

Tool Use Within NASA Software Quality Assurance

Denise Shigeta, Dan Port
Shidler College of Business,
University of Hawaii
dshigeta@gmail.com, dport@hawaii.ed

Allen P. Nikora, Joel Wilf
Jet Propulsion Laboratory,
California Institute of Technology
{Allen.P.Nikora, Joel.M.Wilf}@jpl.nasa.gov

Abstract

As space mission software systems become larger and more complex, it is increasingly important for the software assurance effort to have the ability to effectively assess both the artifacts produced during software system development and the development process itself. Conceptually, assurance is a straightforward idea – it is the result of activities carried out by an organization independent of the software developers to better inform project management of potential technical and programmatic risks, and thus increase management’s confidence in the decisions they ultimately make. In practice, effective assurance for large, complex systems often entails assessing large, complex software artifacts (e.g., requirements specifications, architectural descriptions) as well as substantial amounts of unstructured information (e.g., anomaly reports resulting from testing activities during development). In such an environment, assurance engineers can benefit greatly from appropriate tool support. In order to do so, an assurance organization will need accurate and timely information on the tool support available for various types of assurance activities. In this paper, we investigate the current use of tool support for assurance organizations within NASA, and describe on-going work at JPL for providing assurance organizations with the information about tools they need to use them effectively.

1. Introduction

A large number of tools have been developed that can be used to assess the quality of software systems during development. These include, but are not limited to, (1) modeling and analysis tools such as model checkers, theorem provers, and code analyzers, (2) measurement tools such as software reliability growth models and test coverage analyzers, (3) traceability analysis tools, and (4) tools for assessing product and process compliance to standards. There appears to be significant interest in assurance tool development and evaluation research. Indeed, over 50% of the technical presentations, excluding technical updates, at NASA’s 2009 Software Assurance Symposium (SAS) [1] technical presentations were directly related to

assurance tools (examples: “Automated Tool and Method for System Safety Analysis” and “Technology Infusion of CodeSonar into the Space Network Ground Segment”). However, on a recent survey of JPL quality assurance personnel and assurance customers, 100% of the respondents either agreed or strongly agreed with the statement “tool use and automation for SQA is limited – manual methods dominate”, indicating clearly that these types of tools are frequently not used to their full effect within quality assurance efforts. Further investigation at JPL revealed that impediments to their use include high cost, lack of user training, a steep learning curve, failure to meet critical user needs, lack of institutional coordination, and high overhead in identifying and evaluating potentially useful tools. In addition, there is often little understanding of how these tools are related to each other, and how they are most effectively used together within a given assurance effort. These factors contribute to the observed underutilization of tools in supporting software assurance activities, decreasing the overall effectiveness of the assurance organization, and increasing the likelihood of partial or complete mission failure resulting from a higher residual defect content in the fielded system.

2. NASA Software Assurance

Software systems have become increasingly critical sources of risks in the missions and systems built by NASA. *Software Assurance (SA)* is “the planned and systematic set of activities that ensures that software life cycle processes and products conform to requirements, standards, and procedures.” [2] It is also defined as “the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle, and that the software functions in the intended manner.” [3] Systems and Software Assurance is an umbrella risk identification and mitigation strategy for mission, reliability and safety assurance of software systems. The purpose of full life cycle assurance activities is to identify and reduce risks arising from quality uncertainty. For example, “Are the identified risks sufficiently comprehensive? Has anything important been forgotten? How frequently

has the same mistake been made in the past? What have been the consequences of such mistakes?” Assurance is typically performed by assurance professionals within a Software Quality Assurance (SQA) engagement, while conducting verification and validation (V&V) of systems and software artifacts, or within an independent V&V (IV&V) assessment. The Software Quality Assurance (SQA) group at NASA’s Jet Propulsion Laboratory (JPL) is one of many organizations developing or assuring large mission- and/or safety-critical systems.

Numerous studies have observed that error detection in the earlier phases of a software system lifecycle yields much lower costs of fixing and reworking software systems (a factor of 50 to 200) [4],[5],[6],[7]. Among various systems and software assurance activities, risk assurance and requirements traceability assurance are two critical and valuable ones that can be addressed early and incrementally revisited in the system development life cycle. When a party not directly involved in generating the risk specifications (i.e., risk description and mitigations specified in projects’ risk analysis documents) or Requirements Traceability Matrices (RTM) performs assessment on these artifacts, it is generally called assurance. Assurance of systems and software risk documents (risk assurance) aims to increase the confidence that risks identified are complete, specific, and correct. Requirements assurance is concerned with independently (from requirements developers) assessing and ensuring the quality of the requirements. Chiefly this includes assessing the correctness and completeness of the requirements for which tracing is key. This tracing encompasses requirements at various granularities as well as their relationships with other artifacts (e.g., tracing between requirements and test cases) and within artifacts (e.g., tracing between Nonfunctional and Functional requirements).

3. Need for Software Assurance Tools

As software systems become increasingly complex, they become larger sources of risk in a project. The purpose of all assurance activities is to reduce risk arising from quality uncertainty. Manual methods of SA contribute to this risk via human error and lack of coverage in assuring a system, among others. SA tools can provide increased efficiency, through automation, as well as increased effectiveness, by showing that assurance activities provide more confidence in the software system.

3.1 Tools to Increase Efficiency

Current as well as previous assurance efforts at JPL include work focused on improving the efficiency

of analyzing large bodies of requirements for specific characteristics. Previous work at JPL indicates that defective or misunderstood requirements are a significant source of anomalous behavior observed in a system during mission operations [8],[9]; detailed and accurate analyses of the requirements can reduce the risk of introducing defects into the requirements themselves and propagating those defects into the implemented system. Since a typical space mission of the type developed at JPL can be specified by 10,000 or more individual requirements, unaided manual analysis is no longer efficient for assuring that specification documents accurately reflect the system being developed.

An illustrative example of the type of requirements analysis that assurance personnel perform is the problem of assuring a functional to non-functional requirements traceability matrix (RTM). Determining the correctness and completeness of the many-to-many relationships between functional and non-functional requirements (NFRs) is a particularly tedious and error prone activity for assurance personnel to perform.

For NASA the development of RTMs is a mandated activity. Because of the high risk typically associated with NASA projects, assurance of RTMs is also a mandated activity, albeit one that may not be accomplished as effectively or efficiently as desired. Interviews with SQA personnel indicate that “Spot checking” and “completeness by expectation” are phrases that best describe the heuristics currently applied when assessing traces.

Adding non-functional requirements (NFRs) to a trace analysis provides a further challenge. Assurance staff are responsible for ensuring that all NFRs trace to all appropriate functional requirements (or FRs), and that there are no inappropriate or spurious traces (i.e., requirements that with certainty do not trace, or *anti-traces*). This is also referred to informally as ensuring the *completeness* and *correctness* of the NFR traces. What makes this difficult is that the degree or strength of a trace is not considered; it is a binary relationship, each NFR either traces or anti-traces to an FR. What exactly determines an appropriate degree is generally unspecified, but the intent is to only trace an NFR to an FR upon which it has an observable effect. We refer to the RTM sub-matrix of traces from NFRs to FRs simply as the NFR-FR matrix.

Since there may be missing traces, this is a highly effort-intensive activity because assurance personnel must check all possible traces, not just those already indicated in a traceability matrix. In addition, traceability assurance is a highly detail-oriented and information-intensive activity, unguided and tedious for humans to perform.

There are three major challenges to the manual assurance of NFR to FR traces: size, complexity, and effort/cost. We discuss each below.

Size

Consider a very small software requirements specification consisting of just 50 requirements, 20 of which are FRs and 30 of which are NFRs. There are $20 \times 30 = 600$ possible traces between the FRs and NFRs that may have to be assured. It is not optimal for analysts to assess 600 traces manually, but it is possible. In contrast, the Mars Reconnaissance Orbiter (MRO) had over 7500 requirements, a portion of which were NFRs. If we assume 7300 FRs and 200 NFRs, there could still be 1,460,000 traces to assure.

Complexity

NFR to FR trace assurance is a matching problem, central to graph theory, which can be modeled as a bi-partite graph (the tracing graph) on the two sets of requirements NFR and FR where an edge indicates that a given non-functional requirement affects the related functional requirement. Such bi-partite graphs are informationally equivalent to an NFR-FR matrix. The matching problem is relevant due to the fact that every NFR must trace to at least one FR and that the focus is on validating a “tracing” from the many “valid” combinations of tracings possible (that is, not all valid traces of an NFR-FR are expected to be relevant or of interest).

Even though the tracing graph is expected to be relatively sparse (generally each NFR traces only to a small percentage of FRs), assuring completeness requires examination to ensure that edges are valid and no edges are missing. This implies that the complete bi-partite graph $K(NF, F)$ with $|NF| * |F|$ edges must be reviewed to verify the trace/anti-trace relevancy. This requires $O(|F|^2)$ number of steps¹.

Part of the assurance process is determining the risky and non-risky areas, thus we cannot reduce the complexity by prioritizing or reducing the set of requirements to investigate (the “investigation set”) based solely on external risk or cost.

We have taken a somewhat simplified view of the problem. At JPL, requirements tend to be hierarchical: there is “flow-down” from one level to another (hence the terms upwards/downwards tracing). Thus if there is a trace at one level, this trace will flow-down to the requirements below it. This can significantly reduce the number of traces to be verified. However, in order to “depend” on this hierarchy, one must first validate that the trace is at the appropriate level. Hence, “layering” the requirements only partially reduces the complexity of the assurance problem.

¹ assuming $|F| > |NF|$.

Effort/cost

NFR to FR tracing assurance is a costly and effort consuming activity. Consider the MRO project with 7500 requirements. If we assume that there are 1,460,000 traces to assure and assume an average of 1 minute per trace audit (a highly optimistic estimate), then we expect $|7300| * |200| / 60 = 24,333$ person-hours of effort. With a 40-hr work week, it would take 11.7 people an entire year to complete the work. As a result, assurance personnel rarely perform exhaustive analysis. Rather, they become “familiar” with the requirements and use a variety of approaches to approximate a completeness check. A common approach is to “spot check” to rapidly identify potential problem areas and then to focus on these. Another popular approach is to only validate the existing traces and then prune and expand these. Assurance personnel will augment these approaches by considering related groups of requirements. For example, if there is a trace from a particular NFR to a FR, then it is often fruitful to look at the requirements that are similar or strongly related to that FR for traces.

All these approaches assume a sufficient familiarity with the entire set of requirements and rely heavily on the experience and domain knowledge of assurance personnel. Given this assumption, completeness is addressed by comparing a given trace to what traces are “expected” relative “not expected” in the particular system. Gaps in domain knowledge are unavoidable (a person cannot keep all knowledge of a system in their mind at one time), thus making it difficult to gauge the believability of a completeness audit.

It is clear that the above challenges point to the need for automated tool support. However, such support should not cause a significant change to existing assurance practices – otherwise, accustoming assurance personnel and other stakeholders to the new practices can substantially decrease near-term and intermediate-term efficiency.

Clearly automated traceability techniques could greatly assist in guiding assurance efforts such as NFR-FR tracing. For some types of traceability analysis, Hayes and her colleagues at the University of Kentucky have developed Information Retrieval (IR) based techniques and tools [10]. However, general-purpose approaches do not currently exist to assist with the types of completeness or correctness assessments described above.

3.2 Tools to Increase Effectiveness

At JPL there are three main areas that confidence in the sufficiency of manually performed assurance is an issue. These are: the analysis of risks to identify top-N risk lists with recommended risk scores; the

analysis of requirements to characterize them and identify requirements defects; and to analyze the large anomaly repositories to determine how (or whether) the frequencies and types of software anomalies are changing over time. The common element of these efforts is that each one seeks to assure the quality of a key activity in the software systems lifecycle. While the concept of assuring requirements is common practice, the concept of risk assurance in the software industry is not. For example, are the identified risks sufficiently comprehensive? Has anything important been forgotten? How frequently has the same mistake been made in the past? If a development organization is to improve it must learn from its past and apply that knowledge to the effective management of risk.

Leveson remarks that in modern complex systems, unsafe operations often result from insufficient risk analysis [11]. Here risk is defined as a combination of the likelihood of an accident and the severity of the potential consequences. More generally, Boehm defines software risk a potential for the development or product to have an unsatisfactory outcome to project stakeholders [12]. Unsatisfactory software project and system outcomes (e.g. problems and failures) due to insufficient risk analysis have been extensively reported and documented in the literature such as [8], [11], [12], and [13].

Insufficient risk analysis includes errors such as failure to identify a significant risk (omission), incorrect risk specification, redundant risks, vague or poorly specified risks, risks without mitigation options, and so forth. Such errors contributed to the demise of NASA's Mars Climate Orbiter (MCO) launched in December 1998. It was discovered that the developers of the Ground navigation software used English Units while the flight software developers used Metric Units. The discrepancy in units biased trajectory calculations in route and set MCO too close to Mars during its insertion into orbit where MCO went silent and was lost. The specific problem of incompatible units between system components that led to the MCO mishap was a well-known and documented risk on previous projects, yet the development teams still failed to identify it.

MCO is one of many examples of insufficient risk analysis. Given the rapidly increasing costs and consequences of software errors, understandably there is rising interest in ensuring sufficient risk analysis is performed. This has led organizations such as JPL to employ risk assurance practices. Risk assurance is the use of quality assessment techniques such as verification and validation (V&V) to ensure that sufficient risk analysis has been performed (i.e., the risk analysis is as correct, complete, clear, and actionable as possible).

A primary risk assurance activity is auditing risk documentation. This is generally performed by assurance personnel who read through documents to identify risk analysis errors. To help encourage an independent, objective and “un-blinded” perspective, ideally the auditor has considerable experience across a diversity of projects and is not directly involved in the development. However auditing is fraught with a number of value-degrading challenges relating to its cost-effectiveness, reliability, confidence in results, and the practicality in making use of historical data. One particularly troublesome challenge has been providing confidence in the completeness of a risk analysis i.e. the degree of certainty that all significant risks have been identified. There are two major problems here – (1) accounting for previously unknown risks, or the so-called unknown-unknown's, and (2) blindness or bias against recognizing known risks or risk patterns, sometimes referred to as “unknown knowns.” Several techniques have been developed to deal with these problems. Generally these involve risk identification audit checklists and guidelines based on historical risk experience such as “top-10” risk lists, risk area taxonomies, and risk analysis processes. However, generating these aids can be cumbersome and costly, and it can be difficult to keep them current. Furthermore, when manually generated, these aids too are subject to risk assurance challenges such as completeness.

4. Example Use of Tools for SA

Text-Mining Support for Trace Assurance

We now describe an approach for using text mining to support trace assurance. For clarity, we emphasize a few things up front. First, the method does not aim to generate an RTM. Indeed, the method requires an existing RTM as input i.e., the RTM to be assured. Second, the aim of the method is not to automate the detection of or assure FRs or NFRs. However, a by-product of tracing assurance can help with this. Last, the method is not designed to detect vague or poorly stated requirements.

With the above in mind, we state that a successful method for automated support of trace assurance at JPL would meet the following vital objectives: 1) Must be compatible with the way assurance personnel address trace assurance (e.g., “expected” and “unexpected” traces based on prior experience, domain knowledge, and familiarity with the requirements); 2) Must be empirically driven, adjusting to the quality of the requirements specification (e.g., vaguely specified requirements should result in more conservative automated results) and adjusting to the quality of a given RTM; 3) Must be easily implemented and

integrate with existing requirements managers (e.g., DOORS, RequisitePro, etc.); 4) Must have an established theoretical foundation; Must be practical to use (e.g., low-learning curve) and provide meaningful guidance; 5) Must be based on open methods and technologies (assurance cannot be based on black-box solutions); 6) Must reduce overall effort, increase efficiency of effort, and increase confidence in results. Note that Hayes, Dekhtyar, Sundaram, and Howard have posited essential requirements for any requirements tracing tool as examined from the user’s perspective. Objective (5) ties to their Usability sub requirement (of Believability) [15].

We now describe an approach to meet the above objectives in a series of concepts and examples given below.

Trace investigation sets

The fundamental challenge for trace assurance is effectively managing the verification of a large number of traces and anti-traces. A natural means of addressing this is to employ a divide and conquer strategy that partitions these sets into more manageable *investigation subsets* based on meaningful rules and empirical properties of the requirements. For example, an obvious rule is “each NFR must trace to at least one FR” and the resulting investigation set (a subset of the traceability graph that is under assessment) would simply be all those NFRs without traces. Determining rules and properties and making them actionable (e.g., if an NFR has no traces then it must be removed or be reported as having missing traces) helps address objective (5).

Aligning rules and empirical properties with assurance personnel’s a priori knowledge helps meet objective (1). Partitioning the assurance tasks into investigation sets greatly reduces the complexity and narrows the focus of the assurance effort. Furthermore, each investigation set implies particular assurance activities (e.g., look for a missing trace), “guiding” the effort to be more efficient and effective thereby helping to meet objective (7). If some investigation sets have a low risk (when appropriately defined) of its elements being incorrectly determined (as being in the set, for example), then such sets can be eliminated or “lightly” assured further helping to satisfy objective (7).

To illustrate, assume that as we examine the traceability matrix to be assured, we notice an observable property between a pair of requirements called “high-similarity” (perhaps each requirement contains many of the same words, e.g.) which we believe is highly correlated (but this is not certain) with requirements that have been associated to each other in the traceability matrix (meaning that it is highly

correlated with our notion of *trace*). We note that absence of this property between two requirements does not imply that they *anti-trace*. The absence of “high-similarity” provides no information, whereas the presence of “high-similarity” appears to provide evidence of *trace*.

With this idea in mind, let us examine the notion of trace investigation sets further. In the previous example, we discussed the *trace* set or T. Fig. 1 shows T in the top left; all NFRs trace to at least one FR, but do not trace to every FR. By simply examining the edges that do not exist in T, we obtain the *anti-trace* or AT (shown in the top middle section of Fig. 1). Based on T and AT and our notion of “high similarity” (called HT), we can generate four trace investigation sets L, M, F, N (see Figure 1).

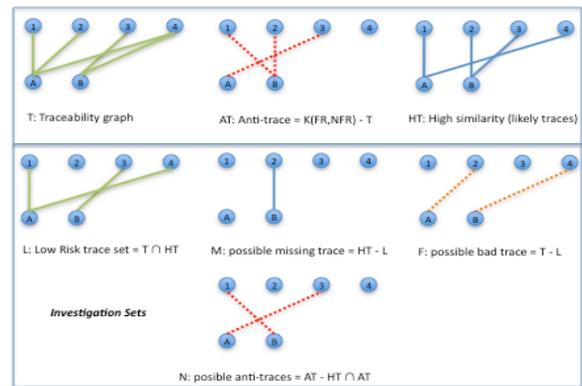


Figure 1 - Partitioning requirements into investigation sets

We consider traces in the investigation set L ($T \cap HT$, as shown in Figure 1) to be *low risk* as they have two independent sources corroborating the trace (they were in the RTM under assessment and we observed the high-similarity property). Items in M are at high-risk of being *possible omissions* from T as we expect requirement pairs with high-similarity to trace (but not the converse). Items in M need to be carefully checked to see if they are indeed traces. We have little information about items in F, but as they did not have high-similarity, they should be checked first as possible bad traces (also called false positives or errors of *commission*). Last, there is little to say about items in N other than that they do not have high-similarity and they did not trace, so we first try to verify that they are anti-traces.

The example just presented, while simplified and overly generic, is in essence our method. The complexity reduction, work avoided (assuming we do not check the low-risk set L), and increased assurance efficiency is self-evident. Increased confidence in the assurance results is in part self-evident, but also

depends greatly on our confidence in the correlation of the high-similarity property and requirements that trace.

Finding properties that are practical to observe and in which experienced assurance personnel have high confidence is a key component of our method. Also, finding properties that determine both *inclusion* and *exclusion* of elements into an investigation set is essential to effectively addressing the trace completeness problem; this is discussed next.

Case study: NFR PROMISE Project 10

The following case study demonstrates how effective the method was compared to a manual trace assurance. The case study is taken from the PROMISE NFR data set [14], and considers Project 10 (P10), the requirements specification for an online version of a game like “Battleship.” P10 has 15 NFRs and 38 FRs and a manual NFR-FR requirements trace was generated and is illustrated in Figure 2 to provide an initial feel for the complexity of the trace assurance task at hand.

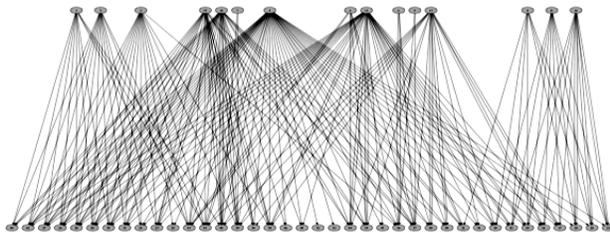


Figure 2 – P10 NFR-FR tracability graph

Using the above as an input to the tool, the investigation sets are generated. It is straightforward to express the rules as list (matrix) index selectors in R. Figure 3 shows snippets of two different ways to report the investigation sets (you are not expected to read these tables, they are illustrative only). The report on the left provides a compact view while the report on the right uses the investigation sets to annotate the RTM with color to help alert assurance staff of potential issues.

R7	R8	R9	R10	R11	R12	R13	R14	...	11	12	13	...	
11 25	14 52	4 40	3 42	1 16	2 26	1 31	1 26	...	21	R14	R11	R11	...
11 50		4 41	4 43	1 17	7 39	3 25	1 27	...	22	R9	R12	R13	...
14 22		4 42	5 36	1 18	8 16	7 38	1 28	...	23	R11	R12	R11	...
		5 38	5 37	1 19	8 17	7 39	1 29	...	24	R10	R12	R11	...
		5 39	7 40	1 20	8 18	7 40	1 30
		5 45	11 18	1 21	8 19	4 24	1 40	
		5 46	11 20	1 22	8 20	4 25	1 41	
		7 41	11 52	1 23	8 23	4 26	1 42	
		7 42	11 53	1 24	8 24	4 27	1 43	
		7 43		1 25	8 25	4 28	1 48	
		11 22		1 32	8 26	4 29	2 23	

Figure 3 - P10 NFR-FR partition investigation sets report examples

These results are used to perform an independent assessment of the investigation set accuracy. One author went through each set element-by-element

assessing the veracity for being in that set (except “no info” sets which make no claims about the requirements). Results are listed in Table 1 where each entry $x \setminus y$ is read, “x were found correct from y elements.” An $(a \sim b)$ entry means the assessor was unsure about $b \sim a$ of the elements. These could be correct, but there is some doubt.

Table 1 – Accuracy of investigation sets

R5	R7	R8	R9	R10	R11	R12
1\2	3\3	1\1	(8~11)\11	(7~9)\9	330\330	(91~101)\109

Our independent verification gives the investigation sets 95%-98% accuracy. The verification effort took 188 minutes. This is not surprising given that the assessor had to review all but 107 of the 507 potential traces and anti-traces.

Next, we had a JPL assurance staff member perform a fully manual P10 trace assurance by means usual to them. Table 2 compares the results of this effort with the author’s assessment guided by the investigation sets generated.

Table 2 – Comparison of manual and investigation sets

	Effort	Missing Traces	False Traces	Duplicates	Verified Traces	Verified Anti-Traces
Manual	227 mins	5	39	2	131	395
Inv-set	94 mins	11	99	2	159	301

Verified trace/anti-trace means that a trace/anti-trace was reviewed and found correct. For the investigation set based assurance, elements in the “low risk” sets R10 and R11 were only “lightly” reviewed to achieve the verification. Here very few elements in the low risk sets were found to be incorrect. In comparison with the manual trace assurance, the set based assurance effort took 58% less effort, found 120% more “high risk” missing traces, and 154% more spurious traces (not so risky, but resource wasteful). The verification rates were comparable, but since any problem found reduces the number of verified elements, it makes little sense to compare the increase or decrease of these. The author’s experience in performing the set guided assurance felt more focused and less tedious than the manual approach. While this is wholly subjective, consider if the elements in the low-risk sets were not reviewed at all. This would remove 59% of the trace/anti-trace review size, and assuming a constant effort per trace/anti-trace review, would result in a decrease in 41% of the effort. Given that in this evaluation we saw a 58% decrease in effort, there is likely further efficiencies present than only reducing the number of items to review (and recall that the author did not entirely eliminate review of the low-

risk elements). Neither the author nor the assessor was familiar with P10 beforehand.

Manual trace assurance was performed on 10 of the 15 projects from the NFR PROMISE data set. These, along with the complete details for the P10 evaluation above will be made available there for review.

5. Current Use of Tools for SA

Several dozen tools having potential applicability to SA are currently being surveyed as part of an ongoing task at JPL to determine each one's applicability or support in different areas of assurance. SA areas identified at JPL include architecture, code, contractor, cost, delivery, product, project, reliability, requirements, resource, risk, safety, schedule, security, test assurance, and assurance management.

Assurance management addresses the issue of knowing whether the appropriate set of and intensity of assurance activities is being done by considering the optimal cost/benefit/risk balance, and assesses whether the project is health from the SQA perspective. Requirements assurance involves peer reviews of software requirements, ensuring completeness of requirements, as well as performing requirements tracing. Reliability assurance looks at the level of risk in the software system and the likelihood of potential failures by evaluating and closing PFRs, performing reliability analyses, and assessing the problem reporting process. A related area is risk assurance, which deals more specifically with risk tracking, monitoring, and assuring the risk management. Safety assurance involves performing and reviewing software safety analyses, assessing the safety compliance, and verifying the safety of the software design.

A preliminary assessment of SA tools and how they apply to each assurance area has been done at JPL, rating the tools on a 5-point scale from "provides explicit support" to "inconsistent with/incompatible or contraindicated." A sample of SA tools and their applicability is summarized in Table 33.

The Constructive Cost Model (COCOMO) is a software cost estimation model, which computes software development effort as a function of the size of the project. COCOMO is especially useful in cost assurance and also contributes significantly to project and schedule assurance, which are directly related to the amount of effort required. The Architecture Analysis and Design Language (AADL) is a tool that can be used to analyze system designs prior to development, as well as support a model-driven approach throughout the life of the system. AADL explicitly supports architecture, reliability and resource assurance. JIRA is a platform used by the development

team that can track bugs and tasks and monitor activity for a project, which is essential to risk assurance. CASRE is a quantitative assessment tool that is used to estimate and forecast the reliability of software systems during tests and operations. Its strength, therefore, falls in the area of reliability assurance. ASCE by Adelard is a tool used to develop, manage and communicate safety cases, and is especially applicable to risk, safety and security assurance. Coverity is a static code analyzer used to find bugs and vulnerabilities in source code. While static code analyzers are very useful in code assurance, they can indirectly apply to safety, risk, and reliability assurance, which are all related to ensuring the success of the source code.

Table 3 - SA tool support or applicability

SA Tool	High support for assurance area	No support for assurance area
COCOMO	Cost, project, schedule	Security
AADL	Architecture, reliability, resource	Cost, process, project, schedule, assurance mgmt.
JIRA	Risk	Cost
CASRE	Reliability	Resource, schedule
ASCE	Risk, safety, security	none
Coverity	Code	Cost, process, requirements, schedule, assurance mgmt.

6. Evaluating Use of Tools for SA

Evaluating SA tools is just as important as important as using the tools themselves. Tool evaluation is essential to knowing which is the best tool for a given assurance task.

6.1 Tool Evaluation Criteria

The evaluation criteria focus on the following aspects of tool acquisition and use:

Applicability: Tools are evaluated on the basis of how well they support assurance activities (e.g. audit and inspection of software artifacts) or assurance process (e.g. findings tracking and reporting). Applicability criteria are specialized to the type of artifact being analyzed – for example, requirements analysis tools are evaluated according to the extent to which they can trace requirements forward and backward, how well they are able to identify ambiguous and inconsistent requirements, and the extent to which they can identify potentially incomplete specifications (e.g. missing functionality, missing inputs or outputs).

Effectiveness: Tools are also evaluated on the basis of how well they perform their claimed capabilities. To the greatest possible extent, these are quantitative criteria. For example, quantitative criteria for evaluating static code analyzers include the proportions of false positives and false negatives, and hit rates such as the proportion of array boundary violations flagged by the analyzer that are actual violations in the implemented systems, or the number of flagged null pointer dereferences that are actual defects in the real system.

Tool Availability: An additional set of evaluation criteria focuses on a tool's availability. These criteria include the type of supplier from which the tool is available (commercial domestic, commercial foreign, NASA-developed, other U.S. Government, university, open-source)², length of time for which the tool has been available (i.e., when was the first version released?), length of time for which the supplier has been in business, and tool cost.

Usability: Criteria for tool usability focus on the effort required by users to learn how to operate the tool; whether the tool's documentation accurately describes its setup and operation, and does so at a sufficiently detailed level; whether the tool runs in multiple development environments (e.g., Eclipse [16]) and under multiple operating systems; and the degree to which operating the tool in one environment is similar to operating it in another; the tool's technology readiness level (TRL). The evaluation criteria also examine the tool's user interface to determine the extent to which it conforms to the published standards for a specific environment (e.g., if it runs under Windows, does it comply with the published standards for the appearance of Windows applications?) as well as the extent to which it complies with accepted guidelines and recommendations on the appearance and behavior of user interfaces (e.g., Shneiderman's work [17])

Relationship To other Tools: In addition to criteria for evaluating tools in isolation, we are developing criteria for evaluating the way in which tools are related to each other. The collaborative use of tools may have synergistic or antagonistic effects on the visibility that assurance engineers and developers have into the quality of the system and the efficiency and effectiveness of assurance activities. For example, the

combined use of a problem tracking system, a version control system, and a source code structural analyzer can be used as input to an analysis that will predict the number of defects that have been inserted into the source code for individual functions or methods within a software system [18].

6.2 A specification for the functionality, behavior, and structure of the tool evaluation framework.

Putting the evaluation framework into practice requires specification for a framework enabling assurance tool users at JPL and other NASA centers to collaborate in the evaluation and selection of assurance tools for specific projects or identified institutional needs. The specification includes descriptions of the following:

(i) The information to be managed by the framework, including descriptions of individual tools, tool evaluation results, and descriptions of assurance tool needs that are claimed to be unmet by currently available tools.

(ii) A comprehensive set of quantitative tool metrics useful for cost-benefit and tool trade-off evaluations comparing tools and manual approaches within the assurance process. Some example metrics include:

- 1) Scalability ratio = {max amount handleable with tool / max amount handleable manually}. The amount of information that can be handled manually can be estimated by analyzing workproducts produced by the SA staff over a number of years over multiple projects.
- 2) Assurance productivity efficiency = {average amount assured per function point with tool / average amount assured per function point manual}
- 3) Accuracy ratio = {average number errors with tool / average number error manual}
- 4) Average accuracy = {average errors with tool}
- 5) Accuracy variance = {variance of errors with tool}
- 6) Coverage fraction [0-1] = {amount tool covers / total amount}
- 7) In-processes efficiency with respect to COCOMO schedule factors (SF) and effort multipliers (EM) = {COCOMO estimate with tool(s) / COCOMO effort without tool(s)}
- 8) Tool efficiency with respect to manual = {effort with tool / effort manual}
- 9) Integrability coefficient [0-1] = {fraction of output compatible with assurance process}

² This attribute of the tool vendor can influence the stability and long-term availability of the tool, although precise relationships are not known at this time.

(iii) Methods by which users interact with the framework and each other to share and analyze information.

(iv) Analyses that users can request be performed by the framework. These include techniques such as traditional statistical analysis (e.g., trending) for structured data, as well as advanced techniques such as data mining, natural language processing, and unsupervised learning (for discovering patterns) in unstructured data such as natural language text.

6.3 Tool Evaluation at JPL

The tool evaluation criteria and evaluation framework described above are currently being developed as part of on-going work at JPL to improve the use of tools in the JPL Software Assurance organization as well as other assurance organizations

these tools are related to each other, and how they are most effectively used together within a given assurance effort. The goal of this work is to create a curated tool evaluation framework that functions as an accurate and effective information exchange between the assurance communities at the various NASA centers. Work to date has focused on:

- Working with assurance personnel to identify the types of activities they perform.
- Developing a list of candidate tools that might support each of the different types of assurance activities.
- Designing a survey that can be used to evaluate the applicability of tools to different assurance areas.
- For each candidate tool, assessing its utility for each of the assurance areas that have been defined.

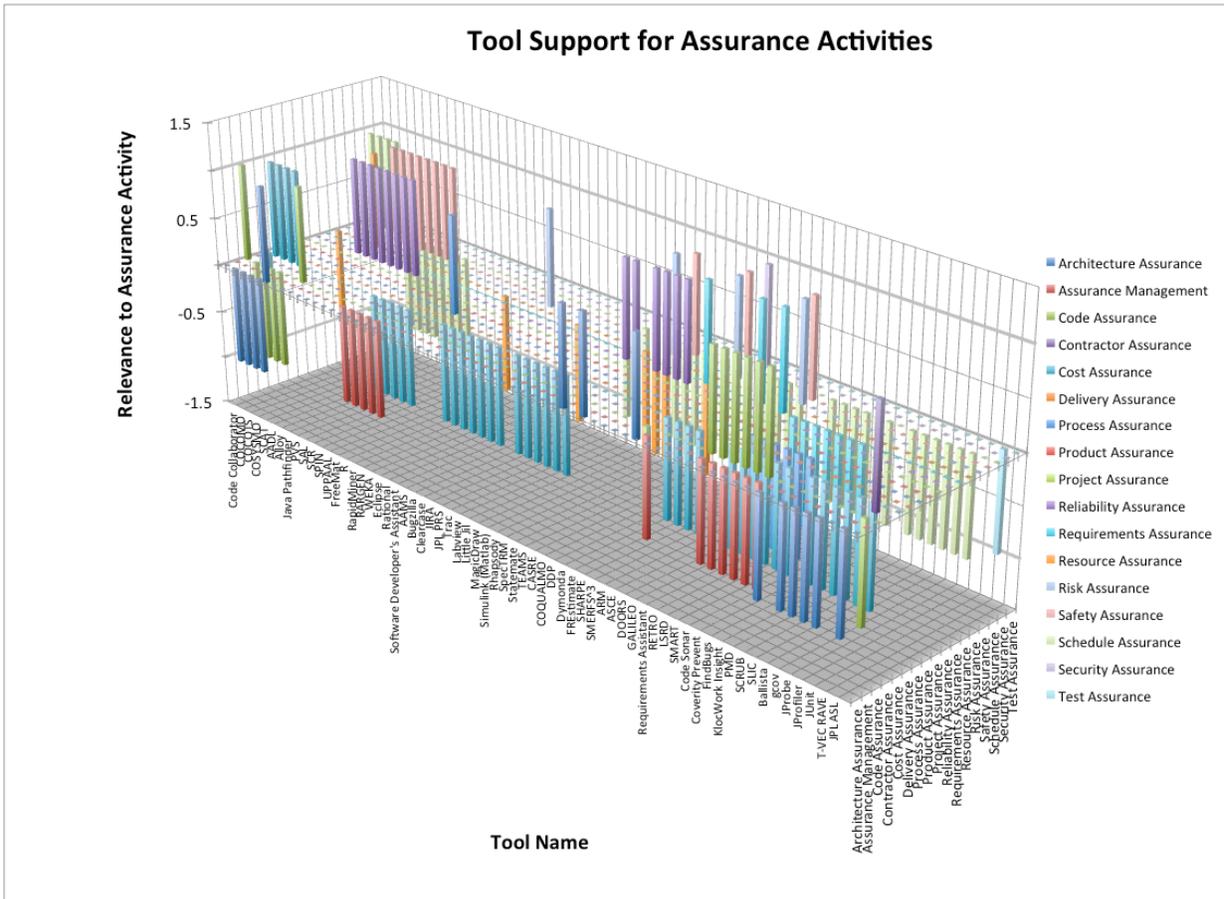


Figure 4 - Evaluations of Tool Applicability to Assurance Areas

within NASA. As mentioned in Section 1, the underutilization of tools for assurance support is partially associated with perceived high overhead in identifying and evaluating potentially useful tools, little understanding on the part of potential users of how

Figure 4 shows an initial evaluation of each candidate tool's utility for each assurance area. Only the extremities of the 5-point scale mentioned in Section 5 appear on this plot. A value of 1 indicates

“provides explicit support”, while -1 indicates “inconsistent with/incompatible or contraindicated.” The inner 3 values of the scale have been collapsed to “0.” This evaluation appears to show that an individual tool is appropriate to only a small number of assurance activity types, meaning it may be important for assurance staff to know how different tools work together if they are interested in using tool support for more than a small number of the tasks they perform. Further work with members of the wider NASA assurance community will provide additional information.

It is important to note that these values cannot be compared with each other except by ordering; it is meaningless to analyze their ratios or the distance between their absolute values.

Evaluations of the tools were collected from a handful of software assurance researchers within NASA and will contribute to those evaluations shown in Figure 2. Analysis of the data collected will be done formally using hypothesis testing to determine whether a specific tool supports an each assurance area. For example, accepting the hypothesis “tool *x* received mostly evaluations of ‘provides explicit support’ for assurance area *y*” would reveal strong support for the assurance area. In order to perform the tests, sufficient data is needed; with 5 categories to evaluate the tool, at least 5 evaluations are needed, but 7 or 8 will provide more confidence. More informally, analysis can be done by looking at the distribution of the evaluations for each tool and assurance area. The reliability of the conclusions drawn from analyzing the evaluations will depend on the amount of data that is collected.

The next steps in this work include:

- Making the survey available to members of the wider NASA assurance communities, and working with members of these communities to elicit information from them.
- Maintaining an affiliated curated repository of information on each candidate tool (e.g., a set of wiki pages) that will be available to members of the NASA assurance community. Community members will be able to retrieve information about a particular tool, and will also be able to contribute to on-going discussions about the value of the tool for specific assurance activities. Our plan is that the curator of the repository will work to keep the information in the repository current, and will also moderate the discussions to ensure that as little extraneous information is introduced into the repository as possible.

Although this work does not address all of the issues related to the underutilization of tools (e.g., time required to learn a tool), it does address some of the

major issues identified by SA personnel (e.g, lack of information on what tools are available, what assurance areas they support, and how they interact). Our hope is that the repository will make it easier for SA personnel to make more effective use of tools, thereby increasing the value they add to the projects they support.

Acknowledgement

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology. The research was sponsored by the National Aeronautics and Space Administration’s Office of Safety and Mission Assurance Software Assurance Research Program. This task is managed locally by JPL’s Assurance Technology Program Office.

References

- [1] NASA’s Office of Safety and Mission Assurance Software Assurance Research Program, Software Assurance Symposium (SAS), Sep 2009, http://www.nasa.gov/centers/ivv/pdf/386081main_SAS_2009_schedules_090911.pdf, viewed June 17, 2012.
- [2] National Aeronautics and Space Administration, "Software Assurance Standard," NASA-Std-8739.8 w/Change 1, July, 2004, p. 8.
- [3] Committee on National Security Systems, "National Information Assurance (IA) Glossary," CNSS Instruction No. 4009, April 2010, p. 69.
- [4] B. W. Boehm, "Software Engineering Economics," Prentice-Hall, 1981
- [5] V. Basili, L. Briand, S. Condon, Yong-Mi Kim, W. L. Melo, and J. D. Valen, "Understanding and Predicting the Process of Software Maintenance Releases," proceedings of 18th International Conference on Software Engineering, Berlin, Mar 1996, pp. 464-474.
- [6] R. E. Park, W. B. Goethert, and W. A. Florac, "Goal-Driven Software Measurement—A Guidebook," Software Engineering Institute, CMU/SEI-96-HB-002, Aug. 1996.
- [7] "The Economic Impacts of Inadequate Infrastructure for Software Testing," NIST, RTI Project 7007.011.
- [8] R. Lutz, C. Mikulski, "Operational Anomalies as a Cause of Safety-Critical Requirements Evolution," The Journal of Systems and Software, vol. 65:2, Feb. 2003, pp. 155-61.
- [9] R. Lutz, C. Mikulski, "Requirements Discovery during the Testing of Safety-Critical Software," proceedings of 25th Int'l Conf on Software Engineering, Portland, OR, 2003, pp. 578-585.
- [10] J. Hayes, A. Dekhtyar, S. Sundaram, A. Holbrook, S. Vadlamudi, A. April, "Requirements Tracing On target

- (RETRO): Improving Software Maintenance through Traceability Recovery,” *Innovations in Systems and Software Engineering: A NASA Journal (ISSE)* 3(3): 193-202 (2007).
- [11] N. Leveson, *Safeware System Safety And Computers*, Addison-Wesley, 1995.
- [12] B. W. Boehm, “Software risk management: Principles and practice”. *IEEE Software*, 8(1): 32-41.
- [13] D. Carney, E. Morris, and P. Place, “Identifying Commercial Off-the-Shelf (COTS) Product Risks: The COTS Usage Risk Evaluation”, TECHNICAL REPORT. CMU/SEI- 2003-TR- 023. September 2003.
- [14] “Predictor Models in Software Engineering (Promise) Software Engineering Repository.” <http://promise.site.uottawa.ca/SERepository>
- [15] J. H. Hayes, A. Dekhtyar, S. Sundaram, S. Howard, “Helping Analysts Trace Requirements: An Objective Look,” in *Proc. of IEEE International Conference on Requirements Engineering*, Sep. 2004, pp. 249-261.
- [16] The Eclipse Foundation, “Eclipse-The Eclipse Foundation open source community website,” <http://www.eclipse.org/>, viewed June 18, 2012.
- [17] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, “Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition),” Addison Wesley, 2009.
- [18] A. Nikora, J. Munson, "Building High-Quality Software Fault Predictors", *Journal of Software Practice and Experience*, 2006, vol 36, no. 9, May, 2006, pp. 949-969, doi:10.1002/spe.737.