Information and Quality Assurance

An Unsolved, Perpetual Problem for Past and Future Generations

Jeffrey Voas, SAIC
Linda Wilbanks, US Dept. of Energy

Every enterprise now lives and dies by the data stored on its computers and networks, and none can long survive if that data isn’t accurate and reliable. In addition to the damage that can come from the actual alteration of data, the impact on a company’s reputation can be substantial if customers are faced with unreliable, poor-quality products.

Quality assurance (QA) and information assurance (IA) programs play crucial roles in protecting assets and developing trustworthy products and services. Many technical solutions and approaches are available to IT teams engaging in IA and QA, and certain broadly accepted principles apply across the board, but success generally depends on proper responses to issues that affect enterprises at local levels. Platform and application heterogeneity present one sort of challenge, and domain-specific requirements and legal issues add other layers of complexity throughout. A clear understanding of what you’re trying to achieve is thus key up front. The following theme articles present case studies and experience reports in several QA and IA initiatives and describe lessons learned that could benefit others facing similar challenges.

Assurance and Trust

This issue of IT Pro focuses on IA and QA in software development, but assurance is at least equally important for systems, security, and safety. Although people often use these concepts interchangeably, doing so is a mistake because they represent separate areas of concern, each with its own specific challenges. For example, systems assurance requires the ability to build assurance cases in which evidence and context are propagated upwardly to enable specific arguments that system-level assurance has been achieved.

IA typically deals with confidence that certain data and information assets are secure and private. QA is a well-known field, particularly in areas such as the six-sigma manufacturing paradigm, which deals with defect-density rates (1 part in 1 million is defective, for instance). Although six-sigma can be applied to software with some success, it’s better known in the physical, manufacturing arena, because software isn’t mass produced, as with hardware parts.

Wiktionary.org lists the following among its definitions of assurance:
The three theme articles in this issue address parts of the assurance puzzle, which includes quality assurance, information assurance, software assurance, and system assurance. To help avoid costly mismatches, consider which piece of the puzzle you need to solve before looking at supposed solutions.
• A declaration tending to inspire full confidence; that which is designed to give confidence.
• The state of being assured; firm persuasion; full confidence or trust; freedom from doubt; certainty.
• Firmness of mind; undoubting, steadiness; intrepidity; courage; confidence; self-reliance.
• Excess of boldness; impudence; audacity; as, his assurance is intolerable.
• Insurance; a contract for the payment of a sum on occasion of a certain event, as loss or death.
• Any written or other legal evidence of the conveyance of property; a conveyance; a deed (in England, the legal evidences of the conveyance of property are called the common assurances of the kingdom).

Ultimately, the goal for assurance is to build trust, which Wiktionary.org defines as:

• Confidence in or reliance on some person or quality.
• Dependence upon something in the future; hope.
• Confidence in the future payment for goods or services supplied; credit.
• Trustworthiness, reliability.
• The confidence vested in a person who has legal ownership of a property to manage for the benefit of another.
• A group of businesspeople or traders organized for mutual benefit to produce and distribute specific commodities or services, and managed by a central body of trustees.

IT workers seek to leverage tools and processes to achieve such reliability and inspire confidence in users and customers. Several key elements come to bear on such efforts.

First, it’s important to note that software and information systems depend on facts. Those facts can be metrics, processes, standards, or other forms of evidence that a system graded by its behavior (thus using a nonphysical rather than a physical perspective) will operate in a manner that’s consistent with the notion that it’s “fit for purpose.”

Fit for purpose is one of three main ideas on how to certify software. In the first school of thought, you certify that you’ve satisfied a certain set of development, testing, or other processes applied during the prerelease phases of the life cycle. Of course, you’re certifying that the processes were followed and completed—demonstrating that they were applied correctly is a trickier issue.

In the second school, you certify that the developed software meets its functional requirements. Various types of testing and analysis, such as formal methods and specific operational profile testing, in large amounts, are available to accomplish this. The tricky part here is in demonstrating compliance with requirements: even if you can satisfy proof of confidence, you could unknowingly wind up with a false sense of accomplishment if the requirements prove to be incorrect, incomplete, or ambiguous.

In the third approach, you seek to certify that the software itself is fit for purpose. The term purpose suggests that two items are present: executable software and an operating environment. An environment is a complex entity that involves the set of inputs the software will receive during execution, as well as the probability that certain events will occur. It also involves the hardware on which the software operates—the operating system, available memory, disk space, drivers, and other background processes that are potentially competing for hardware resources, and so on.

Trust is a difficult issue to get a solid grip on. We all have a common idea as to what it means—generally very much in line with the definition stated earlier. Where IA and software QA exacerbate the problem is that they seek to assure qualities of nonphysical systems, such as software, requirements, processes, and standards. For example, software reliability modeling tries to apply hardware reliability models (based on time, wear-out, and decay) to software. Yet, software is deterministic and static; if left untouched, it can’t wear out or decay. This attempt to overlay a hardware model onto software led to the notion of “software rot,” although in reality, it’s not the software that rots, but rather the world around it. This huge difference between physical and nonphysical systems continues to make IA and software QA challenging goals that will go on for years to come.

The Articles
The articles that follow address multiple aspects of IA and QA. Software QA is a superset
of IA that addresses issues for all of the “-ilities” including reliability (such as mean-time-to-failure, mean-time-to-repair, mean-time-to-debug), as well as testing-stoppage criteria, compliance with standards, requirements elicitation, design-for-maintainability, and design for testability. IA deals mainly with the security and safety of data assets. As such, it’s more concerned with protecting information, whereas software QA includes broader topics related more to the confidence that can be placed in the software. As you read through these articles, consider looking back to these definitions and how the content maps to them. It’s also interesting to see what part of the overall puzzle each article is really addressing.

The first two articles are concerned with building the evidence in support of an assurance case via testing. Note that both deal with software. “Beyond Brute Force: Testing Financial Software,” by Mikhail Kharlamov, Alexey Polovinkin, Ekaterina Kondrateva, and Alexey Lobachev, presents the case for including domain experts as testers to increase efficiency and reliability in identifying faults. The authors use financial software as an example of how accounting for inherent complexities in a field can be difficult using standard testing approaches, which sometimes represent multiple parameters as single inputs. They suggest some specific issues to bear in mind in any test scenarios and point out ways in which integrated teams of IT and domain experts can uncover problems more time- and cost-effectively than domain-neutral testers.

D. Richard Kuhn, Yu Lei, and Raghu Kacker’s “Practical Combinatorial Testing: Beyond Pairwise” describes methods and tools for detecting failures that occur only when multiple components interact. Methods are widely available for pairwise testing, but recent advances in covering-array algorithms, integrated with model checking or other testing approaches, have made it practical to extend combinatorial testing. The authors argue that tests that cover all four-way or higher-strength combinations of parameter values are sufficient to identify all errors and provide high assurance.

The third article in our theme turns to the issue of quality as applied to the new IT paradigm of services, rather than simply software. The authors look at building a certification paradigm in which trusted third parties can ensure that the services you employ and build into your enterprise (via a SOA model, for example) provide the quality assurances that you require. In “Toward Quality-Driven Web Service Discovery,” Eyhab Al-Masri and Qusay H. Mahmoud demonstrate the need for mechanisms to let clients search for services that meet their quality of service requirements. The current absence of standards to define and regulate quality of Web services (QWS) complicates and undermines efforts to integrate QWS into the discovery process. The authors’ proposal for trusted third-party service brokers that could measure quality-related metrics dynamically and provide reliable metrics for comparing among similar services presents a mechanism for improving QA in an increasingly service-oriented world.

The grand challenges for software QA and IA are many. Given that this topic could easily fill volumes of encyclopedias, three articles clearly can’t cover all of the territory. A few particularly hot topics of interest that the current set of articles doesn’t really represent include ROI from software assurance techniques, how to build assurance cases, requirements elicitation, safety assessments, and how to test for malicious code. These difficult problems reasonably guarantee that solutions—even partial ones—will be slow in coming. Indeed, most companies will continue to face such questions for decades to come.

Jeffrey Voas is the director of systems assurance and a technical fellow at SAIC. He is on the IEEE Computer Society Board of Governors and is the past president of the IEEE Reliability Society. Voas is coauthor of Software Fault Injection: Inoculating Programs Against Errors (Wiley & Sons, 1998) and Software Assessment: Reliability, Safety, and Testability (Wiley & Sons, 1995). Contact him at j.voas@ieee.org.

Linda Wilbanks is CIO of the National Nuclear Security Administration within the US Department of Energy. She is on the IT Professional editorial board and has published many articles and conference publications, including book chapters, under Linda H. Rosenberg. Contact her at linda.wilbanks@nmsa.doe.gov.