Software Assurance for Systems of Systems

John Goodenough
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
+1 412-268-6391
jbg@sei.cmu.edu

Linda Northrop
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
+1 412-268-7638
lmn@sei.cmu.edu

ABSTRACT

J*ustified* confidence in system and SoS behavior requires software assurance theories and principles that don't exist today. Using such theories and principles, organizations would have a better basis for confidence in deployed system behavior, and at the same time, these theories and principles could be used to make the assurance process more efficient and effective.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *reliability, validation.* F.3.1 [**Theory of Computation**]: Logics and Meanings of Programs – *logics of programs, invariants.*

General Terms

Reliability, Security, Verification.

Keywords

software assurance, systems of systems, ultra-large-scale systems, failure modes, argumentation, FMECA, FTA

1. INTRODUCTION

Because of the increasing size and interdependent/interconnected nature of today's large information technology (IT) systems and systems of systems (SoSs), achieving an acceptable and *justified* level of confidence is becoming more difficult. Moreover, today's approach to assuring the behavior of such software-reliant systems takes too long. For example, integration and acceptance testing (the typical approach used today to demonstrate acceptable SoS functionality) can take months and may have to be completely redone when changes are made, causing undesirable delays in fielding needed capability. To make the assurance process more efficient (and effective), we need to answer foundational questions such as the following:

- Which assurance activities provide the biggest increase in justified confidence that a system will behave acceptably when fielded?
- Can some assurance activities be curtailed without reducing justified confidence in deployed system behavior? For example, when is it reasonable to stop testing a system, and why?
- What insights do assurance activities yield into the residual risks that are present in a deployed system?
- What evidence is most probative in deciding whether a system should be released?
- What is a principled theoretical basis for asserting that sufficient confidence has been obtained in software-reliant behavior?
- What types of justification are more or less acceptable?
- Is a proposed confidence level well justified by sound principles and theories?

Adequate answers to these and similar questions do not exist today. And although the above questions apply to all kinds of software-reliant systems, they are becoming more important as an increasing number of these systems are systems of systems and ultra-large-scale systems [5].

Because system of systems (SoS) constituents are decisionally autonomous and interact independently, special assurance problems arise when considering SoS assurance. In particular, acceptable interaction depends on commitments that each constituent provides to others and commitments on which it depends.[1] The relevant assurance questions are: (1) is the set of commitments sufficient to ensure a desired SoS behavior or quality attribute, and (2) what is the impact if a commitment is not kept? For example, consider information that is shared across constituents of IT systems of systems. In some cases, this information is time sensitive and should not be used when it is no longer current. How do we gain justified confidence that the performance commitments made among system constituents collectively guarantee that a constituent will always[2] receive information in a timely manner; or alternatively, if timeliness can be guaranteed only probabilistically (which is likely in a SoS), how do we gain justified confidence that a constituent will not act unwittingly on outdated information? How do we gain justified confidence that such devia-

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

[1] Depending on the nature of the constituent and the nature of the interaction, a commitment might be called an interface specification, a service-level agreement, a memorandum of understanding, a guarantee enforced by the system architecture (e.g., a constraint on bandwidth utilization), etc. In general, "commitments" characterize the nature of interactions among SoS constituents. The extent to which commitments are enforced or monitored depends on the SoS, the constituents involved, and the nature of the commitment. The impact of failing to live up to a commitment is a subject for assurance analysis.

[2] "Sufficiently often" would be a better criterion than "always," but is even harder to demonstrate.

tions from desired behavior (*failure modes)* have been adequately mitigated?

Assurance of timely delivery is an SoS-level property. If the commitments are not correct, each constituent could live up to its commitment for timely delivery and yet, some recipient could occasionally receive information that is outdated. How do we analyze system commitments and interactions to uncover such SoS failure modes? Field testing may uncover some of them, but can such exercises give sufficient confidence that something critical will never be missed?

In general, if we are to have *justified* confidence in system and SoS behavior, we need better software-assurance theories and principles. Using such theories and principles, organizations will have a better basis for confidence in deployed system behavior, and at the same time, these theories and principles can be used to make the assurance process more efficient and effective.

2. CURRENT LIMITATIONS

2.1 IT SoS Assurance Today

Today, extensive testing is the primary method for obtaining confidence in the behavior of IT systems of systems. But testing, by itself, is a weak method of assuring that systems of systems will behave acceptably under unusual conditions. In part, this is because most testing effort is devoted to examining system behavior under "sunny day" conditions (i.e., demonstrating that required functionality is present under normal conditions), and in part it is because testing is inherently weak as a method of finding software defects that occur only rarely, e.g., only under heavy loads. Because the current approach usually gives little insight into the behavior of an SoS under untested conditions, systems are released on the basis that testing has (probably) been sufficiently thorough that no severe errors will occur in fielded systems, and that even if they do occur, the system will be sufficiently functional that these errors can be tolerated. The inherent weakness of testing is significant for assuring SoS functional behavior, but it is even more significant for security aspects. The inability to have sufficiently justified confidence in the security of SoS behavior sometimes means that user convenience and capability have to be restricted to satisfy security concerns.

To develop justified confidence in system behavior, one must integrate various types of evidence (such as test results, proofs, modeling and simulation results, architectural analyses, etc.) into an argument demonstrating that a system not only delivers intended functionality and has appropriate security properties but that its failure modes are sufficiently unlikely and are not catastrophic. When the property of interest is reliability, this collection of evidence and argument has been called a *reliability* case in the hardware engineering community [6]. When the property of interest is safety, a different collection of evidence and argument is called a *safety* case by the safety-critical community [7]. When the system is critically dependent on software however, there is little agreement on what constitutes a valid or sound argument for demonstrating that the software will behave acceptably.

A significant deficiency in current assurance practice is that we have only a weak understanding of what makes one software assurance argument stronger than another. For example, we intuitively understand that when a proof of a system property is combined with test evidence, the argument is stronger than if just one of these elements is provided. But what is the theoretical basis for this intuition? More generally, what is the theoretical basis for showing that one combination of argument and evidence is stronger than another?

One approach that has been taken to answer this question is the application of Bayesian reasoning to assurance cases [3]. Assurance cases are a mode of structured argument in which top-level claims about system properties are repeatedly decomposed into subclaims until finally evidence is presented showing that a subclaim is true. In a Bayesian analysis of an assurance case argument, conditional probabilities are associated with subclaims and evidence, and Bayes' Theorem is used to generate a conditional probability that the top-level claim is true. While theoretically interesting, there are practical obstacles that have prevented its use (e.g., obtaining robust estimates of a priori probabilities).

Another approach has been to develop an *operational profile* [4] representing how often certain demands on a system are expected to occur. Tests are prepared that mirror the operational profile. Depending on the rate of errors discovered by the tests, one can draw a statistically sound conclusion about the likelihood of failure in the operational environment. The advantage of this approach is that the most frequently used parts of the system are exercised the most often, and this makes it more likely that latent defects in common system uses will be detected. Consequently, if no defects are detected, one can justifiably argue (from a statistical point of view) that the likelihood of a defect being revealed in actual system operation is lower than some statistically derived value. The disadvantage of this approach is that it is hard to establish an accurate operational profile. Perhaps more importantly, errors that rarely have a critical effect will not necessarily be revealed by this approach, especially if the defect causing the error is difficult to reveal through testing. (Such errors include race conditions, memory leaks, and vulnerabilities targeted by attackers.)

Another approach commonly used in safety-critical systems or in high-assurance applications (e.g., spacecraft software) [2] is to combine testing with detailed analyses and proofs of critical parts of a system, looking for failure modes and the potential impact of possible defects. The argument showing that all critical aspects of a system design have been adequately addressed is documented on a best-effort basis, with more effort being spent on eliminating or mitigating potentially catastrophic failure modes. The quality of the argument and evidence is typically determined by inspection, experience, and consensus. No particular reasoning is used to argue how much additional "justification" is provided by a particular piece of evidence or argument. This approach has not typically been used for IT systems, in part because these assurance approaches are currently time-consuming and expensive and in part because the consequences of IT system failure have not been considered significant enough to warrant their use.

2.2 Developing Justified Confidence for SoSs

Current assurance approaches and research have neglected the special problems posed by systems-of-systems assurance. Because such systems consist of independently interacting entities, it is difficult to ensure that their mutual commitments lead to SoS behavior that is acceptable. Some SoS properties can, in principle, be assured today through analysis. For example, if certain information is time sensitive, architectural performance modeling can

develop upper bounds on end-to-end performance behavior. Often, however, these bounds are unduly pessimistic because they assume worst-case behavior on the part of each constituent. When using replicated databases, one can apply mathematical theories to determine how long it will take to ensure that all replicants have consistent data [8]. But no similar theories are available to assess the integrity and confidentiality of information exchanged among constituents having different abilities and commitments to maintain originator restrictions on use and further dissemination. Developing trust that information will be handled appropriately is not too difficult when it is being used within a security enclave, but the problem becomes much more difficult when critical information is used across organizational units, or across Services, or with coalition partners.

Although agent-based modeling and simulation have been used to gain insights into the collective effect of various types of coordination failures among independently interacting entities, this kind of analysis has not been much applied to IT systems of systems. Consequently, relatively little is understood today about how to impose design constraints that limit or reduce such SoS failure modes.

The current practice is to understand, develop, assure, and then finally field systems. In the understanding and developing phases, software and system engineers often fail to recognize that acceptable behavior of software-reliant systems depends not just on acceptable functionality but also on non-functional properties such as security, reliability, openness, adaptability, and so on. Failure to address these non-functional properties, or quality attributes, early in system development or evolution results in frequent unacceptable system behavior and delays in fielding. There are too few early architectural tradeoff analyses based on quality attribute reasoning.

3. RESEARCH NEEDS

Suggested research falls in two parts: developing theories and principles for determining which assurance activities and arguments contribute most to obtaining justified confidence in a system's behavior, and developing valid assurance arguments to ensure that SoS failure modes are adequately addressed.

3.1 Assurance Argumentation

Failure Mode, Effects, and Criticality Analysis (FMECA) and Fault Tree Analysis (FTA) are standard techniques used to find design errors in hardware systems. The notion of doing FMECAs and FTAs for software systems has been proposed by others (Haapanen 2002) but given how software systems are architected and documented today, it was never quite clear how to trace out the effects. But a structured argument (demonstrating some property of a system) captures the reasons why the system is believed to work. One could use an FMECA/FTA approach on such an argument structure. For example, one could hypothesize that some claim is false or only partially true or only true some of the time (i.e., a "fault" in the argument structure). The next step would be to analyze the effect of this defect on higher level claims in the argument, deciding how the failure might be manifested in actual system operation, how critical the effect of this failure would be on system behavior, how often it would occur, etc. In short, develop an estimate of the impact of falsity of the claim. Effort devoted to demonstrating that the claim is true will "buy down" the impact of its being false. This then becomes a measure of value of each claim in the argument structure and a way to allocate assurance resources based on buy-down of risk.

The above approach identifies the impact of a claim's being false. A complementary approach is to look at what argumentation deficiencies might make the claim false (i.e., what oversights or mistakes might exist in the supporting argument). For example, suppose a proof has been provided that some system property holds. What would make this proof irrelevant to the actual system? Any such defect would mean that the proof has no value in demonstrating that the hypothesized system property holds, and so the overall argument would be weakened. Given this impact, we would look at the various ways in which proofs can be irrelevant (e.g., by being based on an incorrect model of the system's actual implementation) and see what assurance efforts have been performed to eliminate these possible sources of proof irrelevancy. Based on experience, we can estimate how often experienced engineers make these kinds of mistakes, and we then have an estimate of the likelihood that this kind of assurance deficiency is present. This leads to an estimate of the value of gathering assurance evidence showing that the deficiency is not present.

Both approaches provide a new way of evaluating the soundness of structured arguments. Both approaches are likely to be successful because they are based on the standard FMECA and FTA approaches to reasoning about design errors in hardware systems.

3.2 SoS Failure Modes

If we are going to achieve increased confidence in the behavior of a system of systems under all circumstances, we need to understand the ways in which such systems fail, and in particular, the failure modes that are distinct from those of monolithic systems (whose evolution and content is completely under control of a central authority). For example, because SoS constituents evolve independently, it is possible for the collective set of evolutions to gradually degrade some desired overall SoS quality, e.g., end-to-end performance for certain threads. How can such degradation be detected and mitigated, or what constraints can be placed on constituent evolution that help to ensure maintenance of desired SoS properties under all (and changing) operational conditions? If SoS constituents fail to adapt to changing operational conditions because no individual change can satisfy the collective need, then failure to adapt because of failure to collaborate might be considered a particular type of SoS failure mode or failure mechanism.

4. CONCLUSION

Our goal is to make the assurance process more efficient and more effective. By establishing methods for evaluating the soundness of an assurance argument, we will be able to establish criteria for determining which elements of the assurance process contribute most to establishing justified confidence in system behavior. With this information, organizations can begin to intelligently make choices from among assurance techniques, concentrating on those that are most efficient. By addressing SoS failure modes, we can begin to identify patterns of failure that will be increasingly important in a world in which systems are increasingly interdependent. The identification of underlying patterns of SoS failure will lead to quicker identification and mitigation of such failure modes in SoS design, construction, and evolution.

5. ACKNOWLEDGEMENTS

The ideas in this paper are the result of the research and ongoing discussion of the members of the System of Systems Software Assurance Initiative of the Research, Technology, and System Solutions Program of the Carnegie Mellon University's Software Engineering Institute. The authors are technical leaders of that organization and are greatly indebted to their colleagues.

6. REFERENCES

[1] P. Haapanen and A. Helminen. Failure Mode and Effects Analysis of Software-Based Automation Systems. Helsinki: STUK, 2002.

[2] T. P. Kelly and R. A. Weaver. "The Goal Structuring Notation—A Safety Argument Notation." Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases. IEEE, 2004.

[3] B. Littlewood and D. Wright. "The use of multi-legged arguments to increase confidence in safety claims for software based systems: a study based on a BBN analysis of an idealised example." IEEE Transactions on Software Engineering 33, no. 5 (May 2007): 347–365.

[4] J. D. Musa. "Operational Profiles in Software-Reliability Engineering." IEEE Software 10, no. 2 (March 1993): 14-32.

[5] Northrop, L. et al. 2006. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA.

[6] SAE International. SAE JA1002, Software Reliability Program Standard. SAE International, 2003.

[7] UK Ministry of Defence DEF STAN 00-55 Requirements for Safety Related Software in Defence Equipment. 1997.

[8] W. Vogels. "Eventually Consistent." CACM 52, no. 1 (January 2009): 40-44.