A Discussion of the Software Quality Assurance Role

Ronald Kirk Kandt

Jet Propulsion Laboratory, California Institute of Technology

4800 Oak Grove Drive

Pasadena, California 91109

Phone: 818-393-0907

E-mail: ronald.k.kandt@jpl.nasa.gov

Abstract

The basic idea underlying this paper is that the conventional understanding of the role of a Software Quality Assurance (SQA) engineer is unduly limited. This is because few have asked who the customers of a SQA engineer are. Once you do this, you can better define what tasks a SQA engineer should perform, as well as identify the knowledge and skills that such a person should have. The consequence of doing this is that a SQA engineer can provide greater value to his or her customers. It is the position of this paper that a SQA engineer providing significant value to his or her customers must not only assume the role of an auditor, but also that of a software and systems engineer. This is because software engineers and their managers particularly value contributions that directly impact products and their development. These ideas are summarized as lessons learned, based on my experience at Jet Propulsion Laboratory (JPL).

Keywords

Lessons learned, assurance job skills, experience, and responsibilities

1. Introduction

JPL is a Federally Funded Research and Development Center managed by the California Institute of Technology under contract to the National Aeronautics and Space Administration (NASA). The primary mission of JPL is to explore and observe the farthest reaches of the solar system. To do this, it develops numerous spacecraft, each controlled by software that resides on these spacecraft. Several outside suppliers and engineering organizations at JPL develop this software and several people, called SQA engineers, assure its quality.

The purpose of this paper is to describe the role of a SQA engineer. The specific objectives of this paper are to identify the customers of a SQA engineer, the activities a SQA engineer must perform, and the requisite knowledge and skills of a SQA engineer. JPL is the organization that I use to illustrate these objectives. A statement of the SQA engineering role for JPL is proposed, as are several lessons that the reader can use to tailor this role statement for use within their own organization.

2. The Role of a SQA Engineer

When trying to define the role of a SQA engineer, I first examined books (Chrissis et al, 2007; Schulmeyer, 2008) and the ACM, IEEE, and SEI web sites. To my surprise, only one of these sources defined the role of a SQA engineer; the others only provided descriptions of SQA activities and metrics. The *Handbook of Software Quality Assurance* (Schulmeyer, 2008), for example, provided a sample job description that specified a SQA engineer must have 4 years of software related experience, of which 1 year should be in SQA, and a B.S. in computer science, information technology, or a related technical discipline. It also stated that the duties of a SQA engineer were to: (1) participate in software design reviews, testing, configuration control, problem reporting and resolution, and change control; (2) audit, monitor, evaluate, and report on software subcontractor activities; (3) produce write-ups and estimates for SQA activities; and (4) interface with software engineering, software configuration management and the software process organizations.

JPL, on the other hand, provides the following job description for a SQA engineer: "A SQA engineer: (1) plans and executes a systematic set of activities to ensure that software lifecycle processes and products conform to applicable requirements, standards, and procedures; (2) ensures that planned and implemented process and product standards conform to applicable requirements, and are appropriate for the risk posture of the project; (3) ensures that planned corrective actions meet acceptable reliability standards; and (4) ensures that safety-critical software is identified and tracked, and that risks are mitigated to ensure safe operation of the software." I concluded from my examinations that there is no common definition of a SQA engineer and there are significant omissions in each definition.

3. The Customers of a SQA Engineer

Every SQA engineer has customers. These customers generally are project managers, line managers, process owners, and funding agents (i.e., entities that pay you). Customers may be different for one organization than for another, but they tend to be very similar among organizations. The customers at JPL include people from the assurance organization, one or more software engineering organizations, and the Software Process Engineering Group (SPEG). To simplify the following discussion, I assume only one software engineering organization and one SQA engineer works on a project.

Key personnel within an engineering organization at JPL are the project manager and the software manager. The project manager directly or indirectly authorizes and funds all work on the project. Similarly, the software manager authorizes, funds, and monitors all software engineering work on the project, provides information to the SQA engineer, reviews the reports that the SQA engineer produces, and reports software development status to the project manager.

Key personnel within the assurance organization are the mission assurance manager, SQA engineer, and SQA manager. The mission assurance manager authorizes and funds all assurance activities on a project, reviews the work of the SQA engineer, and reports his or her concerns to the project manager. The mission assurance manager has the final say in whether a spacecraft is ready for launch. The SQA engineer performs all SQA activities for the project and reports results to the mission assurance manager, the software engineering manager, and the SQA manager. The SQA manager summarizes the concerns of the SQA engineer and reports them to his manager. Higher-level quality assurance managers repeat this process until the manager of the mission assurance organization, who is the supervisor of the mission assurance manager, hears the summarized concerns.

The remaining key customer of a SQA engineer is a project representative of the SEPG, which JPL calls a software process engineer. The software process engineer receives records from a SQA engineer that the SEPG uses to improve the institutional standard software development processes, which it maintains. Figure 1 illustrates all of these relationships.

Place Figure 1 about here.

4. The Tasks a SQA Engineer Performs

What should a SQA engineer do? A SQA engineer should perform two basic types of tasks: audits and assessments. An *audit* is a systematic review of artifacts – typically records and documents – to verify their accuracy. For example, if a release description document stated that only the files f_1, f_2, $\cdots$ f_n comprised the entire release then an audit would ensure that those and only those files comprise the release. An *assessment*, on the other hand, is the evaluation of the quality of artifacts or activities to determine their adequacy or value and to identify their associated risks. SQA engineers at JPL audit software deliveries and assess development processes and various work products,

including the delivered products. Recently, SQA engineers at JPL have begun to assess managerial and technical decisions.

4.1. Standard Activities

JPL uses standard processes, as required by CMMI Level 3 (Chrissis et al, 2007). These processes describe the activities that the SQA organization performs. These activities include, for example, metrics collection and process tailoring. However, the key activities that SQA engineers at JPL perform for projects are described below.

- Software Delivery Audits For each software delivery, a SQA engineer performs a software delivery audit that satisfies criteria typically required by physical and functional configuration audits. More specifically, a software assurance engineer ensures that requirements have been allocated to the delivery and that they have been verified. In addition, a SQA engineer verifies that delivered documentation is useful and accurate and that open anomaly reports and action items have suitable action plans.

- Compliance Assessments A SQA engineer assesses software management plans for compliance with institutional software process and product requirements. If deviations occur, the SQA engineer writes findings and communicates them to members of the engineering and quality assurance organizations and project management. For each requirement deviation, the SQA engineer identifies the risk of the deviation as well as various mitigation strategies. After each compliance assessment, a project either revises its plans to satisfy the requirements or writes the necessary waivers, including rationales for the waivers that require approval by the appropriate process owner.

- Work Product Assessments A SQA engineer evaluates several types of work products. Key work products include those that define requirements and designs, as well as the code that results from them. Each assessment of requirements ensures that the number of defined nonfunctional requirements is adequate and each requirement is succinct, unambiguous, and verifiable. Further, each assessment ensures that each requirement includes a sound rationale, a method of verification, traces to higher-level requirements, and allocations to one or more software components.

  Currently, there are no institutional requirements for assessing the quality of a software architecture. In place of such a standard, a SQA Engineer uses the Architecture Tradeoff Analysis Method (ATAM) (Clements et al,

2001) to evaluate software architectures. Each assessment of a software architecture determines whether the architecture exhibits low coupling and appropriate segmentation and layering. In addition, an assessment ensures that the software architecture identifies the interfaces between the components of the architecture, as well as the method of communication among these components and the data that each component accesses, including the method of access. Similarly, during assessments of detailed designs, a SQA engineer focuses primarily on determining whether someone has assigned a cohesive set of responsibilities to a component, whether a component is loosely coupled with the rest of the system, and whether the implemented data structures and algorithms are reasonable.

When examining code, a SQA engineer attempts to verify that it satisfies institutional coding standards. Significant concerns include buffer overruns, uninitialized variables, redundancies in the code, incorrect bit manipulation, improper handling of exceptions, and incorrect or inadequate use of semaphores. A JPL SQA engineer focuses on these issues because they pose the greatest risk to the software that JPL develops.

- Process Area Assessments A SQA engineer periodically assesses the degree to which a project follows its software management plans. A set of heuristics based on the results of prior assessments and other events normally control the periodicity of these assessments.

- Milestone Review Assessments The institution defines requirements that identify what is supposed to occur at each milestone review (e.g., Preliminary Design Review and Critical Design Review). These requirements govern the material that project team members present at each review, as well as how to disposition action items that arise from each milestone review. A SQA engineer ensures that the material that is presented at a milestone review satisfies the requirements of the institution and that the resulting action items are properly dispositioned.

4.2. New Activities

Software managers and engineers make many decisions that underlie core software engineering and management activities (e.g., architectural design). Before these decisions are made, these people should conduct appropriate analyses. Afterwards, these analyses should be reviewed to determine that the final decisions are appropriate, as well as to ensure that the associated risks of these decisions have been appropriately quantified. Although SQA engineers at JPL seldom make, review, or approve such decisions as part of either a process or a product assessment, they

should do so. As a consequence, the SQA organization at JPL has initiated the creation of a catalog of critical decisions that it expects software teams to make. This catalog identifies the expected alternatives for each decision and the associated risks for each alternative. The creation of such a catalog is important because the root cause of many JPL mission failures is bad decisions caused by inadequate or omitted analysis (Kandt, 2010). In addition, decisions lacking proper analysis sometimes lead to significant or unexpected increases in engineering costs. Decision assessment is being discussed in the following sections to illustrate the importance of assessing project decisions.

4.2.1. Managerial Decision Assessments

The decisions that managers make often have a significant impact on work. The following description identifies the logical computing environment of a typical software team at JPL and how it has chosen to administer this environment. It is used to illustrate why managerial decisions should be thoroughly analyzed and why SQA engineers should assess those decisions.

The software team uses numerous blade servers and test-beds that are connected to a variety of laptops and desktop computers. The blade servers, in turn, are connected to institutional storage servers and are used to store software. Documentation, on the other hand, is stored in an institutional documentation repository that is accessible using any machine on the internal network through a web browser.

This environment is maintained by several different organizations. The desktops and laptops are administered by one organization, whereas another organization administers the blade servers. The institutional storage servers are maintained by yet another organization, which differs from the maintainers of the documentation repository. Finally, one last organization maintains the internal computer network. So, there are 5 separate organizations that the team must interface with to maintain its computing infrastructure.

During software development, the team encountered several problems. For example, user access to machines was denied, system software was inadvertently changed, and files periodically, and briefly, appeared to not exist. Performing thorough decision analysis most likely would have eliminated these problems. For example, consider whether the selected infrastructure could have supported the desired method for maintaining and releasing artifacts. For the software team of this example, it could not because the design artifacts and test results were maintained in

the document repository and there was no automated way to associate specific versions of those artifacts with a specific release of the software system under development.

Similarly, one should analyze whether the chosen software development tools would work within the network topology. This would require an understanding of the network topology and an analysis of whether its latency adversely affected the performance of the software development tools. Finally, the software team should have performed an analysis of whether being dependent on so much external infrastructure and so many organizations posed an acceptable level of risk. Although these support organizations individually provided value to the project, the risk associated with this infrastructure, including the use of five separate support organizations, outweighs the value provided by the infrastructure and the organizations supporting it. Hence, a software team should consider whether the hardware and software of the infrastructure is reliable and whether the organization that administers this infrastructure can satisfy the service needs of the team.

In sum, if analyses like these were performed, the infrastructure and choice of the components of the integrated development environment most likely would have been different. At the very least, the team could have developed strategies that would have lowered development risk. Following are a few questions that should be identified in a "decision analysis" catalog, along with the mitigation strategies, expected outcomes, and associated risks.

- Who should maintain the software infrastructure?
- How much new development technology should a project adopt?
- When can line managers and software engineering experts influence the decisions that a project makes?
- When should a project make and reconsider decisions?

The selection of these questions is based on problems that have arisen in several projects at JPL that, if thought about more thoroughly, might have resulted in increased efficiency of their software development teams.

4.2.2. Technical Decision Assessments

There are a variety of technical decisions that software teams make whose decisions should be thoroughly documented and analyzed. Later, a SQA engineer should assess these decisions to determine whether the analysis was accurate and reasonable and what the associated risks are, if any. This is one reason why SQA engineers need to

create and use a catalog of decisions that software engineers often need to make, and then assure that they make them, as needed.

Following are a few trades that should be in such a catalog. These trades generally have a significant impact on a software system's performance and reliability.

- Will the software team develop software based on an operating system? If so, will it be a RTOS? In either case, is an evaluation of the advantages and disadvantages of each option considered?
- If an operating system provides both kernel- and user-level implementations of threads, which one will be used?
- How should components communicate with one another (e.g., direct function call, shared memory, or message passing)?
- What components are processes, threads, and libraries? How does one determine the priority for each process or thread?
- How many devices should a project allow to control a MIL-STD1553B bus and will these devices use dynamic bus control?
- What type of memory management scheme will non-volatile memory use? What factors should be used to perform the analyses?

In sum, documenting these decisions, and the analyses leading up to these decisions, should result in greater mission success, especially after independent assessment of them.

5. Requisite Knowledge of a SQA Engineer

A key aspect of the role of a SQA engineer is to assist software engineers and their managers to perform their function in a quality manner. To do so requires a SQA engineer to have general knowledge of software engineering. This knowledge must span configuration management, requirements engineering, software design, and software verification. Specifically, the SQA engineer must know how to assess the quality of individual requirements and sets of requirements, architectural designs, and detailed designs of the components of the architecture; and possess a thorough knowledge of verification methods that include both review and testing techniques.

A SQA engineer must also possess a general knowledge of computer science. He or she must have a basic understanding of data structures, algorithms, computer architecture, and operating systems. At JPL, SQA engineers also must possess detailed knowledge of real-time programming, and the processors, memory devices, and bus protocols used by embedded software systems. Likewise, they must know how custom boards developed by JPL engineers implement these protocols to connect the processors, memory devices, and instrument payloads together. The reader may wonder why all this detailed knowledge is required – such knowledge enables a SQA engineer to assess the appropriateness of a variety of technical decisions.

A SQA engineer must also be knowledgeable of systems engineering (Forsberg et al, 2005). Systems engineering requires many of the same skills that software engineering requires – engineering requirements, developing architectures, verifying solutions, and validating that the solutions satisfy the customer's needs. However, systems engineering places a greater emphasis on decision analysis. Systems engineering must often examine multiple alternatives for achieving goals to produce an integrated solution that satisfies all goals. Systems engineering requires one to balance multiple system qualities while achieving an acceptable level of risk. Together, computer science, software engineering, and systems engineering knowledge permits a SQA engineer to assess the likelihood that a given process and a collection of technical decisions will result in a product having the specified qualities.

Furthermore, it is essential that a SQA engineer have knowledge of the application domain. At JPL, this means an SQA engineer must have knowledge in ground, flight, and instrument software. Without such domain knowledge, the SQA engineer cannot do much more than perform a checklist function. For example, when domain knowledge is lacking, a SQA engineer can perform compliance evaluations in one of two ways: by asking project personnel whether they have complied with a standard or requirement or verifying that compliance records exist. Without domain knowledge, the SQA engineer is unable to evaluate the given answers or to assess the adequacy of the compliance records. Hence, a checklist function generally provides little value to a project or institution.

6. Requisite Skills of a SQA Engineer

To communicate effectively with software engineers, a SQA engineer requires several skills. A SQA engineer must be proficient in the tools that software engineers use, including the environment in which they use these tools. For the purposes of this discussion, the environment includes the operating system and development environment that

software engineers generally use. To maintain proficiency in these tools and environments implies that SQA engineers must regularly use them. Probably the best way to do this is make their use part of their daily routine. In other words, SQA engineers should use the same host and development environments as the software engineers to do their work.

At JPL, most software engineers use some variant of UNIX as the host operating system and some variant of Eclipse to develop software. For example, flight software engineers use Linux and WindRiver WorkBench to develop software, which they deploy as part of an extended VxWorks kernel. Hence, a SQA engineer should use a workstation that uses the Linux operating system and Eclipse development environment to manage the work that they do and the records and reports they generate. For example, JPL SQA engineers are now using the same commercial product to manage findings that most JPL software engineers use to manage action items and failure reports, which is accessible through Eclipse.

Being able to communicate well with others is also an important skill of a SQA engineer. A SQA engineer must be able to write clear and concise prose in the reports he or she generates. Such prose must include effective arguments that can convince others to accept proposed recommendations. This is because it is seldom adequate to say one must change a practice because it violates an institutional requirement or deviates from accepted practice. The SQA engineer must provide a rationale for each recommendation, supported by concrete evidence. Finally, a SQA engineer must also have the verbal communication skills to communicate effectively with customers.

7. Reconsidering the Role of a SQA Engineer

Now that I have described existing definitions of a SQA engineer, identified the customers of a SQA engineer, explained the tasks that a SQA engineer performs, and discussed the knowledge and skills they need to perform these tasks, I can provide a description of the role of an SQA engineer appropriate for JPL. First, a SQA engineer must have general knowledge of computer science. More specifically, a SQA engineer must have a thorough understanding of data structures, algorithms, computer architecture, operating systems, real-time programming, software engineering, and systems engineering. Such knowledge is generally demonstrated by achieving a B.S. in Computer Science and a M.S. in Computer Science, Software Engineering, or Systems Engineering. A SQA

engineer should be proficient in the commonly used programming languages of the organizations he or she assures; at JPL, this is generally C, C++, Java, and Python.

Second, a SQA engineer must be skilled in the use of the operating systems and development environments in use by the organizations he or she assures; at JPL, this generally involves the use of a derivative of the UNIX operating system, Eclipse or an Eclipse-based integrated development environment, SVN, and JIRA. In addition, a SQA engineer must be able to communicate effectively with others, both verbally and in writing.

Third, a SQA engineer will communicate with several customers. The principal customers are the mission assurance manager, the SQA manager, software managers, and software process engineers. A SQA engineer will communicate with these people at various periodic meetings, such as monthly management reviews.

Fourth, a SQA engineer will perform several audits and analyses. Some of these audits and analyses will be driven by a defined periodicity, whereas others will by driven by events. Key events include milestone reviews and peer reviews of the software engineering organizations. Each audit or assessment follows a defined process using various checklists. At the end of each audit or assessment, the SQA engineer will generate reports for his or her customers that describe the performed audits and analyses, as well as general project status information. If findings are generated, the SQA engineer will track them to closure.

Fifth, the scope of work that a SQA engineer performs includes all software engineering activities, whether a subcontractor or an internal software engineering organization performs it. Similarly, a SQA engineer begins working on a project during the initial planning phase and stops working on it no sooner than one year after the start of maintenance. Critical tasks at the start of a project that a SQA engineer performs are the production of a cost estimate for SQA services and an evaluation of software engineering cost estimates. A critical activity that a SQA engineer performs at the end of development is witnessing the testing of software.

8. Summary

In the past, the SQA organization has assumed that the assurance organization was its sole customer, primarily because that is who pays it. More recently, the SQA organization has reconsidered who its customers are and reevaluated how it can better serve them. The primary result of this effort has been to include the project engineering

organizations as customers and to expand service to them. For example, on one project I performed all the software builds, ran all the code analysis tools, and examined and dispositioned the results of these tools, including those of the compilers (Kandt, 2009). Performing these activities gave me greater insight into the quality of the primary product of the software development team, which is real-time software that is embedded in spacecraft. Performing these activities required me to have many of the skills of the software engineers, in addition to those of a SQA engineer. However, the primary expansion of the role of the SQA engineer at JPL is to ensure that key management and technical decisions are made, that the resulting decisions are reasonable, and that the risks of those decisions are quantified. The rationale for expanding the role of the SQA engineer to include decision analysis is that most JPL mission failures are caused by bad decisions resulting from inadequate or omitted analysis (Kandt, 2010). Hence, the assessment of management and engineering decisions by SQA engineers is vitally important.

At JPL, SQA engineers are generally perceived as being the enforcers of software process and product standards. Although this is a necessary goal of a SQA engineer, it is not sufficient to provide value to all its customers and it is a perception that we are trying to change. As a result, we have concluded that, to perform effectively, a SQA engineer must have largely the same knowledge of the software engineers that build the software and the process engineers that define the development processes. Since we believe the quality of the SQA staff is strongly correlated with the engineering organization's ability to transition SQA personnel to the engineering organization, a goal of our SQA organization is to infuse SQA engineers into the engineering organizations as software engineers.

Following are some key lessons resulting from my work:

1. Defining a role statement for a SQA engineer clarifies whom a SQA engineer must satisfy and what he or she must do to satisfy them. In other words, the role statement bounds what a SQA engineer should do and for whom.
2. The role of a SQA engineer is dependent on the needs of all of his or her customers. Since these customers, and their needs, may vary, the SQA engineer must tailor the activities that he performs and the deliverables he produces such that all of his customers – those involved in engineering and assurance – receive value. To delight customers in the engineering organization, an SQA engineer should consider performing engineering tasks that provide the SQA organization with greater insight into product quality.

3. To provide maximum value to software engineers, SQA engineers must be able to communicate with software engineers using notations, languages, and tools for which they are accustomed. In addition, SQA personnel must persuasively argue their points of view, based on supportive evidence derived from actual work experience. That is, academic analyses typically will be viewed less favorably than the experience of their organizational peers.
4. The decisions that software engineers and managers make have a greater impact on product quality than the software development processes they use (Kandt, 2010). We have overwhelming evidence that poor management decisions lead to greater inefficiencies and lower effectiveness. Similarly, we have overwhelming evidence that mission failures and near-failures are almost always traceable to poor technical decisions, few of which are preventable by software development processes. Limited funding and schedule constraints are, unfortunately, significant causal factors influencing the applicability of this lesson.

In sum, the reader can use my experience and these lessons to improve the performance of the SQA function within his or her organization.

References

1 M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI: Guidelines for Process Integration and Product Development*, 2nd edition, Addison-Wesley, 2007.

2 P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley Professional, 2001.

3 R. K. Kandt, Experiences in Improving Flight Software Development Processes, *IEEE Software*, Volume 26, Issue 3, May-June 2009, Pages 58-64.

4 R. K. Kandt, Essential Practices for Developing Flight Software, submitted to *Software Process: Improvement and Practice*, 2010.

5 K. Forsberg, H. Mooz, and H. Cotterman, *Visualizing Project Management: Models and Frameworks for Mastering Complex Systems*, 3rd edition, Wiley, 2005.

6 G. G. Schulmeyer, *Handbook of Software Quality Assurance*, 4th edition, Artech House, 2008.

Acknowledgments

Milt Lavin reviewed preliminary versions of this paper. His comments helped to improve its quality, as did those of the independent reviewers. The author performed this work at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Figure 1: Relationships among key roles