

The Software Quality Certification Triangle

Jeffrey Voas

Reliable Software Technologies

Abstract

The problem of certifying software's quality is being attacked in three distinct ways: (1) accrediting personnel, (2) certifying the development organization, and (3) assessing the goodness of the software. Here, we will discuss these approaches and briefly discuss how hybrid approaches can be derived from them depending on the type of software that needs certification.

1 Introduction

Quality software is often considered elusive. It is hard to achieve, and even harder to determine whether it was achieved. That is, it is more difficult to confidently know that you have developed good software than it is to build good software. In the physical sciences, the reverse is true: it is easier to measure the degree of perfection than it is to achieve perfection.

One reason why it is so hard to measure software quality stems from the many practical and theoretical deficiencies of software testing. For example, consider that to be 99% confident that a program has a probability of failure of less than one in a million, you must test the software over 5 million times without ever observing a failure. Testing 5 million times requires that you have an oracle that is correct (an *oracle* is the person or other software program that knows what the correct software output is for each of those 5 million test cases). Rarely does a perfect oracle exist, and creating 5 million test cases could prove intractable. And even if you have the oracle and test cases, you still must test using them.

Problems such as these have made many in the community decide that *quality assessment of a software product is impractical*. This has led toward alternate approaches to software quality assessment. The key two competing approaches are: (1) process maturity assessment, and (2) accreditation of software professionals. The remainder of this article will look at the pros and cons of these three approaches (certifying the Product, Process, and Personnel). All of these approaches are aimed at predicting the quality of software.

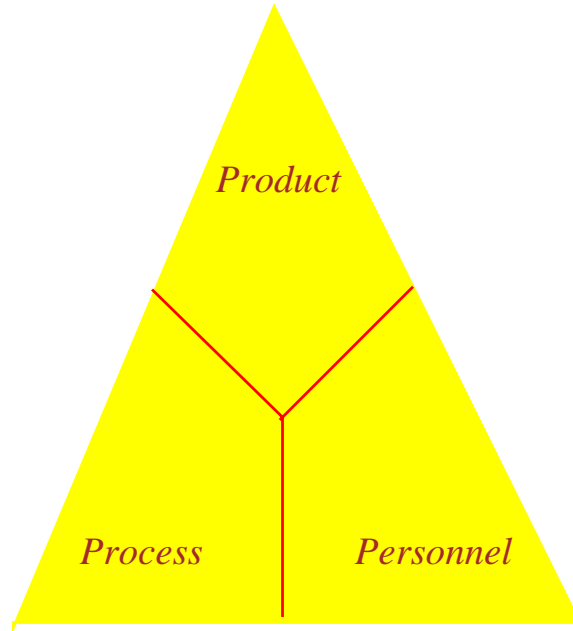


Figure 1: The Software Quality Certification Triangle

2 Accrediting Personnel

There are various ways to accredit (i.e., certify) personnel. The rigor with which personnel are certified will depend on the criticality of the services that the person will offer.

Professional licensing examinations, practical experience, and earned degrees are different ways in which professionals can be accredited. For example, graduating from law school says something about a person's ability to practice law. It says, less, however than had the person also passed the bar. If this were not true, there would be no need for state bar exams.

The intuition behind certifying "people skills" is simple: it should not be left up to the untrained consumer to be responsible for determining whether a candidate is qualified to perform the services that the person claims to be. For example, how can Joe Public be expected to determine whether a dentist is truly qualified? Only if Joe Public were a dentist would he have any hope of making such a determination. By forcing dental school graduates to pass an examination prepared by dentists, the state takes the responsibility away from Joe Public. Further, if a certified professional does not live up to the expectations of his or her peers, the professional could be found liable and could lose their certification.

Like the more traditional professions (accounting, medicine, lawyers, pilots), the software industry is beginning to standardize on core principles that each professional should know. Microsoft claims that there are greater than 160,000 individuals that have become Microsoft certified as either product specialists, solution developers, trainers, and systems engineers [2]. This type of certification is *voluntary* and expensive; however the costs of attaining it

can be made up in the first year of working from the extra income that the certificate will enable. For example, to become a Microsoft Certified Systems Engineer (MCSE), you can expect to spend around \$8,000 to \$12,000 taking classes (and the total time it will take to get certified is around 6 months) [2]. And a person can expect to make about that much in additional income over a person that does not have the MCSE certificate.

Like doctors, lawyers, and CPAs, rumblings are also being heard concerning *mandatory* software engineering personnel certification. A vote by the Texas Board of Professional Engineers on February 18, 1998 stated the Board's intention to recognize software engineering as a legitimate engineering discipline with plans to license professional engineers in that area. (A complete position statement from the Texas Board can be found at <http://www.main.org/peboard/softweng.htm>.) On June 17, 1998, the Texas Board gave unanimous approval to all proposals in this statement. In July, 1999, the Texas Board will begin licensing software engineers that can satisfy the following [1]:

1. Possession of an engineering degree, a computer science degree or some other high-level math or science degree that the Board will evaluate for adequacy
2. At least 16 years of creditable experience performing engineering work (12 years for those holding a degree approved by the Engineering Accreditation Commission of the Accreditation Board for Engineering Technology, Inc. (EAC/ABET))
3. References from at least 9 people, 5 of whom must be licensed engineers
4. Submission of documented credentials as required

After the Texas Board releases a professional software engineering exam in 1999, individuals with less experience will be allowed to apply for a Texas P.E. license by taking that exam.

3 Assessing the Software Product

Generally speaking, there are two approaches to product-based assessment of quality: white-box and black-box. White-box assessment techniques would include activities such as collecting static code metrics or measuring the degree of coverage achieved during unit testing. Black-box techniques would include reliability testing.

White-box and black-box techniques are not panaceas, however. For example, it is unclear as to what relationship a code complexity metric has to do with the reliability of the software. And reliability is based on logical correctness and the operational environment, not structural properties. Further, we cannot even exhaustively test a simple program that reads in two 32-bit integers [4].

For today's push toward COTS software, white-box certification techniques are not even applicable by COTS consumers. Naturally white-box techniques can be applied by vendors if they volunteer to. This means that those COTS software consumers that are genuinely concerned about what lurks in the software they purchase need to decompile back to source in order to apply white-box analyses (such as coverage testing or inspections).

Most licenses for COTS software deem this act as a violation of the licensing agreement. Further, pending global legislation is about to weaken the ability of consumers to get such analysis done by independent corporations or consultants. There is a global treaty up for US approval called the World Intellectual Property Organization (WIPO) Treaty. The treaty includes language that makes it illegal to reverse-engineer software to expose security vulnerabilities. The treaty will make it illegal for corporations and consulting services to conduct real-world testing of security software. Supposedly, research organizations will still be allowed to do so, however.

President Clinton has announced his intentions to sign it and it is expected to pass in the House. The US Senate has already passed this measure by a score of 99-0. This legislation is part of a global attempt to produce treaties that reduce the amount of copyright infringement on information technology. But the downside is that it disallows a consumer the right to independently certify the security of the software they purchase (without the vendor's permission).

4 Certifying Processes

Because of the limitations associated with different forms of product assessment (testing as well as techniques such as formal verification), the notion of "directly assessing software quality" became dismissed as being implausible in the mid 80s. This opened the door to ideas such as using "process maturity assessment" and other indirect approaches. The most famous process assessment model is the SEI's CMM. This model and other manufacturing-like standards rely on one premise: *good processes deliver good software*. This premise has also lead to government regulatory standards for software certification in areas of avionics, medical devices, and electric power generation.

The premise here is plausible. All that a developer need do is to score themselves using a pre-defined ranking scheme (for what is and is not good software development procedures) and then apply that score to their software. So for example, if development organization A is ranked higher than organization B, then software from A will be labeled as having more quality than software from B.

The problem is that good processes cannot 'guarantee' good software [6]. If performed properly, good processes simply increase the likelihood. When processes are not performed

properly, the likelihood is reduced. However given a fixed set of development processes, it is still possible that organization A, who improperly applies the set, produces better software than organization B, who properly applies the set. And this does not account for the issues relating to which processes are the “best.” These facts, taken together, diminish the notion that process assessment will ever be a satisfactory substitute for product assessment.

Ask yourself this: would you buy a car without test driving it? Few would, but this is precisely what we do when we employ process assessments in lieu of product assessment. Process assessments are analogous to having the car manufacturer telling you what the phases were that were undertaken during manufacturing, but that is certainly no substitute for taking a test drive.

5 Summary

The hypothesis that certified personnel equates to higher quality software is easy to disprove. The hypothesis that a more mature process equates to higher quality software can also be easily debunked. True product assessment that studies the dynamic behavior of software is clearly the best approach to certifying software quality, but problems relating to feasibility often reduce our ability to perform it with any degree of thoroughness.

The best approach will be to create a variety of different certification schemes based on: (1) the different types of exams or processes used from each of our three categories, and (2) the criticality of the software (safety-critical, games, etc.). That is, aspects of each of these three broad approaches can be combined into a single standard. For example, knowing that: (1) an organization has a certain process maturity, (2) the personnel that developed and tested the software were licensed, and (3) the software received certain forms of quality assessment should result in greater confidence in the software’s quality than if only one of these facts were known. The problem, naturally, is how to quantify subjective characteristics such as personnel accreditation. Nonetheless, it is plausible to develop different software quality certification schemes that weigh different techniques within our three approach groups appropriately with respect to the criticality of the software.

Before we end, let’s examine what role quality certification can play with respect to software *insurability*. Software insurability refers to the software-induced risk that an insurer is willing to take in exchange for an insurance premium. The insurer is not actually insuring the software, but is instead insuring the object that the software controls. But before offering insurance for that object, the insurer must understand the worst-case scenarios that can result if the software is defective.

Consider the fact that Swedish insurer Trugg-Hansa made the following exclusion effective May 1, 1998 in the general conditions of their business insurance policies:

“The policy will not cover damage, cost, legal or other liability caused directly or indirectly or connected to time-related disturbance in computer functionality.”

This demonstrates the extreme, defensive posturing being seen as a result of the Y2000 problem. But of equal significance, it opens the door for non-time-related exclusions for other anomalous software behaviors. For example, exclusions might someday read like:

“The policy will not cover damage, cost, legal or other liability caused directly or indirectly or connected to disturbances in computer functionality.”

Such a waiver enables an insurer to dump responsibility for all computer-related problems. The onus is placed on consumers to know the quality of the computer systems that they employ. The consumer now bears his or her own liability, without access to an insurer to step in as their surrogate in case of a mishap. This represents *a first* in our industry: *where insurers are so concerned about software failures that they start including exclusions in their policies*. When you couple situations such as this with the WIPO Treaty, the disregard for consumer protection that exists in the current version of the Uniform Commercial Code Article 2B [5, 3], you immediately see that the need for independent third-party certification concerning the processes, product, and personnel could not be greater.

Interestingly enough, a business has been formed to address this problem: the Software Testing Assurance Corporation (Stamford, CT). This company was founded up in 1998 to provide independent certification. Their first certification offering will assess the testing processes used on converted Y2000 software. Their current offering is effectively all process assessment with a small degree of product assessment included (See their standard at <http://www.STACorp.com/draft/standard.htm>). This independent certification is available only to corporations that seek business disruption insurance in the event that their computer systems were to fail as a result of Y2000 software problems. The founding of this organization opens up the door for additional software quality certification standards for information systems when business risks are directly tied to software quality and insurance protection is sought.

5.1 Acknowledgements

I appreciate the efforts of Don Bagert who has kept me up to date with Texas’s certification plans.

References

- [1] <http://www.main.org/peboard/softweng.htm>.

- [2] J. AYALA. Training the Microsoft Way. *Windows NT Magazine*, 4(3):122–129, March 1998.
- [3] C. KANER. Article 2B is Fundamentally Unfair to Mass-Market Software Customers, October 1997. Submitted to the American Law Institute for its Article 2B review.
- [4] J. C. HUANG. An Approach to Testing. *ACM Computing Surveys*, 17(3):113–128, September 1975.
- [5] THE AMERICAN LAW INSTITUTE AND NATIONAL CONFERENCE OF COMMISSIONERS ON UNIFORM LAWS. Uniform Commercial Code Article 2B (DRAFT), November 1997.
- [6] J. VOAS. Can Clean Pipes Produce Dirty Water? *IEEE Software*, 14(4):93–95, July 1997.