

Limited Software Warranties

Jeffrey Voas

Reliable Software Technologies

21351 Ridgeway Circle, #400

Dulles, VA 20166 USA

`jmvoas@rstcorp.com`

Abstract

Because there are different types of software (*e.g.*, language, application, target environment, etc.), different software certification methodologies are needed. Software process improvement schemes have not taken this approach and have therefore suffered in widespread adoption as a result. Their “one approach fits all” perspective is one reason why we are now seeing more customized process improvement schemes being created (*e.g.*, CMM-SSE and the Common Criteria are recent “newcomers” that address developing software with security requirements) and even the call to certify software professionals.

This paper presents a framework for customizing certification methodologies according to: (1) the specific needs of the organization requesting assurances about the software’s integrity, and (2) the peculiarities of that type of software. Each methodology must mirror the nuances of the type of software it was designed for. For example, certifying that a desktop plug-in will behave appropriately requires a different set of assessment technologies than the set of technologies needed to certify that an aircraft control system will behave appropriately. The goal of creating a certification framework is to provide a more systematic way to create and compare software certification methodologies which today does not exist. And from there, we will be able to offer limited software warranties.

1 Introduction

As software quality and information integrity become an increasing concern, the need for criteria that can accurately assess integrity and can detect defects grows. Consider the comments of the US Department of Defense’s CIO, Mr. Money, (June 17, 1999 issue of *Federal Computer Week*):

“the quality of software we are getting today is crap. Vendors are not building quality in. We are finding holes in it.”

Gary Beach, publisher of CIO Magazine, wrote in his April 1, 1999 column:

“Are you tired of software vendors sending you service packs to fix bugs that should have been stamped out earlier? Off-the-shelf commercial software isn’t good enough anymore. Service packs, indeed! Many CIOs I’ve talked to call them disservice packs.”

The National Academy of Sciences’s 1998 “Trust in Cyberspace Report” went further than these statements and attempted to find a reason for the poor quality. They stated:

“The absence of standard metrics and a recognized organization to conduct assessment of trustworthiness is an important contributing factor to the problem of imperfect information. In some industries, such as pharmaceuticals, regulatory mandate has resolved this problem by requiring the development and disclosure of information.”

which hinted that our inability to gauge software quality is the root of the problem.

While these comments reveal a heightened dissatisfaction with the *status quo* in software quality, they are not the only reason why I believe that some external force will soon be brought to bear on the software industry to improve the quality of commercial software. Legislation (*e.g.*, Y2K limited liability and the Uniform Computer Information Transactions Act (UCITA)) is also playing a role here. Without going into UCITA’s details, consider excerpts from Barbara Simons’s (President of the ACM) July 12, 1999 letter to the NCCUSL:

“The computing professional must strive to achieve quality and to be cognizant of the serious negative consequences that may result from poor quality in a system.”
The UCITA will hinder this pursuit, and result in possible harm to the general public. UCITA makes it too easy for software publishers to avoid facing any legal consequences for defective software.”

Although UCITA is still legislation and not yet law in the United States, if it becomes law, the question must be raised as to *where* will consumers turn if they are unable to accept the consequences of low quality commercial software [2]?

I believe the answer will be to organizations that provide independent assurances about software and information integrity. If true, this opens up the stage for independent, third-party software certification. The question, however, is in what form will such services be offered? And what assessment technologies should be used to judge a product’s quality? Further, one certification methodology is not going to be appropriate for all types of software because there are so many different types. Thus numerous certification methodologies will be needed and each must be designed according to: (1) what the software does, and (2) what level of integrity is guaranteed by a particular “seal of approval.”

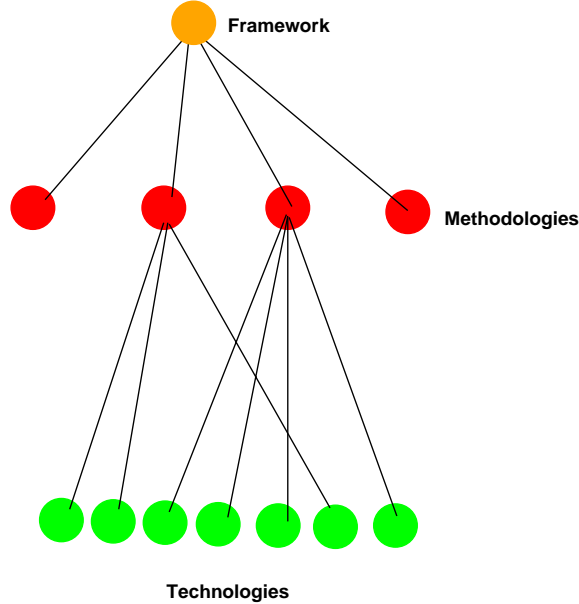


Figure 1: Basic Certification Hierarchy

2 Terminology

We begin by differentiating “product-based certification” from independent verification & validation (IV&V). Product-based certification occurs *after* the software is built in order to determine whether the software will meet its functional, performance, and quality expectations. IV&V occurs during each phase of the software development process to ensure that each intermediate work product produced during a life-cycle phase is satisfactory.

Therefore the key difference is that product-based certification begins after IV&V ends. The common feature of these two processes is that *independent* oversight (i.e., someone unrelated to the software publisher) performs the oversight. The justification for having independent overseers is to provide an additional check as to whether the current state of the software project or software product is satisfactory. I contend that the goal of product-based certification should be to warrant, to the consumer, that the software will exhibit specific behavioral characteristics.

Several other noteworthy terms that will be used throughout this paper are: framework, methodology, and technology. A *technology* is simply a software assessment technique (*e.g.*, branch coverage testing). A *methodology* is a process involving one more technologies that may be time-ordered. For example, perform technique *A* and then take the results from that and perform technique *B*. In short, a methodology is a recipe. The *framework* is an abstract model that allows us to compare and contrast methodologies. Figure 1 shows how these entities are related. The beauty of having a framework is that it provides a means for developing new, customized certification methodologies for different types of software applications.

3 Process, Personnel, and Product

The concept of software certification is not new. The main reason why it has not become a widely used process has been *ad hoc* attempts to formalize certification schemes that were not theoretically sound nor practical. Because of this, there has been as much distrust in certification as there has been in software.

One reason for this is that the most widely applied approach for determining whether software will be of “good quality” is *process* maturity assessment of an organization [3]. Process maturity assessment is based on the hypothesis that organizations with better organized teams and more advanced software engineering tools and techniques will produce higher quality products.¹ While I do not dispute this, it is not always true; it is a generalization.

The second (and less frequently used) approach is *personnel* accreditation. The hypothesis is that better trained employees produce better quality software. For example, the American Society for Quality (ASQ) defines certification as:

“formal recognition that an individual has demonstrated a proficiency within and a comprehension of a specified body of knowledge at a point in time. It is peer recognition and not registration or licensure.”

The IEEE is also seeking a scheme for accrediting software engineering professionals:

“The IEEE Computer Society is developing a program to certify software engineering professionals. This program will provide formal recognition of professionals who have successfully achieved a level of proficiency commonly accepted and valued by the industry.”

Once again, although I support the generalization that better people produce better software, it is not possible to tie this hypothesis to a specific piece of software. One small bug from the brightest individual can still result in a dangerous software product.

The third approach and the focus of this paper is *product* certification. This approach assesses the software itself to determine its quality. Here, rigorous, repeatable, and reproducible assessment technologies are needed in order to provide trust of the product. If less rigorous assessment technologies or technologies whose results are ambiguous (*e.g.*, static metrics) are employed, we quickly return to the quagmire created by process and personnel certification: not knowing how good or bad the behavior of the software will be.

The main problem with the first two approaches is their lack of *measurable, objective* data. It is vital that all data collected be correlateable to the quality of the software. Further, the results of these approaches may not be repeatable or reproducible. A *repeatable* approach is one that will produce the same result over and over if the same certifier re-applies the

¹An interesting anecdote demonstrating this problem occurred when a college student and professor provided a product that was significantly more reliable (using formal methods) than the same product produced by an SEI-CMM Level 4 organization (however neither product met pre-defined reliability requirements) [4].

methodology. A *reproducible* approach is one that will produce the same result even if a different certifier is used.

As an analogy, consider a dog show. The dog that wins the top award, “best in show”, will more likely win because of who the judge is (and what types of dogs he or she prefers) and less of whether that dog is really the best. If the judge were different, the winner would likely be different. Such a situation clearly undermines the fairness of the outcome.

The third approach can provide measurable, objective data. But its current problem is practical: a lack of automated technologies and methodologies that *fairly* determine whether a software package will behave as desired. Automated technologies are the only way to ensure repeatability and reproducibility in the certification process. This problem is simply “supply vs. demand”: when the demand for rigorous certification technologies is great enough, automated technologies will follow.

Note, however, that the key here is that the automated technologies are “rigorous,” meaning that they must be tough and thorough benchmarks. Organizations such as Underwriter’s Laboratory, who have established a credible trackrecord in certifying physical products for over 100 years, still today have serious problems setting optimum safety standards for such products [1]. And unfortunately, software is even harder to certify to any degree of confidence than physical products are.

4 The Framework

Our *product-based* software certification framework has eight dimensions. The goal is to create a set with one member from each dimension. The set will represent a particular type (or what we sometimes refer to as “species”) of software. A certification methodology can then be built for that species.

The eight dimensions are:

1. **Safety, Security, Reliability, Availability, Fault tolerance, Performance** Software “ilities” represent desirable software characteristics. Here we have listed six. More can be added. These are the major behaviors that we want software packages to exhibit.
2. **Hard Real-time, Soft Real-time, Time indifferent** For some software packages, the ability to compute within time constraints is vital.
3. **Embedded, Desktop, Mainframe, Internet** Platform is highly important because as we all know, when the environment changes, so may the behavior of the software.
4. **“In a vacuum”, “With respect to the environment”** Software can be certified for generic use or with respect to a certain environment.
5. **Avionics, Medical, Finance, Telecom, Automotive, Nuclear, Maritime** It is also vital to consider the industry stake-holders and what they need guarantees of when building a certification methodology

6. **Executable format, Source code format** Whether source code is available for consideration during certification will affect which software assessment technologies are applicable.
7. **Custom software, COTS** Whether the software was written from scratch or is intended for the mass market will govern how the software is certified.
8. **Previous version certified, Never certified** Whether the software is an upgrade or a totally new product will determine how much assessment of the product is needed.

This framework results in 8064 permutations. For example, (safety, hard real-time, embedded, “with respect to the environment”, automotive, executable, COTS, and never certified) is one permutation. Note, however, that not all permutations make sense. Also note that by adding more possibilities to each dimension, the number of possible permutations increases.

This framework is analogous to a restaurant buffet. Buffets offer a variety of meats, vegetables, breads, desserts, etc. For people with specific dietary restrictions, certain food groups are off-limits while others are acceptable. This certification framework is no different. Each member of a set suggests what technologies will be needed to certify software defined by that set.

I contend that it is absolutely necessary to have certification methodologies that match the peculiarities of different software species. Precedence for this exists in other industries. For instance, consider drug-interaction warnings provided by pharmaceutical companies on the *fact sheets* that accompany medications. Their warnings list all known pre-existing health conditions that preclude patients from taking a certain drug. The warnings explicitly mention side effects and often mention results from laboratory studies (including those using animals). Unfortunately, the software industry provides nothing similar.

5 Software Warranties

Once we have a certification methodology for a particular type of software, instantiations of that type can be warrantied. Product liability law in the United States is based on two types of warranties, namely (1) express and (2) implied. This classification scheme is beginning to be discussed by the legal community for application to the US software industry. The recently adopted draft of the Uniform Commercial Information Transactions Act states that a licensor of software(or his employees) can create an express warranty in several ways:

“(1) An affirmation of fact or promise made by the licensor to its licensee in any manner, including in a medium for communication to the public such as advertising, which relates to the information and becomes part of the basis of the bargain creates an express warranty that the information to be furnished under the agreement must conform to the affirmation or promise. (2) Any description of the information which is made part of the basis of the bargain creates an express

warranty that the information must conform to the description. (3) Any sample, model, or demonstration of a final product which is made part of the basis of the bargain creates an express warranty that the performance of the information must reasonably conform to the performance of the sample, model, or demonstration, taking into account such differences as would appear to a reasonable person in the position of the licensee between the sample, model, or demonstration and the information as it will be used.”

Implied warranties are distinct from express warranties. They

“rest on inferences from a common factual situation or set of conditions so that no particular language is necessary to create them.”

Implied warranties operate differently from express warranties because an implied warranty exists unless the merchant or licensor affirmatively disclaims or modifies the warranty.

As an example, consider a word processing product and consider the “file open” and “save file” features of that product. It is assumed, when you buy such a product, that these features will exist in the product. No express warranty is needed stating the product has these features. However if you were to purchase a product and it did not have these features, you’d clearly be entitled to a refund even if every other feature of the product were completely reliable and there was an express warranty stating so (except in the case where an express warranty was included stating these two features were not included).

The *ultimate* goal of software certification should be to create express warranties defining levels of quality that the user can expect. Such a warranty could read as follows:

When feature Z of version Y of product X is used on machine M configured in manner C , the product will fail approximately one time per thousand executions of Z given that only even integers are fed into Z .

However until that day arrives, simply have a fact sheet on a product would also be beneficial. For example, having a statement such as the following on a product fact sheet, although not a warranty, still provides insight to a potential user as to the quality of service that they can expect and under what assumptions they can expect it:

For version Y of product X , when feature Z was tested one million times using even integers on a machine M configured in manner C , X crashed 1000 times.

The idea here is straightforward: if someone else plans to use Z (of X and Y) in a similar manner (i.e., even integers) on the same type of a machine that has the same configuration, then they should expect a similar percentage of failures. Clearly, detailed information must be provided about C and M in order to make such a quality prediction understandable to users. However there is no practical or theoretical reason as to why the software industry does not develop such fact sheets with such details other than the fact that publishers have a long history of trying to avoid all responsibility for defective products.

6 Summary

It is prudent to certify specific software applications according to their platforms, domains, environments, profiles, “ilities”, etc. Once done, software certificates and fact sheets can be published that are unambiguous and limited in scope.

At this point, attempts to provide broader warranties should be viewed suspiciously because of our immature ability to precisely measure software quality. We lack fine-grained, automated certification tools whose results are repeatable and reproducible.

Notwithstanding, the benefit of certifying software for specific environments is beginning to be noticed. Linux recently decided to protect their reputation from problems created by third-party hardware and software:

“SAN FRANCISCO, MAY 19, 1999 Linuxcare, Inc. (www.linuxcare.com), the leader in technical support, consulting, education, and product certification for Linux, announced today the launch of Linuxcare Labs, a comprehensive program for independently certifying hardware and software for use with the Linux operating system.”

And finally, I recognize that the first software warranties and fact sheets may be complicated and difficult to understand. Because of this, their adoption by the software community at-large will likely be slow. But unfortunately, consumers cannot have it both ways. They cannot believe certificates containing a short little phrase like “everything in this product works perfectly.” Warranties and fact sheets detailing numerous product limitations are the only guarantees that can be justified. And if over time a particular software package receives a broad range of limited warranties, these warranties can easily be merged into a single certificate that is broader in scope.

References

- [1] C. E. MAYER. UL: Still Safety’s Symbol? Underwriters Laboratories Draws Fire on Product Tests, *Washington Post*, Wednesday, November 24, 1999, page A01.
- [2] C. VOSSLER AND J. VOAS. *Advances in Computers*, chapter Defective Software: An Overview of Legal Remedies and Technical Measures Available to Consumers. Academic Press, to appear in 2000.
- [3] J. VOAS. The Software Quality Certification Triangle. *Crosstalk*, 11(11):12–14, November 1998.
- [4] J. WIDMAIER. Building More Reliable Software: Traditional Software Engineering or Formal Methods? In *Proceedings of ISSRE’99 Industry Practices and Fast Abstracts*, pages 253–260, November 1999.