Software Assurance: The Need for Definitions

Wm. Arthur Conklin
University of Houston
waconklin@uh.edu

Abstract
Software Assurance is a subject that has differing definitions depending upon who is providing them. An examination of the literature, including end user materials has produced a series of definitions. The disconnect between these definitions provides for miscommunication and disagreement about paths and options during the software development progress. This research project set out to discover the different definitions and construct a model where the differences can be resolved. Two principle forms of the definition have emerged, one product related and the other process related. The results are useful to educators, practitioners and students alike, each using the appropriate definition at the appropriate moment to achieve greater levels of understanding of options associated with secure software development.

1. Introduction
Software is a key element in many of today’s complex systems, whether they are aerospace, defense, healthcare, automotive, telecommunication or critical infrastructure. The software element has also increased in complexity, with many systems being comprised of millions or tens of millions of lines of code (Charette 2009). The development of complex software systems has created a need for software assurance to ensure the systems perform as required.

The lack of a consistent process and definition for software assurance results in differing priorities and focus for development teams with different organizations. As the importance of software assurance increases with the increasing complexity of software integration into critical infrastructure elements, the need for consistency could not be greater. To develop a science, one of the initial elements is the development of definitions and taxonomy to facilitate communication between researchers, scientists and practitioners. This effort is hampered when multiple, overlapping definitions are common in the environment.

2. Software assurance definitions
There are numerous definitions associated with software assurance. These definitions come from government agencies, trade and industry groups and academics. The vast majority of these definitions are similar, yet they have some unique differences.

These definitions have many common elements, such as a description of “a level of confidence” and “free from vulnerabilities”. These are references that tie the definitions into existing body of knowledge of security. They also have some differences, some minor, in wording, and some major, in focus.

The first step in analyzing the playing field comes from enumerating the definitions, a wide sampling of which follow.

2.1 Government definitions
The US government has produced a series of definitions associated with software assurance. From Department of Defense, we get a couple of similar definitions. From the Committee on National Security Systems comes the “official” glossary definition:

“a level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during
its life cycle and that the software functions in the intended manner” (CNSS 2006)

And from the Assistant Secretary of Defense with responsibility in this area comes a more streamlined definition:

“Software assurance (SwA) relates to the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software.” (Mitchell Komaroff OASD (NII)/DCIO 2005)

The National Institute of Science and Technology defines software assurance as:

"The planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures to help achieve:

Trustworthiness - No exploitable vulnerabilities exist, either of malicious or unintentional origin, and
Predictable Execution - Justifiable confidence that software, when executed, functions as intended." (Black 2005)

2.2 Aerospace definitions

The National Aeronautics and Space Administration (NASA) defines software assurance as:

“The planned and systematic set of activities that ensure that software processes and products conform to requirements, standards, and procedures.” (National Aeronautics and Space Administration (NASA) 1992)

The Federal Aviation Administration defines software assurance as:

“Software Assurance is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner.” (Federal Aviation Administration 2009)

2.3 Healthcare definition

From healthcare, in the form of academic literature, the following definition can be found:

“Software assurance is a rigorous, lifecycle phase-independent set of activities which ensure completeness, safety, and reliability of software processes and products. This is accomplished by guaranteeing conformance to all requirements, standards, procedures, and regulations.” (Cooper and Pauley 2006).

2.4 Industry definition

SAFECode is an organization comprised of firms in the software development space that has compiled a common set of operational models of software development.

“Confidence that software, hardware and services are free from intentional and unintentional vulnerabilities and that the software functions as intended.” (SAFECode 2008)

2.5 Related definitions

System assurance is related to software assurance, as many systems are just that, software. A DoD definition for system assurance is the following:

“System assurance is the justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle.” (NDIASAC 2008)

The Department of Homeland Security initiated a Software Assurance Working Group, which developed a series of definitive guides and directives including one that suggests that one of the aims of the assurance process is to build software-based systems that

“limit the damage resulting from any failures caused by attack-triggered faults, ensuring that the effects of any attack are not propagated, and recover as quickly as possible from those failures as well as ensuring that the software will continue to operate correctly in the presence of most attacks by resisting the exploitation of weaknesses in the software by the attacker, or by tolerating the failures that result from such exploits” (Redwine 2006)

The concept of viewing software as a system opens up a wide array of additional definitions in the form of systems related definitions. Systems is a discipline with its own set of standards and processes, definitions and taxonomy, and is beyond the scope of
this paper. As can be seen from the definition above, this additional material adds little in the way of additional meaning, and reducing the extraneous material associated with systems will make this paper cleaner and more to the point.

3. Software quality assurance

A variation on software assurance is the inclusion of the term quality. Software quality assurance adds the elements of quality to the elements of assurance. One of the first steps to make this transition is to define quality. Quality has many definitions, depending upon use, but the common generic form is one related to fitness for use. (Schulmeyer and McManus 1998)

Using an IEEE definition, quality assurance is defined as “a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.” (Jay and Mayer 1990; Radatz, Geraci et al. 1990) Software differs from standard hardware based quality issues in that software does not tend to fail because of fatigue. Repetitive use is not a quality issue associated with software. Instead, software suffers from scope creep. Systems employ software to allow the system to change and adapt over time, and this results in scope creep, which can lead to failure.

Because of the changing environment associated with software, one of the effective mechanisms to ensure quality or fitness for use is the same as in manufacturing based quality systems; process control. By focusing on process control, and the elements of the software creation process, one of the outcomes is software that has higher levels of fitness for use.

The inclusion of quality into the definition of software assurance is another divisive element for some groups of practitioners, as they tend to want to separate quality and safety issues from pure assurance issues.

4. Classes of definitions

One common difference between the definitions has been to point of focus. Some of the definitions are focused on the end product, software. The other set of definitions is focused on the process to get to the end product, software, either by development of acquisition. This difference is important to each constituency, as it is used in defining work processes and procedures.

This difference is important in another manner. For the product to have assurance, the process to obtain the product must be capable of delivering the desired level of assurance. So, in reality, both aspects are needed. A complete set of definitions and taxonomy needs to take into account both the attributes of the end product, software, and the process employed to develop and deploy it.

One method of separating and understanding the previous definitions is via the focus. Some of the definitions are focused on process aspects, while the others are focused on the state of the output of the process. These should not be seen as in conflict, merely focused on different aspects of a continuum of elements associated with the overall objective of developing a level of confidence in how the end system will perform.

The definitions from NIST, NASA and the healthcare definition from academia all focus on the process aspect of achieving software assurance. These definitions are not of an outcome, but of the methodology to obtain the outcome. The definitions from the Department of Defense, the FAA and SAFECode all deal with aspects of the output, the software.

One interesting nuance is the minor differences in the output related measures as they relate to types and causes of vulnerabilities. The DoD includes the qualifier for vulnerabilities, “either intentionally designed or accidently inserted”. The FAA definition is exactly the same as the CNSS definition from the Department of Defense. As this is the “official” definition posted by the department, it is not surprising to see it adopted by other agencies. The Department of Homeland Security has not posted an official definition, but in the postings on the subject, they tend to include all of the aspects of the CNSS definition.

5. Security and adversaries

An interesting addition to the trials and tribulations associated with software driven processes is the addition of the adversary. Early software efforts were not typically plagued by security issues – there was no significant “hacker” or adversary to abuse the system. As computer systems became more ubiquitous and more involved in our everyday lives and the economic aspects of society, they became targets for criminals. Today, computer systems are under siege and criminal activity has made computer security a significant issue for any system that can be tied to economic gain.

Computer or information security is a critical system attribute, for when it is not controlled, the system outputs are no longer completely under the initial design control and variances in outputs can
lead to lost efficiencies, failures and other issues. If an aspect of assurance is the operation of the system in a desired and predictable manner, then computer security becomes a foundational element to enable this level of operation to occur.

Security has advanced from simple elements to a well-defined and complex risk management based solution employed to effect change across the whole as opposed to individual components. Software assurance plays an important role in this overall risk management posture. Software can be used for critical business functionality and when relied upon for those elements, the management of failure associated with software is an essential aspect of risk management.

The development of language, terminology and operational relationships that play nicely with corporate risk management efforts is an essential element in the long-term success of a software assurance program. Risk management science is used to understand the causes of, the operational methods of dealing with and ultimately the management decisions associated with risks to the system. Software assurance efforts need to be integrated into the overall risk management methodology and posture of an organization. This connection can be moderated by the use of connectors that are weighted per system criticality measures.

6. Operationalizing the definition

To develop an effective, all inclusive set of definitions and taxonomy elements, the issues of quality, safety and security cannot be ignored. These elements can lead to failures, and by all accounts, the reduction and elimination of failures is one of the objectives of software assurance. The other aspect that cannot be ignored is the processes associated with the development, deployment and operation of software. Again, these aspects can result in system failure, so they need to be included.

Software assurance process is an oversight activity encompassing all phases of software development, including deployment and operation. As not all software products require the same level of assurance, or the same set of assurances against elements such as safety, quality and security, the details of the process requirements are tied to levels desired in the software output. One of the tenets of security is the concept of good enough security, security that provides sufficient risk reduction, but does not waste resources by attempting to be too secure.

The assurance processes employed in software development, including the breadth and depth of their scope, will depend on several factors one of which is software criticality. Software criticality is defined as the mission essential component of software – how important is the software to the overall system success. In safety critical systems, such as aerospace and healthcare, criticality can refer to an element of hazard avoidance. A system with high criticality can represent a hazard in the event of failure, and thus higher levels of assurance are called for as part of the risk management exercise associated with the system.

Using the term software assurance process to be inclusive of the elements employed to achieve assurance through system development efforts introduces the concept of assurance cases. Assurance cases are used when it is important to show that a system exhibits some complex property such as safety, security, or quality. An assurance case is a body of evidence organized into an argument demonstrating that specific claim(s) about a system are true. For instance a software assurance case can demonstrate that authentication failures result in system lockouts, preventing continual attempts to break in. This assurance case relates to a security property.

Assurance cases are an important part of software assurance as they operationalize aspects of the desired outcome, software assurance, in a testable and verifiable form. One of the challenges with the set of assurance cases is to develop cases and systems such that the assurance cases are not at crossed purposes. It is conceivable to define security cases that result in reduced operations, resulting in operational failures. This is not a failing of the assurance case aspect, but a warning to system designers that they have not taken all requirements into account as a holistic set, rather than as individual isolated elements.

One of the common elements of the definitions described earlier is that the software performs as intended. This is one of the weaker aspects, for many security issues occur in spite of the software performing as intended, in fact, some of them rely upon performance as intended. The key is for the system to perform as intended and only as intended. This is a system property known as correctness and is an important issue for some security related failure modes.

Taking the information gathered this far, we are ready to introduce some uniform definitions that operationalize the elements of software assurance and capture the requirements from the various agency and industry groups.
First is a definition for the software assurance process. Software assurance process is the collective set of activities employed to assure control over the software development, deployment and operational activities so that the results of the process are as intended and only as intended. This process aspect can include elements associated with software functionality, safety, security and quality. Proper functioning can be referred to as reliability, and the entire system, hardware and software as deployed in the environment is the standard unit against which the measures are recorded.

To assist in the design and development of the software, and the software process, the concept of assurance cases are used. Assurance cases are the collection of sets of bodies of evidence, organized as arguments that demonstrate claims associated with requirements, whether functional, safety, security, quality or process related.

Software assurance is then defined as the level of confidence associated with the output of the process, that the software can and will perform as desired, only as desired under the environmental conditions it is designed for, in the face of adversity and adversaries. Currently, this path is one that has been marked by the reduction of vulnerabilities, although in some cases, vulnerabilities are mitigated by design and defense in depth when the vulnerability is designed in via a faulty standard.

This issue of faulty standards is a real and increasingly troublesome one. The use of the Internet as a functional element of critical infrastructures brings for the following concern. How can you design critical infrastructure that has safety and security concerns for the general populace using components that are themselves specifically designed without safety and security in mind.

Because in the end analysis of a system, there are emergent properties that can act as mitigating factors this concern is noteworthy, but in most cases manageable. The challenge is in remembering that it is there, and not ignoring the increased risks when developing business cases.

It is important to build definitions and structure that operationalizes software assurance in a manner that is consistent with modern security operations built around risk management. It will be impossible to design, build and deploy perfect systems. Even if not theoretically impossible, perfect systems are generally not economically tuned. As safety, security and quality each require resources to address, excess quantities of any of these elements will result in increased resource utilization and higher costs. This is necessary to get to the level of desired operation in each of these aspects. Too much attainment will result in excess resource utilization, increasing system cost, reducing system return on investment and ultimately driving the system to failure from lack of resources.

7. Analysis

Examining the different versions of the definitions provides some insight into the challenges and issues associated with the defining organization. SAFECode, for instance, being a group of industry players, has adopted a consensus definition that is very lean and to the point. This definition is the free from extraneous qualifiers and remarkably close to the CNSS definition adopted by others. The CNSS definition clearly serves as a template for much of the thinking in the arena of software engineering, and it is longer than SAFECode definition because of the mentioning of the timing of error, being “any time during the lifecycle”. The addition of qualifiers to assist readers in understanding details of a definition are common when the definition is developed via a committee. Different members of a committee see the subject through slightly different lenses and add that perspective in the form of qualifiers. Generally, these qualifiers in the end relate to some form of “all” as in the CNSS definition including “any time during the lifecycle”. The additional value of these additions is twofold. During the development of a definition, this committee thinking ensures that all bases are covered and that a definition does not miss some critical element. If, after all considerations are made, the additional qualifying statements do not add differentiating value, they reasonably can be left off, making the definition smaller and easier to digest, yet retaining the same all-encompassing meaning. Frequently the qualifiers are left in place for a separate reason. This is to either tie the definition to an existing standard or body of work, or to endorse certain specific elements as important enough to deserve additional consideration as part of the definition.

A good example of this is in the NIST definition, which, being process oriented also addresses process related issues that the definers considered important with respect to software assurance. The terms “Trustworthy” and “Predictable Execution” tie their definition into other bodies of their work.

An important analysis question is whether or not the extra terms add value that needs to be embraced by the entire software community. In the case of the NIST definition, this would add additional terms to the other industry adopters – such as SAFECode, Healthcare, etc. The value of this additional
information is one that each community needs to address and determine on its merits and in the context of each discipline. One important finding is that even with all the qualifying terms, there are some common threads that can be used to create the universal definition.

The common threads are the terms “level of confidence”, “free of vulnerabilities”, and “perform as intended” for the product related definitions. The terms “planned and systematic set of activities” can be added to address process related definitions.

8. Academic definition

Examining the material presented in this paper, it is possible to combine definitions and build a single, universal set of definitions. Whether or not industry and government choose to embrace these definitions, because of how they are constructed, they will still serve academics and not be in conflict with what is currently being used in these other venues.

Beginning with the definition for process, because the process leads to the outcome, the term software assurance process is defined as “The planned and systematic set of activities designed to ensure that software processes and products conform to requirements, standards, and procedures.” Because this definition leaves the defining of requirements up to the end user, it is possible to have a definition of assurance with no teeth if the requirements are not appropriately defined. To guard against this, requirements for assurance is defined as “the set of items to ensure that the software process and products include Trustworthiness - No exploitable vulnerabilities exist, either of malicious or unintentional origin, and, Predictable Execution - Justifiable confidence that process or software, when executed, functions as intended and only as intended.”

Assurance cases are a tool used to demonstrate evidence of compliance with assurance objectives. Assurance cases are defined as “a body of evidence or collection of demonstrable facts, organized into an argument that demonstrates to a reasonable level of confidence that specific claim(s) about a system are true.” The level of rigor and degree of confidence are specifics that need to be determined at the time of implementation.

The objective of the software assurance process and assurance cases is to produce software items that have a defined attribute of software assurance. This can be defined as “a level of confidence that software is free from vulnerabilities and that the software functions in the intended manner and only in the intended manner.” This definition is similar to the CNSS definition, yet it omits the phrase “, either intentionally designed into the software or accidentally inserted at any time during its life cycle” referring to the vulnerabilities. This is because by leaving it out, there is no caveat, and hence all vulnerabilities are included. This includes vulnerabilities or errors that do not have to be exploited. It could be argued that the entire aspect of vulnerabilities can be left out of the definition, for the bottom line for assurance is whether or not the software performs as intended and only as intended, that any deviation from this is considered a failure.

9. Conclusions

Attempting to change the definitions scattered across numerous government agencies, industry and trade groups is a daunting and possibly impossible task. What is possible is the alignment of one group, specifically academics to a unified vocabulary and taxonomy. This will eventually lead to changes in the entire landscape and drive the change into the other organizations over time.

To achieve this alignment will require academics to embrace definitions and taxonomy elements in a common form and in a consistent manner. This paper is just the first step of the journey. Like any 12 step program, the first step is the acknowledgement of a problem. As seen from the many definitions, we have a problem.

The next steps are determining the path to proper behavior. In this case, the good news is that the path is relatively simple. The definitions are not that different and alignment can be achieved with minor changes to each. Thus, the uniform set of definitions supplied in this paper act as a superset of definitions, providing for the functionality of the different systems across the wide spectrum of software driven enterprises.

One of the challenges associated with advancing the science of software assurance comes from the field of complex systems science. In simple systems, it is common to perform system decomposition and treat each element independently. This works because the lack of complexity also results in the lack of emergent properties. One of the hallmarks of complex systems is their ability to exhibit emergent properties. Emergent properties are those properties exhibited by the overall system, yet attributable to no single aspect of the system. Series of overlapping defenses can result in much greater levels of protection than that expected by the individual contributions of each element. Safety, quality and security have all benefited from and in some cases rely upon emergent properties of a system.
These emergent properties make issues such as assurance cases, simple in the individual, but extremely complex as an overall set. Emergent properties make it difficult, if not impossible to separate issues such as quality, security and safety when examining the concept of assurance. Simply developing and agreeing on definitions does not solve the problems of managing assurance in the complex world of software today. The definitions merely act as a foundational starting point. Research and advancement of the science of systems of systems is necessary to gain the insights and tools necessary to understand and manipulate the significantly higher levels of complexity we are seeing today and in the future.

10. References


Radatz, J., A. Geraci, et al. (1990). IEEE standard glossary of software engineering terminology. NY, USA, The Institute of Electrical and Electronics Engineers,.


