by the NATO Science Committee Garmisch**, Germany, October 7-11, 1968

7. Glass, R., Software Runaways: *Lessons Learned from Massive Software Project Failures*, Prentice-Hall PTR, 1997.

8. Weiner, L., *Digital Woes: Why We Should Not Depend On Software*, Addison-Wesley, 1993.

9. Johnson, J., "Chaos: The Dollar Drain of IT Project Failures," Application Development Trends, January 1995, pp. 41-47.

IV&V Resources

- Lewis, Robert O., ***Independent verification and validation: a life cycle engineering process for quality software***, Wiley-IEEE, 1992.
- Schulmeyer, C. G. and Mackenzie, G. R., ***Verification and Validation of Modern Software-Intensive Systems***, Prentice Hall-PTR, 2000.
- Rakitin, S., ***Software Verification and Validation for Practitioners and Managers***, 2nd edition, Artech House 2001.
- Arthur, J., et. al., "**Evaluating the Effectiveness of Independent Verification & Validation**," *IEEE Computer*, October 1999.
- **NASA IV&V Overview**.

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…**

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

- Consulting
- Training
- Auditing

Phone: 508.529.4282
Fax: 508.529.7799

www.swqual.com
info@swqual.com