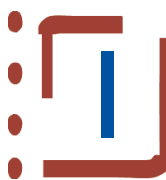


Software development is increasingly an “acquire and glue” process. How do you know when you can trust a COTS component to do what you expect it to in your system? Software certification is one viable answer. If our industry doesn't act soon to police itself, governments might step in to fill that void.

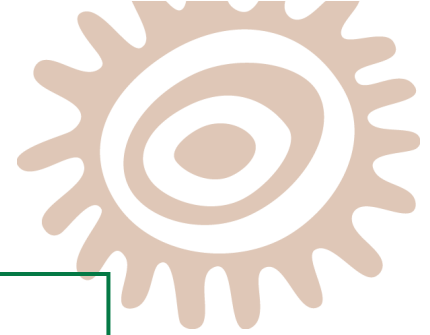
Certification: Reducing the Hidden Costs of Poor Quality

Jeffrey Voas, Reliable Software Technologies

 In 1997, an estimated 25.5 percent of a typical corporation's software portfolio was commercial off-the-shelf software.¹ Forecasts had that figure rising in 1998 to around 28.5 percent and to exceed 40 percent in the next four years.

Today, commercial software comes packaged in various levels of granularity, from complete applications to component libraries. Access to so much functionality is changing the software landscape. For example, access to COTS utilities decreases the need for developers to rewrite functionality that they can license, but it increases the need for integration and testing resources. Also, we cannot ignore the possibility that bad COTS components will be embedded in numerous applications, causing widespread damage.

Regardless, access to commercial software means industry can field more complex systems quicker and more cheaply. Sadly, these savings often ignore the substantial long-term impact of acquired software during system maintenance.



Thus software development is slowly shifting away from producing “one of a kind” systems to an “acquire and glue” process, in which developers create systems from reused subsystems. This fact alone is not alarming until it is coupled with software’s endemic and perennial quality problems—in systems as well as in subsystems.

Such problems become magnified for heavily COTS-dependent systems, since the licensee of the acquired subsystem code cannot apply most quality improvement techniques. Of course, the software publisher can (and should) apply these techniques, but a licensee cannot easily verify whether the publisher has opted to do so. This can cause serious short- and long-term problems.

ENTER CERTIFICATION

This is where independent software certification comes in.

Recognize that the notion of building software from pieces of code is not new. For example, using a language-provided construct such as ‘+’ is an example of employing COTS software. A function such as `cosine` does as well, and languages such as Fortran have provided these capabilities for years. So is the concern over commercially acquired software well-founded?

I believe it is. The fundamental difference between cosine utilities and commercial components and libraries is the level of granularity. When programming only involved using lower-level language-defined functions, code-level defects were likely. Component-based development moves us up a level to where we write more glue software and less component code, so we introduce fewer coding errors. However, we are gluing someone else’s components that potentially have numerous defects, unknown side effects, and other behavioral anomalies, and we cannot debug the acquired components as we could if we had created the system from scratch. Hence, the finished product might well have as many defects as if we had developed the entire product ourselves.

Our fears concerning acquired software could be reduced if we knew specifics about the software: For example, how was it developed and who developed it? Such software would engender more confidence if someone other than the developer would address such questions and certify, for example, that

- ◆ the code has no embedded malicious behav-

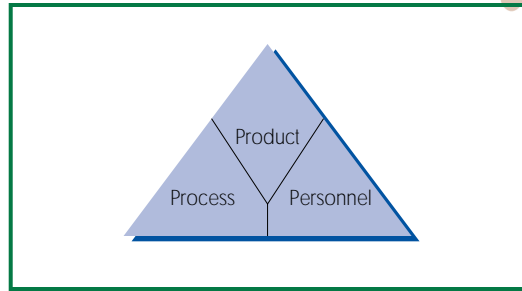


Figure 1. The software quality certification triangle. We can approach certification from any one of these aspects, but a combination of all three will provide the best balance.

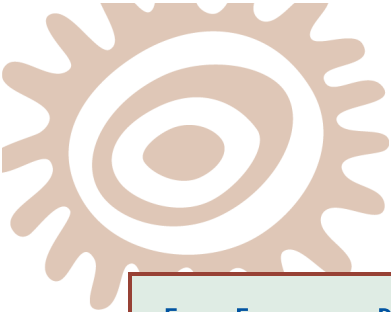
iors (Trojan Horses, Easter Eggs, logic bombs, viruses, trap doors);

- ◆ state-of-the-practice development and testing processes were applied properly; and
- ◆ the developers have passed professional exams demonstrating programming proficiency.

This suggests, then, that a reasonable starting point for software certification might be one or more of three approaches: accrediting developers for demonstrating specific skill sets, assessing the code’s behavior, and certifying that processes are properly followed. In the past, I have referred to this notion of three distinct approaches to certification as the software quality certification triangle (see Figure 1).² Although you can approach software certification in any one of these ways (the third, product certification, most strongly correlates with the “goodness” of the software), a combination of the three provides a balanced approach. This is preferable, since any one of the three can be inadvertently misapplied.

As I stated in my May/June 1999 Quality Time column, I fear that the status quo in software quality is moving us closer to the day when governments will regulate the software and information technology industries. Ideally, we can avoid this nightmare. However, until the software industry does a better job of self-policing (as opposed to hiding behind shrinkwrapped warranties), that day gets closer. To keep this from happening, we need software certification schemes that let us justifiably trust software. If we don’t, the government will take the first widespread crisis as justification for stepping in.

Perhaps the software industry is not as worried about government intervention as I am, but it may finally be taking certification seriously. The most recent evidence I have seen comes from Linuxcare



FOR FURTHER READING

Software Evaluation for Certification: Principles, Practice, and Legal Liability, A. Rae, P. Robert, and Hans-Ludwig Hausen, McGraw-Hill, London, 1995. Here is an all-around good book on the topic, and the only one I know of exclusively devoted to certification.

Proceedings of the 1999 International Conference on Software Assurance Certification, RST Corp., Dulles, Va., 1999. This contains about 25 papers representing state-of-the-art thinking on how to certify software.

Computer, June 1998. Here are several good articles on how to know if COTS software is good or not.

The Use of Computers in Safety-Critical Applications, HSE Books, Suffolk, UK, 1998, <http://www.open.gov.uk/hse/hsehome.htm>. This book discusses how certification is done in the UK for safety-critical software-based systems.

The Procurement of Safety Critical Software in Defence Equipment, UK Ministry of Defence, Def-Stan 00-55, Issue 2, London, Aug. 1997.

SPICE: The Theory and Practice of Software Process Improvement and Capability Determination, Khaled El Emam, Jean-Normand Drouin, and Walcelio Melo, IEEE Computer Society, Los Alamitos, Calif., 1998. This is a good book on the SPICE process for certifying software development processes.

The Art of Software Testing, Glenford Myers, John Wiley & Sons, New York, 1979.

The following companies are doing process, personnel, and product certification:

- ◆ DNV (www.dnv.com/certification/default.htm)
- ◆ International Computer Security Association (www.icsa.net)
- ◆ KeyLabs (www.keylabs.com)
- ◆ Linuxcare (www.linuxcare.com)
- ◆ TUV (www.tuvps.com/industry/techequip.htm)
- ◆ Verisign (www.verisign.com)

(www.linuxcare.com), a leader in technical support, consulting, education, and product certification for the Linux operating system. As it announced on 19 May 1999, Linuxcare is launching Linuxcare Labs, a comprehensive program for independently certifying hardware and software that is intended for use with the Linux operating system. As part of the launch, Linuxcare Labs released test results for certified hardware components from Dell, Compaq, and PFU. These are the first products to receive the "Linuxcare Labs Certified" mark, a distinction that is intended to instill consumer confidence in offerings from Linux vendors and OEMs. This hints that the software industry might finally be realizing that interoperability is a serious concern that specialized certification techniques can address.

THE FOCUS ARTICLES

This focus section presents four articles.

In "Using Immersive Virtual Environments for Certification," Carolina Cruz-Neira and Robyn Lutz define immersive virtual environments, then go on to argue that VE technology is now mature enough for use in software certification. In particular, they discuss how VEs currently help in designing and evaluating safety-critical systems such as flight simulators, complex machinery training, and emergency rescue strategies, all of which require human operators.

In "Software Safety Certification: A Multidomain Problem," Patricia Rodriguez-Dapena discusses how software-intensive, safety-critical, multidomain applications—for example, space systems used for aircraft guidance and navigation—can be problematic for certifiers. The article deals with these concerns:

- ◆ Which organizations can legally support multidomain and international system safety certifications?
 - ◆ How do system safety requirements translate to software requirements?
 - ◆ What are the derived design and implementation constraints for software development?
 - ◆ What software verification methods and techniques can help demonstrate that a system is safe?
- In addition, the article mentions several international initiatives intended to improve the situation.

Our third article, "Toward Credible IT Testing and Certification" by Shukri A. Wakid, D. Richard Kuhn, and Dolores R. Wallace, discusses the freely available test methods being created by the Information Technology Laboratory of the US National Institute of Standards and Technology for the commercial sector. ITL's approach adapts the principles of measurement science to measuring a software product's conformance to particular standards as well as to measuring the software's performance and dependability. ITL plans to provide its test suites to accredited test laboratories, certified by either the NIST-administered National Voluntary Laboratory Accreditation Program or by the private sector.

In "Certifying Software for High-Assurance Environments," I propose a framework for certifying code that is destined for systems with high-assurance requirements. The methodology deliberately ignores how the software was developed. Thus the framework for certifying code is product oriented, not process oriented or personnel process oriented. The methodology stresses the benefit of testing sys-



tems on off-nominal cases and at the extremes; because these cases are often ignored during operational testing, they are also the events that cause systems to be less robust. Therefore, the certification process must consider how code will react to these events.

I hope these articles help you better understand why software certification is important. I encourage you to voice your opinions, through writing articles and speaking at conferences,

on where the field should head. There are still no fixed nor proven answers on what is the best way to instill confidence in commercial software, and the emerging field of software certification needs the best minds available to help shape its future. ♦

REFERENCES

1. *INForum*, META Group, Stamford, Conn., 1998, p. 23.
2. J. Voas, "The Software Quality Certification Triangle," *Crosstalk*, Vol. 11, No. 11, Nov. 1998, pp.12-14.

About the Author



Jeffrey Voas is the co-founder and chief scientist at Reliable Software Technologies and the Quality Time editor for *IEEE Software*. He is also the principal investigator on research initiatives for

DARPA, the US National Institute of Standards and Technology, and the US Army. His professional interests include information security metrics, software dependability metrics, software liability and certification, software safety and testing, and information warfare tactics.

Voas has coauthored two books, *Software Assessment: Reliability, Safety, and Testability* and *Software Fault Injection: Inoculating Programs Against Errors* (John Wiley & Sons). He received his PhD in computer science from the College of William & Mary and is a senior member of the IEEE, IEEE Reliability Society, and IEEE Computer Society.

Voas can be reached at Reliable Software Technologies, Ste. 400, 21351 Ridgeway Circle, Dulles, VA 20166; e-mail jmvoas@rstcorp.com.

Computer Scientists & Engineers

your Vision

our Future

Southwest Research Institute is seeking to fill Modeling and Simulation, Software Engineering, and Software Design positions. All require a BS or MS in Computer Science, Computer Engineering, Electrical Engineering, or equivalent, programming in C, C++, and Java, on PC or UNIX based workstations. Candidates must have excellent communications skills and work well independently or as a member of a team. U.S. citizenship required for most positions.

Modeling and Simulation

Analyze, design, develop and implement modeling and simulation algorithms and software for training systems. Perform software lifecycle phases including requirements generation, software design, prototype development, testing, implementation, integration, maintenance and support, and documentation; interact with Institute staff and clients; travel to client sites. **Requires** experience using Ada; experience with distributed simulation (live, virtual, and constructive); knowledge of computer systems, software architectures, hardware-software interfacing techniques; and the ability to work with MS Word, Excel, and Project.

Software Engineering

Develop software for military and industrial applications on workstations and PCs. Perform software lifecycle phases including requirements generation, software design, prototype development, testing, implementation, integration, maintenance and support, and documentation; and make technical presentations. **Requires** course work in Computer Architecture, Software Engineering, Data Structures, Operating Systems, Compiler Theory, and Database Management courses or their equivalents.

Software Design

Design and develop software for military and industrial applications. Manage and execute software development including requirements analyses; system level design and documentation; software design, planning, validation, testing, integration, and maintenance. Applications include real-time data acquisition, graphic user interfaces, system level control processing and embedded algorithm implementation, and network interfaces (Internet, LAN/WAN, etc). **Requires** five years' experience in software system design and development with emphasis on Windows 95®, Windows NT, and UNIX (Xwindows). Must be able to take concepts and produce working software solutions, interface with clients, and make technical presentations.

Please submit your résumé to: Southwest Research Institute, Personnel Department #99172, P.O. Drawer 28510, San Antonio, TX 78228-0510 • E-mail: personnel@swri.org or Fax: (210) 522-3990

www.swri.org/jis3

An Equal Opportunity/Alternative Action Employer • A Public and Community Services Employer

- Ada
- C/C++
- CORBA
- Data Modeling
- Digital and Analog Electronics Design
- High-Level Architecture
- Information Technology
- Java
- NT/CIP
- Oracle Applications
- Paradox
- Project Management
- UNIX



Southwest
Research
Institute