Principles and Measurement Models for Software Assurance

Nancy R. Mead, CERT, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA

Dan Shoemaker, Department of Computer and Information Systems, College of Liberal Arts & Education, University of Detroit Mercy, Detroit, MI, USA

Carol Woody, CERT, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA

ABSTRACT

Ensuring and sustaining software product integrity requires that all project stakeholders share a common understanding of the status of the product throughout the development and sustainment processes. Accurately measuring the product's status helps achieve this shared understanding. This paper presents an effective measurement model organized by seven principles that capture the fundamental managerial and technical concerns of development and sustainment. These principles guided the development of the measures presented in the paper. Data from the quantitative measures help organizational stakeholders make decisions about the performance of their overall software assurance processes. Complementary risk-based data help them make decisions relative to the assessment of risk. The quantitative and risk-based measures form a comprehensive model to assess program and organizational performance. An organization using this model will be able to assess its performance to ensure secure and trustworthy products.

Keywords: *Measurement, Software Assurance, Software Product Integrity, Software Reliability, Software Trust*

DEFECTS ARE NOT AN OPTION

Software is a vital part of our culture. In fact, it might be successfully argued that our culture is built on software. Given that importance, it is essential that we can trust the software products we use. Unfortunately, however, "common software engineering practices permit dangerous defects that let attackers compromise millions of computers every year" (President's Information Technology Advisory Committee, 2005). In practical terms, the exploitation of defects costs the U.S. economy an average of $60 billion dollars annually (Newman, 2002). Worse, it is estimated that "in the future, the Nation may [be at even greater risk] as adversaries—both foreign and domestic—become increasingly sophisticated in their ability to insert malicious code into critical software systems" (Redwine, 2006). As a result, the exploitation of a software flaw in a basic infrastructure component such as power or communication could lead to a significant national disaster (Clark & Schmidt, 2002).

The Critical Infrastructure Taskforce sums it up this way: "The nation's economy

DOI: 10.4018/jsse.2013010101

is increasingly dependent on cyberspace. This has introduced unknown interdependencies and single points of failure. A digital disaster strikes some enterprise every day, [and] infrastructure disruptions have cascading impacts, multiplying their cyber and physical effects" (Clark & Schmidt, 2002). Most of the defects we are talking about here are traceable to programming or design flaws, and they do not have to be actively exploited in order to be considered a threat (President's Information Technology Advisory Committee, 2005; Jones, 2005; Redwine, 2006). They result from the fact that "commercial software engineering lacks the rigorous controls needed to [ensure defect-free] products at acceptable cost" (President's Information Technology Advisory Committee, 2005).

Given all the conjecture, it seems especially important to be able to assure that we can trust the software products that are critical to our national well-being. However, we figuratively know less about the condition of our major software products than we do about the circumstances on Mars.

That situation is mainly due to software being invisible, complex, and highly dynamic. Without the ability to know the precise state of the artifact's underlying makeup, it is almost impossible to ensure and sustain product integrity. Thus, making software products and the processes used to develop them visible and understandable is an important aspect of the overall responsibility to ensure the products' integrity. This clear understanding is tied to the ability to accurately measure what is going on in the construction and sustainment of the product at any point in its life cycle.

SEVEN HISTORIC PRINCIPLES FOR SOFTWARE ASSURANCE

Actionable data have to be collected, analyzed, and reported to assure the proper management of the integrity of any product. Because the software production and sustainment process is more adaptive than it is linear and is subject to change at the whim of the developer (and the customer), data provide an objective basis for making informed decisions about performance.

Product performance data have to be unambiguously understood to ensure that they are consistently interpreted. If each project stakeholder interprets data differently, poor and incorrect decisions could result. Therefore, it is essential that project stakeholders have the same understanding about what a given piece of data means. A common point of view requires a fundamental point of reference to guide stakeholders' interpretation.

The generic perspective for software assurance is based on these principles established in 1974 by Saltzer and Schroeder:

1. **Economy of Mechanism:** Keep the design as simple and small as possible.
2. **Fail-Safe Defaults:** Access can only be gained by permission (whitelisting) rather than exclusion.
3. **Complete Mediation:** Every access to every object must be checked for authority.
4. **Open Design:** Only allow access based on tokens of permission.
5. **Separation of Privilege:** Utilize multiple tokens of permission to gain access.
6. **Least Privilege:** Allow the minimum access necessary to complete the job.
7. **Least Common Mechanism:** Reduce common objects (information hiding).
8. **Psychological Acceptability:** Assure ease of use. (Saltzer & Shroeder, 1974)

Saltzer and Schroeder's principles still provide an excellent guide for general information security work. However, they were developed in a day when "buffer overflow," "malicious code," "cross-site scripting," and "zero-day exploits" were not part of the software profession. So in this modern world of large-scale, highly networked, software-dependent systems, it is necessary to develop a new set of principles to guide software assurance work.

The Committee on National Security Systems (CNSS, 2010) defines software assurance as follows: "Software Assurance (SwA) is the