Developing a Usage-Based Software Certification Process

Jeffrey Voas
Reliable Software Technologies

The author proposes an alternative to process-based methods of assuring software quality: a certification process that will provide product-based and trustworthy quality guarantees for commercial software packages.

The methods for certifying software quality continue to multiply. Popular, process-based approaches—such as ISO 9000 and SEI-CMM—make software publishers take oaths concerning which development standards and processes they will use. These approaches often require auditors to spot-check a publisher's documentation and oaths.[1] Yet even if a certification auditor can verify the publisher's veracity, that verification alone does not guarantee high-quality software.

Given these problems, I propose a certification methodology that does not employ publisher oaths and auditors. I believe that completely independent product certification offers the only approach consumers can trust. Publishers should be amenable to independent agencies taking on this role because it allows the publishers to avoid warranting their own software.

By hiring a third party to grant software certificates that contain information that could be used to support the creation of express warranties concerning quality, publishers shift this responsibility onto someone else, much like a doctor who orders a second opinion or another test. End users will also benefit from independent agencies that offer unbiased assessments. The business case for creating independent agencies to certify software quality is therefore strong.

I call the agencies that perform such certification Software Certification Laboratories. The beauty of establishing independent SCLs is that they will provide a fair playing field for each publisher, assuming that each product under review receives equal treatment.

I suspect that certifier liability is the key reason SCLs have not become widespread already. When certified software fails in the field, the certifier bears some level of liability. Courts in the US are notorious for holding to unusually high standards those persons and organizations that represent themselves as professionals—such as doctors, lawyers, and engineers.

SCLs bear a similar liability with similar consequences for miscertification.[2] To reduce this liability, we must employ accurate methods for making certification decisions, and ideally, those methods will be automated. Unfortunately, even if an SCL employs the best static analysis and dynamic testing techniques, it may still fail to consider the actual stresses that software will experience in users' hands. Thus, SCLs suffer from the problem of accurately determining how well behaved a soft-

0018-9162/00/$10.00 © 2000 IEEE

ware system will be—the key piece of information we need from certifiers.

The certification process I describe employs automated processes to greatly reduce this liability while also eliminating the need to dispatch human auditors. The process harnesses the testing resources of end users, drawing upon proven methods such as that which made Linux the most popular and reliable of all Unix flavors.[3,4] Totally product-based, my proposed process assesses how well behaved the software is, not the maturity of the processes used to develop the code.

LIMITED SOFTWARE WARRANTY MODEL

My company, Reliable Software Technologies, seeks first to certify mass-marketed software applications and components that have many potential uses. In the short term, we plan to certify applications such as Microsoft Word, PowerPoint, and Excel. Long term, we hope to certify smaller components that could be embedded in a variety of applications. Such a certification process will, we hope, foster greater software reuse for different software types.

Component-based software engineering (CBSE) simply involves building software systems from software parts. Yet the virtues of CBSE strategies have been touted for years, with little evidence that component-based development is close to becoming the de facto standard for building software systems. Why the delay, when glue technologies like CORBA, ActiveX, COM, and DCOM for component interoperability—along with reusable component libraries—already exist?

I suspect that CBSE has not become ubiquitous because of widespread distrust of software components and a lack of knowledge about how to design for reuse. That any two components can be glued together rapidly is not sufficient grounds for popularizing CBSE. Integrators must be confident that the right components were glued together and correctly, and that the glued components are dependable.

Further, trust in a component's dependability must come from someone other than the software publisher. SCLs that might hope to employ in-house testing to certify software will suffer the same problems software publishers face: inadequate levels of testing and incorrect assumptions about usage. Indeed, the degree of testing an SCL can perform cost-effectively may be less than that already done in-house by the software publisher. Thus, it's dubious to think that SCLs can grant certificates based solely on in-house testing.

Software warranties must be the end result of massive amounts of operational usage. Such testing will have demonstrated product stability in fixed environments and fixed marketplace sectors, such as the embedded, desktop, and Web-based markets. Without this feature, these limited warranties will either be too restrictive or lack credibility.

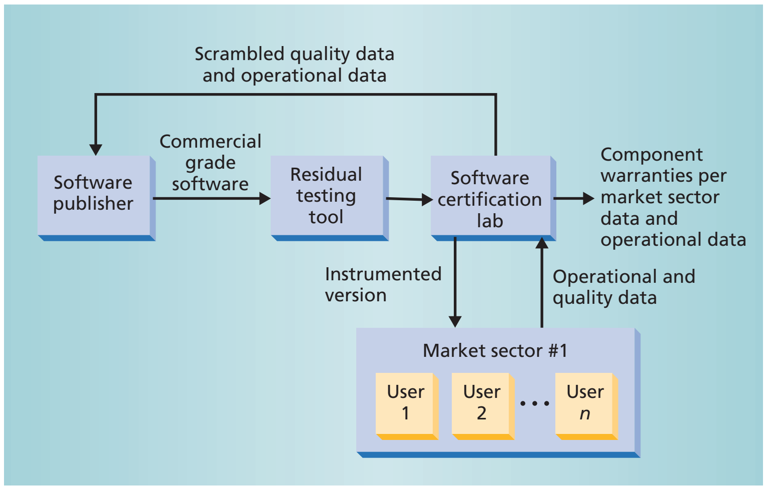
Scrambled quality data and operational data
Software publisher
Commercial grade software
Residual testing tool
Software certification lab
Component warranties per market sector data and operational data
Instrumented version
Operational and quality data
Market sector #1
User 1
User 2
...
User n

Figure 1. The basic certification process. A software publisher provides a commercial-grade software candidate to a software certification lab, which embeds a residual testing tool in the product to capture operational performance data. The results, coupled with specific market-sector requirements, determine the type and extent of certification granted.

USER-BASED SOFTWARE CERTIFICATION

If SCLs and publishers cannot perform adequate product testing, who can? I assert that the disorganized body of software testers known as *the users* can be unified to overcome this problem—if we do so in a way advantageous to them. The issue then becomes how to best tap their expertise to make software warranties a reality.

User-based product certification provides the best means for granting a software warranty because the process evaluates the product's performance as customers use it under real-life conditions. Our approach collects valuable field data from the user's environment as nonintrusively as possible. Currently, organizations rarely collect such data and therefore it is lost.

We will exploit testing technologies similar to the *residual testing* technologies described by Christina Pavlopoulou,[5] which continued to test fielded software for structural coverage as they operated via instrumentation embedded in operational software. This information can reveal which software code is executed, whether assertions fail, and whether the software itself fails.

In our approach, we will also collect information on how the product is used. We can measure the frequency with which certain features are called and simply collect files of inputs that we use to build operational profiles. These information sets will serve as the basis for issuing limited software warranties.

Figure 1 shows our proposed certification process. Here, a software publisher subjects the finished product to a residual testing tool that creates one instrumented copy. Nonidentical copies could also be built, but doing so will make the SCL's tasks more complicated. The publisher gives the copies to the SCL, who provides the instrumented versions to prequalified users from different marketplace sectors. These users will use the product in a manner consistent with how the SCL wants them to use it. Microsoft's beta-user program provides an excellent example of how to restrict possible users to only those who will actually provide the greatest amount of information.

Note that it would be possible, and potentially preferable, for the SCL to embed the instrumentation and not have the publisher do so. This could be accomplished by simply allowing the SCL to have access to certain source code files or interfaces. After all, the reason for using a third-party independent organization is to reduce, as much as possible, any role that the publisher might have in spying on users or impacting the outcome of the certification process.

Periodically, the SCL gathers the information from the user sites, then merges it and computes the statistics. Doing so lets the SCL provide publishers with statistical data on how their product was used and how it behaved in the field. The SCL must generate statistics from the participants in a way that makes a backward trace to any specific user highly improbable.

An SCL seeks to determine whether a product should be warranted and what it should be warranted for: which platforms, market sectors, and environments. As it gathers additional data over time, the SCL may be in a position to broaden a warranty and make it less restrictive.

Safety-critical systems present a special challenge. I do not suggest that SCLs seek volunteers to fly uncertified software-controlled aircraft or use uncertified software-controlled medical devices to determine if the software should be certified. Instead, I propose that SCLs should certify software first for noncritical market sectors, before it is used in critical sectors. After achieving noncritical certification, the product could be employed with confidence by vendors of safety-critical applications *if* its utility was previously certified for a noncritical environment that mirrors the safety-critical application.

If the software behaves well in this and other safety-critical applications, and if the SCL can collect enough evidence to demonstrate confidence in the software's utility, then it can build a case that warrants certifying the product's utility for the safety-critical market sec-

Key Characteristics of the User Participation Model

So that users, software certification labs, and publishers will accept our model, we deliberately built in the following characteristics:

- Certification is performed on a stable, nonbeta version of the product.
- The software publisher does not perform the certification. Rather, the publisher delivers a fixed number of product versions to the SCL. These versions have residual testing capabilities built in. The SCL licenses the versions to prequalified users from various application areas such as the safety-critical, desktop, and Web domains.
- The SCL determines the testing methods and usage type information collected. This information should be encrypted during transfer back to the SCL. To diminish concerns regarding user privacy, details concerning the type of information being collected must be made available to participating users.
- SCLs base their certification decisions on significant operational experience created by the user base. This approach employs orders of magnitude more testing than the publisher or SCL could perform. Also, certification decisions are based on real operational scenarios—not hypothesized ones, as is frequently the case now.
- Background processes that minimally interfere with users collect the user data. Users expend no resources to trap and collect the information, other than the computer resources necessary to gather and store it. Although they don't need to perform manual tasks to collect the information, their versions will suffer performance degradation due to the background processing cycles required to perform the postdeployment testing. Users who participate in this accelerated testing program will be compensated with reduced-cost or free software.
- The SCL is the exclusive recipient of all user-generated testing data. The raw information does not go to the publisher. SCLs agree to legal confidentiality between themselves and users. Once the SCL scrubs the raw information and ensures that user identities cannot be traced from the composite information, it passes the data back to the publishers. This process allows publishers to improve the quality of future upgrades and focus in-house testing toward their user base.
- A software warranty is limited to different market sectors and operating environments. When testing collects enough data from the field to affirm that a component works properly in a particular market sector, the SCL will provide software warranties specific to that sector. For example, an SCL warranty might read as "Software product X is warranted to perform with a reliability of 99.9 in the Windows-NT desktop environment." What constitutes "enough data" is a function of basic statistics and the type of certification sought. For example, to be 99 percent confident that a system's true

tor. Clearly, caution must be taken here, but recall that we are dealing with postbeta software to begin with. We are never dealing with untested software, even when a piece of software is first embedded into a safety-critical environment.

BENEFITS

As outlined in the "Key Characteristics of the User Participation Model" sidebar, our approach offers several benefits. These benefits cannot be realized, however, until we know precisely what and how much data is sufficient to justify issuing limited software warranties. Ultimately, the amount of data needed will depend on how strong and broad a warranty we seek.

We envision this data to include results from reliability assessments, assertions, exception calls monitoring, operational usage, and code coverage analysis. Although other analyses are possible, these seem to be the more important indicators of whether a component has been exercised thoroughly enough to warrant its quality for specific environments and market sectors. In future research on this topic, we plan to work with technology insurers to select the residual testing technologies that provide data sufficient for offering software insurance premiums and software warranties.

How will we know that the tool adds sufficient instrumentation if the instrumentation is done by the publisher? How do we certify that the tool is adding the correct instrumentation? Answering these questions is no different nor more difficult than how we qualify other software tools in various regulated domains, like commercial avionics. For example, the FAA has an official procedure for software tool qualification. We can apply a similar scheme here to ensure that the tool provides correct instrumentation. Further, the SCL will be responsible for ensuring that the tool's user—the publisher—adds adequate instrumentation. To gain quick approval, unscrupulous software publishers might attempt to force the tool to add bogus instrumentation that fails to perform testing. Again, the SCL must mitigate this risk.

RELATED APPROACHES

Although unique, the model I propose for combining SCLs with residual testing technologies builds upon previously employed methods and techniques. Specifically, it bears a strong conceptual similarity to

probability of failure is less than 0.00001, you need 460,000 test cases.

- Users consent to participate. Those who do not wish to participate in this process do not have to. They are free to license nonbeta versions that do not undergo residual testing.
- Publishers compensate SCLs for their services. The SCL will also own the collected data. Because the data has economic value, the fees that SCLs place on publishers should be reasonable. Note, however, that publishers are gaining both invaluable data on how their software is being used and enormous amounts of product testing. Therefore, it is reasonable that they pay a fair price.
- No auditors are needed. Most existing certification models require that trained auditors visit publisher sites and sift through requirements documents and test plans. Our scheme avoids the pitfalls associated with human auditor error.
- Unlike ISO 9000 and the CMM, our product-based certification allows innovative processes to emerge. If a developer organization can produce certifiably good products that conform to an in-house standard, it has developed new processes that other development organizations should consider. This open approach avoids the risk that ISO and CMM may lock us into old process ideas prematurely.
- Our certification process creates a framework for performing accelerated testing, which requires the distributed environment our model defines. The highest reliability goals, as defined by Rick Butler and George Finelli,[1] frequently require such testing. The process can also benefit reliability models such as Squeeze Play that Keith Miller and I developed, which often requires massive amounts of testing to overcome low testability scores.[2]
- Our process offers publishers the benefit of fewer point releases. Once their product has undergone certification and succeeded, publishers should need to release fewer bug-fixing versions.
- If software packages can be confidently warranted for certain environments and under specific assumptions, users with equivalent environments no longer need to invest heavily in retesting the licensed software.
- Because these methods are automated, this certification approach is repeatable and reproducible, which ensures that different SCLs will produce the same warranty because they will receive the same field data.
- This process forces publishers to define their software's correct behavior. This feature alone could result in higher-quality software products.
- This model has the potential to reduce bloatware by allowing publishers unprecedented information concerning what features are used and which ones are avoided.

References

1. R.W. Butler and G.B. Finelli, "The Infeasibility of Experimental Quantification of Life-Critical Software Reliability," *Proc. Conf. Software for Critical Systems*, ACM Press, New York, 1991, pp. 66-76.
2. J. Voas and K. Miller, "Software Testability: The New Verification," *IEEE Software*, May-June 1995, pp. 17-28.

Our approach dissolves the distrust associated with schemes that use auditors or that measure process maturity.

Netscape 4.5's Quality Feedback Agent and to Pure Software's PureVision. More generally, several prominent trends in software development point toward the approach I recommend.

Netscape

Netscape 4.5 contains an option called The Netscape Quality Feedback Agent, which sends feedback to Netscape's developers. The agent is activated when Communicator encounters some type of runtime problem. When activated, the agent collects relevant technical data and displays a form in which a user can type comments. Netscape uses this data to debug known problems and identify new ones. Unfortunately, most users do not activate this feature.

PureVision

Pure Software's PureVision performed crude residual testing functions. It worked similar to the way our approach defines residual testing: A publisher produced self-monitoring copies of its product for installation at user sites.[6] The installed copies sent back a report to the publisher that included the version number, execution start and stop times, features used, amount of memory used at exit, and each user's ID. If the product failed, the software added exit codes and a stack dump to the report.

Pure knew that users would be wary of publishers looking over their shoulders. Thus, they included an option whereby a user could inspect a report before PureVision sent it to Pure, as well as an option to cancel the send. According to former Pure employees, PureVision went out of business because users proved unwilling to provide detailed, nontechnical information such as who used the software, when, and on which host. In contrast, the information we will collect is more technical: Our approach spies on the software and its correct behavior rather than on the user. Further, we employ an SCL to keep participant users and publishers away from each other, which is something the PureVision model did not do.

Software Testing Assurance Corporation

Software Testing Assurance Corporation, a for-profit venture started in 1997, attempted to be the sole certifier for Y2K insurance. It based its certification procedure on a public-domain testing standard for Y2K-remediated code and employed STAC-approved auditors to make site visits to companies that sought Y2K software insurance. In the end, however, these insurance policies were never offered.

Microsoft

A long-time champion of using beta testing to collect information on how their products perform, Microsoft employs users to test each product before its commercial release—and compensates them for doing so. Prequalified users receive advance copies of a product at reduced rates in exchange for feedback concerning product stability, usability, and reliability. Microsoft uses this information to decide when a product is ready for general release. This practice demonstrates users' willingness to participate in testing if they're rewarded with discounted software. Also, Michael Cusamano and Richard Selby claim that Microsoft gathers detailed user profiles from specifically instrumented versions of their software.[7]

Linux

Finally, consider Linux, a Unix operating system project that began with a handful of enthusiasts in 1991 and today has nearly 8 million users. The product of hundreds of user-developers, all of whom donated their time to write the system, Linux is considered the most reliable of all Unix operating systems.[4] In fact, Linux's success is often used to support the assertion that only open-source products are truly trustworthy.

Lessons learned

What can we surmise from these anecdotes? Microsoft and Linux show that users will participate in efforts that result in improved software; PureVision and Netscape demonstrate that publishers have a serious desire to attain field information; STAC reveals industry's interest in forming for-profit software certification corporations.

Will users trust an SCL to act as an intermediary between themselves and a software publisher? As long as the confidentiality agreements between SCL and publisher, and between SCL and user, are binding, I believe they will.

Demand for SCL services translates into business opportunities for users who can overcome the liability risks—which is why several SCLs already exist. For example, KeyLabs certifies language purity for Java, but makes no promise about software quality. Because language purity is trivial to test for, KeyLabs exposes itself to minimal liability.

Our approach dissolves the distrust associated with schemes that use auditors or that measure process maturity. We capture user information that is traditionally discarded: whether the software behaves correctly and how it's used in the field. Our approach does not invade user privacy; participant users agree to license instrumented versions from intermediaries between them and the publishers. Further, participant users are informed as to what information is collected.

We envision that our proposed approach will initially be applied to complete applications. Once it has been applied at that level of granularity and we know better how to implement this scheme, it should be ready for smaller-sized objects like plug-ins and components. This approach clearly has the potential to decrease the cost of software development by fostering CBSE.

Finally, attracting more participant users and only releasing instrumented versions to persons committed to seriously stressing a product can reduce the time it takes to attain a limited software warranty. Doing so will reduce the time all users must wait to enjoy uninstrumented, warranted software. ❖

Acknowledgments

I thank the anonymous referees for their comments on this article. I also thank Chuck Howell, John McManus, Roger Alexander, Dolores Wallace, Frank Ackerman, Lora Kassab, and Keith Miller for their comments on earlier versions of this article.

References

1. J. Voas, "The Software Quality Certification Triangle," *Crosstalk*, Nov. 1998, pp. 12-14.
2. J. Voas, "Software Certification Laboratories: To Be or Not to Be Liable?" *Crosstalk*, Apr. 1998, pp. 21-23.
3. B.P. Miller et al., *Fuzz Revisited: A Re-Examination of the Reliability of UNIX Utilities and Services*, tech. report, Computer Sciences Dept., University of Wisconsin, Madison, Wisc., 1995.
4. B.P. Miller, L. Fredriksen, and B. So, "An Empirical Study of the Reliability of Unix Utilities," *Comm. ACM*, Dec. 1990, pp. 32-44.
5. C. Pavlopoulou, *Residual Coverage Monitoring of Java Programs*, master's thesis, Purdue University, West Lafayette, Ind., 1997.
6. L. Bingley, *PureVision: Shedding Light on Black Art Betas*, APT Data Services, New York, June 1995, p. 17.
7. M. Cusamano and R. Selby, *Microsoft Secrets*, Simon & Schuster, New York, 1998.

***Jeffrey Voas** is a cofounder of Reliable Software Technologies. His research interests include software dependability assessment, software certification, and software testing. He received a PhD in computer science from The College of William and Mary. Contact him at jmvoas@rstcorp.com.*