Increasing the Likelihood of Success of a Software Assurance Program

Steven F. Mattern
Apogen Technologies, Inc.

As individual systems and collective system-of-systems become more complex, software-intensive, safety-critical and costly, software assurance becomes extremely important. Software assurance helps to reduce the likelihood of failure in terms of safety-related mishaps, system unavailability, loss of mission accomplishment, or security breech of the system or its related assets. Unfortunately, many program managers think they do not have the luxury of sufficient dollars and schedule to adequately address software assurance in the early phases of the acquisition life cycle. One only has to quantify the cost of one system failure or one system loss to comprehend its ultimate worth. Investing the resources in a software assurance program during the design, code, and test phases of a software development program will significantly reduce the likelihood of costly mishaps, failures, or system breeches during system operations and support.

When exposed to the quagmire of buzzwords, definitions, and terminology attached to most government system acquisitions and software development programs, there are many that apply to software assurance. The ones that come immediately to mind are *return-on-investment*, *value added*, *designing it in*, *system integrity*, *safe*, *reliable*, *available*, *secure*, and *risk managed*. While these words may be in the vocabulary of some design teams, there is usually a lack of passionate dedication in implementation that is attached to a lack of sufficient programmatic funding. Unfortunately, software assurance is one of those attributes of system design that is often not an enforced criterion for delivery, installation, or operation of complex systems. While customers and stakeholders are screaming for delivery of the over-budget, behind-schedule items of interest, software assurance is many times reduced to a *nice to have if there is remaining time and money*.

As our systems and system-of-systems become more complex, mission-critical, and safety-critical, it is definitely time for a paradigm shift toward a commitment to software assurance. Adequate funding and passionate commitment are the first steps toward a successful software assurance goal. Closely coupled with the two is a defined and planned process to be implemented to increase the likelihood of software assurance success. Contrary to some development program philosophies, *hope is not an effective methodology* to ensure this success. Success will be based on a defined and documented (stated) set of criterion to be managed within each of the acquisition life cycle phases. In addition, to fully understand the success criterion for software assurance there has to be a complete understanding of system failure [1]. This will be explored further as the elements of software assurance are defined and discussed. A successful software assurance program can then be measured in terms of cost savings, on-time deliveries, and software that is unlikely to contribute to unintended or undesired behavior. These attributes of success are more likely to be realized on a software development program when the output of the software assurance tasks actually minimizes the number of logical or functional flaws in the design architecture.

There are numerous methods or ways to increase the likelihood of a successful software assurance program for software development projects. These methods can be described as the essential elements of a software assurance program (see sidebar). Individually, each element can and will make a contribution to success. Collectively, they will provide a *knockout punch* in ensuring the likelihood of software assurance.

Essential Elements for Software Assurance

- An Adequately Funded Contract
- Program Management Support
- Defined Common Terminology
- Specialty Engineering Support
- Risk Management
- A Defined Set of Processes, Tasks, and Products
- Qualified Practitioners
- Defined and Applicable Metrics

An Adequately Funded Contract

If the Request for Proposal (RFP), the proposal, and resulting contract are not supportive of software assurance methods, the result is clearly *gloom and doom* for the specialty engineer who is actually tasked to perform the work. This is an age-old problem and it definitely needs modern-day rectification. *Business as usual* seldom makes an impact on the engineering disciplines involved with software assurance processes, tasks, and products. Admitting we collectively have a historical and contractual problem is the first step to successful mitigation or control of the issue. So, what can we do differently?

The Government Role

Government agencies need to get serious about ensuring the likelihood of success in this critical area. Modern development programs contain software-intensive, safety-critical, highly reliable, secure, survivable, and operationally effective requirements to go along with system-of-systems integration. The government needs to remain fully committed to ensure that processes are contractually in place within the software development life cycle to produce software that is safe, secure, reliable, and available to perform as intended. Boehm, Kind, and Turner in the *7 Myths about Software Engineering That Impact Defense Acquisitions* [2] state that these processes must be *architected in*. In addition, it must be adequately proposed and funded. Software assurance should be on the government checklist for RFP development. The government should ask for it specifically to be included in the contract proposal, and it should be part of the criteria for proposal evaluation and source selection. The RFP should ask that software assurance be adequately addressed in the software development plan [3] to be delivered as part of the contract proposal.

The Contractor Role

Historically, contractors seldom address software assurance adequately in proposal preparation and conversely it is definitely not sufficiently addressed in the cost volume of the proposal. This is due primarily to program managers' *perception of worth* for software assurance as compared to the more important, documented, and weighted source selection criteria that are contained in an RFP. If the government

specifically requests that software assurance be included in the software development plan and the associated costs for its implementation, the first step to success is complete. Most contractors are willing, if not able, to include software assurance in their development processes if it is adequately requested in the RFP and funded in the contract.

Program Management Support

Tightly coupled with an adequately funded contract is program and project management support. While a perfectly worded and sufficiently funded contract will provide motivation to the program manager, a demonstrated return on investment helps to solidify the motivational factors for supporting software assurance tasks. Program managers must be able to assess the engineering disciplines involved in software assurance activities and obtain a level of confidence that they are *getting their money's worth* with their implementation. This *money's worth* issue is due to the fact that the assurance activities are implemented to prevent *bad* things from occurring (mishaps, security breeches, and mission failures). Therefore, if nothing bad happens, did the program waste a lot of resources combating a very unlikely set of events? Most people can relate well to the discussions several years ago surrounding the Y2K issue and resources expended to combat this very critical issue. But, when the clock struck midnight and no major catastrophe occurred, it set off a series of discussions about whether we had wasted our critical resources on a non-occurring event. On the other hand, most will agree that one single Y2K national catastrophe would have cost more than we spent fighting against it. A good software assurance program will combat the likelihood of software contributing to catastrophic events.

The essential elements of a software assurance activity that will help add to the program manager's confidence to expend critical resources include *risk management*, including the severity and likelihood of failure; *a defined set of processes, tasks and products* that make sense to the design and test teams; *qualified practitioners* with adequate and demonstrated capability; and *defined and applicable metrics* – believable measurements of progress and completeness.

Defined Common Terminology

Historically, specific words have different meanings for individual design teams or stakeholders. The simple term *software assurance* possesses any number of different definitions among commercial contractors, individual design teams, government agencies, or international organizations. Government agencies and their supporting contractors must begin speaking a common language with defined and documented definitions. This is essential for agencies like the Department of Defense (DoD) to make the necessary strides in the development of system-of-systems that will be deployed, operated, and maintained by any or all of the armed services (and some in commercial airspace or similar environments). Software assurance efforts can and will benefit if this occurs.

To demonstrate the problem, a search was accomplished on the Internet using Google. The objective of the search was to find the *best* definition of software assurance available. Looking through the first 100 hits, the following were the only two definitions worthy of consideration:

- **Software Assurance.** A planned program whereby a customer prepays a percentage of their license price for future software releases [4].
- **Software Assurance.** A planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures [5].

However, by knowing exactly where to look, one can find the following definitions from sources other than Google:

- **Software Assurance.** ... 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 matter [6].
- **Software Assurance.** 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 [7].

In our Google search, the context of the Microsoft Corporation software assurance is prepaying for future application updates, and NASA's definition is moving toward what we would expect it to be (in 1993). But as we move forward to 2006, we are still faced with either conflicting or dissimilar definitions. These example definitions use the same two words, in sometimes separate environments with demonstrated confusion a possible net result.

For the purpose of this article, let us define software assurance as a *planned and defined set of activities that ensures software is complete in design to include safety, security, and reliability*. These activities ensure that the software functions as intended, not performing unintended or undesired functions, and adheres to a predefined set of standards that can be audited and verified.

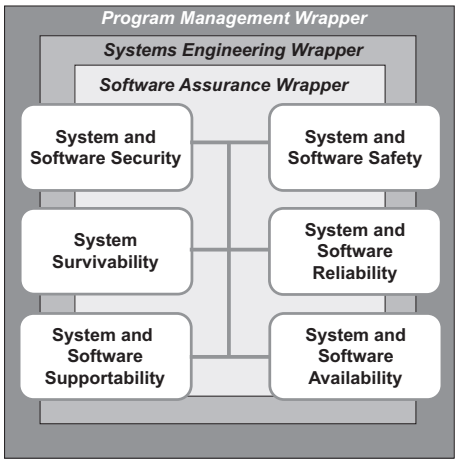

Figure 1: *Specialty Engineering Support*

Specialty Engineering Support

Specialty engineering is a general term used for engineering other than the typical hardware, software, or systems engineers on a design project. It normally includes system safety, reliability, logistics support, human factors, maintainability, security, and survivability. Unfortunately, these specialty engineers on most modern DoD projects do not represent a unified and integrated front that is needed to perform the necessary tasks for the software assurance activity. As described previously, most members of a product team or functional design team find it difficult to even agree on a definition of what software assurance is (or is not).

As depicted in Figure 1, software assurance includes multiple specialty engineering disciplines. Both the *system* and the *software* elements of the discipline are included as we know that system functionality and the hardware and human interface elements of the system are closely coupled with the functionality of the software.

Systems engineering is a key element to the success of the software assurance activity. It is here that the functional analysis is accomplished and that safety, security, and mission-critical functions are identified and managed in the design process. In addition, functional, physical, and logical interfaces are identified and managed as this is where system failure often initiates. The lead systems engineer should be integrating and managing the specialty engineering disciplines and ensuring processes, tasks, and products of each possess value. The lead systems engineer will also ensure

that the specialty engineers analyze the key functions of interest with a focus on both system failure and system success. They will also ensure that there is no duplication of effort and that each discipline is tied into the risk management activities of the program.

Risk Management

Program management is responsible for ensuring that programmatic and technical risk is identified, managed, and mitigated to acceptable levels of risk on a development program. To assist in this task, the *undesired events* of each of the specialty engineering disciplines must be identified and assessed using the risk criticality matrices that are common to risk management methods.

Table 1 [8] provides an example of the *look and feel* of a common criticality matrix. The matrix is used in risk management to categorize and prioritize resources based upon the perception of risk. Risk, in this context, can be in terms of safety, security, availability, reliability, mission risk, and/or supportability. This is all based upon the potential of losing functional or physical attributes of the system, and the command and control the software has over them.

Along with the criticality matrix is the acceptance or management authority over the risk identified. In the example, unresolved high-risk issues could only be accepted at the Program Executive Officer (PEO) level whereas project managers would be able to accept unresolved low-risk issues. The example matrix and the identification of the corresponding acceptance authority provided here are examples only. Individual programs must specifically define and tailor their own matrices.

A Defined Set of Processes, Tasks, and Products

The most important element of software assurance resides in the defined processes, tasks, and products that are produced and implemented by both the software design team and the specialty engineers. Without a defined and approved process, the entire software assurance effort reverts to an ad-hoc effort at best.

The specific disciplines within specialty engineering have one thing in common: they can all identify the *undesired events* that they do not want to occur or experience. In terms of safety, mishaps are the undesired event; in terms of reliability, it is an inoperable system; in terms of security, it is a functional or physical breech leading to system compromise. Regardless of the discipline and the undesired event, there are common tools and techniques to categorize the severity and likelihood of occurrence, identify failure modes and causes, and identify specific system (and subsystem) related requirements to eliminate, mitigate, or control these events or conditions to acceptable levels of risk. Because of this commonality and to simplify the example, only safety will be used within the context of the following discussion to illustrate the process. There are two basic processes to develop safer software: a software safety assurance process and a software safety hazard analysis process.

Software Safety Assurance Process

The Federal Aviation Administration (FAA) has been using a software safety assurance approach to develop safer software for years. This process is predicated on documented software level definitions that define the requirements, design, code, and test criteria to the software development team (refer to the top half of Figure 2). According to the FAA, *the software level is based upon the contribution of software to potential failure conditions as determined by the system safety assessment process* [9]. For the FAA, it is the utilization of RTCA DO-178B that provides the specific criteria for software functionality Levels A through E in descending order of consequence severity (Note: RTCA organized in 1935 as the Radio Technical Commission for Aeronautics, but it is now known as RTCA Inc.). Level A functionality possesses catastrophic severity consequences if it were to be lost, degraded, or if it functioned out of time or out of sequence. Each level possesses a lesser consequence of failure. Level B functionality possessed *critical* consequences whereas Level E functionality possessed no safety impact should it fail. Once the safety team defines the software level definition of a specific function, the software development teams implement the design, code, and test criteria required for FAA certification. This process has numerous benefits, including the following:

- The identification and categorization of functionality based upon safety consequences.
- Increased levels of development and test rigor for high-consequence functionality.
- The functional and physical partitioning of high-consequence functionality to reduce the likelihood of non-critical functions contributing to catastrophic or critical failure.
- The prioritization of critical resources (dollars, schedule, manpower) based upon sound risk management principles.
- *Safer* software and thus safer systems.

Software Safety Hazard Analysis Process

The DoD has relied on a hazard analysis process to develop safer software (refer to bottom half of Figure 2). This approach relied on the identification of system-level mishaps and their corresponding hazards. Mishaps and hazards are then categorized in terms of severity and likelihood of occurrence. By analyzing *snapshots* of a design as it matures, specific hardware, software, and human error causal factors are identified. Once these causal factors are identified in the context of the hazard initiation and failure pathway to potential mishap, then specific safety-related software requirements can be identified to mitigate or control the hazard to acceptable levels of safety risk. These safety-

Table 1: *Example of a Criticality Matrix* [8]

Risk Management – Criticality Matrix				
Probability	**Severity**			
	Catastrophic	Critical	Marginal	Negligible
Frequent	1	3	8	15
Probable	2	5	11	16
Occasional	4	7	12	18
Remote	6	10	14	19
Improbable	9	13	17	20

1-5	High Risk	Accepted — PEO Level
6-12	Medium Risk	Accepted — Program Manager Level
13-18	Low Risk	Accepted — Project Manager Level
19-20	Very Low Risk	Accepted — Systems Engineer Level

related requirements can be in the form of specific design architecture changes, or the addition of fault detection, fault tolerance, or fault recovery. These requirements are traced through the design implementation, the coded product, and then to the test and verification efforts. Benefits of this approach include the following:

- Safety-critical functions can be graphically modeled for ease of in-depth safety analysis [10].
- The causes of failure can be identified and mitigated at their specific initiation points.
- Failure of software functionality is analyzed in context with its hardware and human interfaces.
- Fault detection, isolation, annunciation, tolerance, and recovery can be more precise in its design implementation.
- *Safer* software, thus a safer system.

Integrating the Two Processes

The system safety community is beginning to realize that each of the two processes does indeed result in safer software. But, to obtain the *safest* software possible, modern-day developments need to integrate both the software safety assurance and the software safety hazard analysis processes into the software development and test activities [11].

Safety is used in this example of processes because the software safety community may have more mature methods, tools, and techniques to address software assurance. Because the software safety community processes mature tools and techniques, each specialty engineering discipline should closely evaluate what the safety community is using as it would either directly apply or could be modified slightly to yield exceptional results.

The Tasks and the Products

The defined processes for software assurance represents the *what* that needs to be accomplished. The specific *how to* tasks to fulfill the process are defined and implemented by individual teams. There are many ways to accomplish these tasks as long as the process is being followed. Specific tools such as Fault Tree Analysis and Failure Modes and Effects Analysis are common to the industry and provide the means to completely understand the context of failure. In addition, it is important that the tasks accomplished produce the engineering evidence and audit trail *products* for either system certification or customer acceptance.

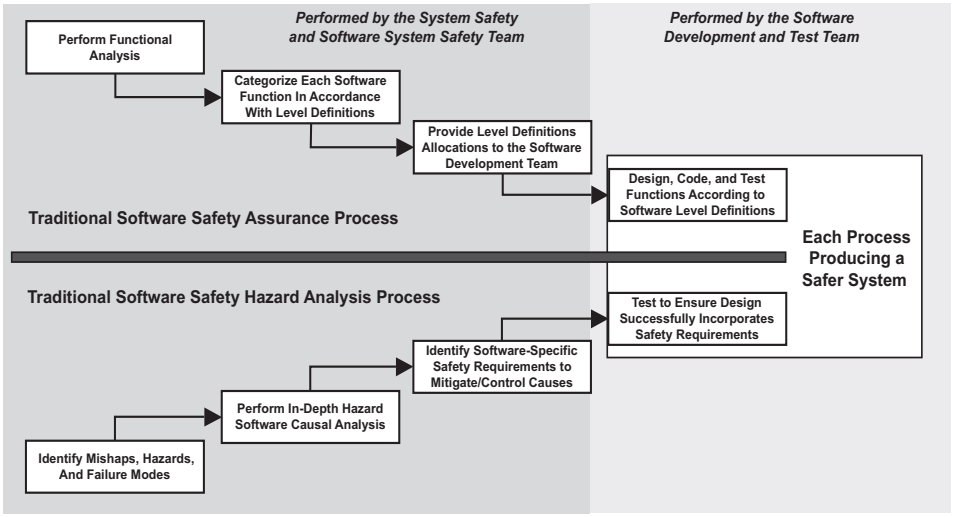

Figure 2: *System Safety and Software Safety Assurance Processes*

Qualified Practitioners

Specialty engineering is not a profession where *body count* is the most important factor. One good, trained, qualified performer is much better than three untrained, struggling practitioners. Contractors have a bad habit of putting engineers with dwindling contractual coverage into positions with specialty engineering just to keep them employed. This is far from optimal and should not be accepted by the stakeholder of the contract.

Also, a lesson learned (and re-learned over and over again) is associated with the original designer being tasked to do the safety or reliability analysis due to the lack of a qualified person being on staff. To put it bluntly, asking the designer to perform failure analysis on their own design is like asking a brand new mother to identify and document how ugly her new baby is. Just as new mothers only see the beauty of their new child, design engineers only see the natural beauty of their design. Asking them to see or identify failure modes and conditions (ugliness) of their design will historically not yield the true results required for a high-fidelity failure analysis.

The bottom line here is that specialty engineers (system safety, reliability, vulnerability, security, etc.) are trained in the art and science of systematically breaking the system down to identify the failure conditions and their contributing failure pathways and initiators. They are trained to categorize and prioritize risk based upon severity and likelihood of occurrence in order to facilitate wise decision making from management and design engineering in risk mitigation and control. These experts are the individuals that should be doing the work.

Defined and Applicable Metrics

In order to confirm or obtain confidence in a defined software assurance process, there must be applicable metrics that record the score of *how well we are doing*. Applicable metrics provide the technical and managerial decision makers with certainty that the resources expended (upfront in the development process) are actually providing the necessary value to the development effort. In addition, these provide the ultimate stakeholder or customer with the confidence that the system is meeting its assurance requirements or objectives.

Here again, it is important to have qualified individuals for any given project who are adept at establishing credible and verifiable metrics. To support this premise, one must consider that many customers desire or require quantified (or quantifiable) metrics that are supported by engineering evidences and artifacts. It is extremely important to have high-fidelity, quantified results that can be supported and verified by a repeatable process.

On a recent project, my company used a specific tool to provide a quantifiable set of metrics to the customer. This tool is modifiable whereby the metric outputs are based upon the inputs of specific assessment criterion. By evaluating the number and types of findings, the level of risk mitigations and controls, and the overall fault tolerance of the system, a specific output score is generated. While this effort was directed at one phase of the acquisition life cycle, tools like this can be modifiable and used in each of the software life cycle phases.

Regardless of the tools or metrics used, they should include ways to measure each functional discipline of the software

assurance activity. That is, safety, security, reliability, etc. should all be assessed, measured, and scored for decision-making and auditing purposes.

Summary

Software assurance is a maturing discipline that is vital for complex software development projects that possess safety, security, reliability, and mission critical attributes. Although the essential elements for a software assurance program presented here are described in a basic abstract format, each element should be further defined, expanded, and refined for individual DoD software-development projects. Each element presented is important to the success formula. However, if one element of this formula is absent, do not let that hinder the inclusion of the remaining elements. By implementing these elements, program, project, and technical managers can increase the likelihood of having a defendable and high fidelity software assurance program.◆

References

1. Raheja, Dev G. Assurance Technologies – Principles and Practices. McGraw-Hill, 1991.
2. Boehm, B., Peter Kind, and Richard Turner. "Risky Business; 7 Myths about Software Engineering That Impact Defense Acquisitions." Project Manager (May-June 2002).
3. Rosenberg, Linda H. "Lessons Learned in Software Quality Assurance." Software Tech News 6.2 (Dec. 2002).
4. Semilof, Margie. "Microsoft Licensing: A Special Report: Licensing 6.0 and Software Assurance: Who Should Renew?" 16 Sept. 2003 <http://searchwinit.techtarget.com/originalContent/0,289142,sid1_gci927989,00.html>.
5. National Aeronautics and Space Administration. "Software Assurance Standard: NASA-STD-2201-93." 1992.
6. Committee on National Information Security Systems. "National Information Assurance Glossary." Instruction No. 4009, 2006 <www.cnss.gov/Assets/pdf/cussi_4009.pdf>.
7. Department of Defense (DoD). "DoD Software Assurance Initiative." 13 Sept. 2005 <https://acc.dau.mil/CommunityBrowser.aspk?id=25749>.
8. Mattern, Steven F. "Introduction to Risk Management." System Safety Management Course. University of Washington, Seattle, WA. Mar. 2006.
9. RTCA/DO-178B. "Software Considerations in Airborne Systems and Equipment Certification. Requirements and Technical Concepts for Aviation." 1 Dec., 1992.
10. Mattern, S, E. Elcock, and E. Larsen. "IMPACT – A New Tool for the Software Safety Engineering Toolbox, Integrated Message and Process Analysis Control Technique." Proc. 20th International System Safety Society Conference Denver, CO, 2002.
11. Mattern, Steven F. "Comparing Software Safety Engineering with Software Integrity Methods and Techniques The Implications to Future." Department of Defense Acquisitions, 22nd International System Safety Society Proceedings. Providence, RI, 2004.

About the Author

Steven F. Mattern is vice president with Apogen Technologies, in McLean, VA. He manages the Software and Systems Analysis Division that specializes in software development and software assurance technologies. Engineers in his division have been performing software assurance-related tasks for more than 12 years for both government and commercial clients. Mattern is a Fellow member of the International System Safety Society and holds the position of Director of Education and Professional Development for the Society. He is the integrating author of the *Tri-Services Software System Safety Handbook*, and currently teaches the System Safety Management and Software Safety Engineering Courses at the University of Washington. Mattern has a Bachelor of Science degree in Industrial/Electronic Technology from the University of Wyoming and a Master of Arts in Computer Resource Management from Webster University.

Apogen Technologies, Inc.
1308 Bellevue BLVD N
Bellevue, NE 68005
Phone: (402) 502-3657
E-mail: steve.mattern@apogen.com