What is Software Quality Assurance?

Dr. Linda H. Rosenberg
NASA

Thursday, 2 May 2002
Track 7: 9:00 - 9:40
Room 251 A - C

Software directly impacts not only mission success but also mission safety. Software Quality Assurance (SQA) is critical to the success of every mission at NASA, but the roles and responsibilities are often misunderstood. SQA covers all phases of the software development process, including safety, reliability, independent verification and validation, and metrics. The purpose of this article is to help the reader understand software quality assurance.

Within the complex systems developed throughout the aerospace industry, software is playing an increasingly important role in mission success. Methods for developing and assuring software are often not well understood by program managers and, thus, are often simply ignored. In such a case, ignorance is far from bliss; it is dangerous. During the past few years, NASA has emphasized the faster, better, and cheaper approach to developing missions, thereby making it more important than ever to ensure the quality of its software products. It is this imperative that makes the role of Software Quality Assurance (SQA) critical in the short term, but also linked to mission success in the long term.

Assuring software quality requires that engineering knowledge and discipline be applied at all phases of the development life cycle. And just as with hardware, the final step in developing quality products culminates in rigorous testing before release. Quality assurance engineers are also required to possess sufficient domain knowledge to evaluate the completeness and correctness of system requirements, and they must have the ability to determine whether the design has incorporated all requirements accurately. Ultimately, these specialists are responsible for advising management when or whether a product is reliable and meets quality standards.

This article starts by discussing what is meant by SQA. It then discusses the aspects of how software quality assurance is applied to both the products and the process. The article continues with some of the major components of software assurance. Software metrics are used to help numerically determine the quality of the products, noting they are underutilized and often poorly understood. Another area of quality assurance not well understood is independent verification and validation (IV&V); this article will touch briefly on the role it plays in quality assurance. Finally, it discusses the ways in which software safety and reliability are assessed from a quality perspective. These two areas are often neglected despite their critical role in mission success.

Definitions

Software quality assurance is a combination of three concepts: quality, software quality, and software quality assurance.

"In the real work of software development, criteria for quality are identified and applied to differing extents as a result of trade-off decisions."

While the terms are often used interchangeably, we need to understand the basics of quality before we can understand the components and problems of software quality assurance.

Quality Defined

Before defining software quality, we need to define what is meant by quality. The Institute of Electrical and Electronics Engineers' (IEEE) *Standard Glossary of Software Engineering Terminology* defines quality as "the degree to which a system, component, or process meets (1) specified requirements, and (2) customer or user needs or expectations [1]." The International Standards Organization (ISO) defines quality as "the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs [2]." IEEE and ISO definitions associate quality with the ability of the product or service to fulfill its function. This is achieved through the features and characteristics of the product.

While this definition seems to be clear and unambiguous, the concept of quality really is not. Kitchenham states quality is "hard to define, impossible to measure, easy to recognize [3]." Gillies states that "Quality is generally transparent when present, but easily recognized in its absence [4]."

Therefore, while we can define quality in theory, in practice, and in use, an absolute definition is elusive.

Software Quality Defined

Software quality is defined in the *Handbook of Software Quality Assurance* in multiple ways but concludes with this definition: "Software quality is the fitness for use of the software product [5]." This definition implies the evaluation of software quality related to the specification and application of software quality. There are, however, criteria that help in the evaluation of software quality. For each project, the appropriate criteria need to be identified for the environment.

Two of the most often-cited models applying the criteria are the GE model proposed by McCall, which was later adapted by Watts, and the Boehm model [4]. Below is a combined list of definitions of quality criteria for software.

- Correctness: extent to which a program fulfills its specifications.
- Efficiency: use of resources execution and storage.
- Flexibility: ease of making changes required by changes in the operating environment.
- Integrity: protection of the program from unauthorized access.
- Interoperability: effort required to couple the system to another system.
- Maintainability: effort required to locate and fix a fault in the program within its operating environment.

- Portability: effort required to transfer a program from one environment to another.
- Reliability: ability not to fail.
- Reusability: ease of re-using software in a different context.
- Testability: ease of testing the program to ensure that it is error-free and meets its specification.
- Usability: ease of use of the software.

In a perfect world all of these criteria would be met, but software is not developed or run in such a world, and trade-offs are a part of all development projects. Often the most efficient software is not portable, as portability would require additional code, decreasing the efficiency. Usability is subjective and varies depending on the system users. When using the above criteria to define the assurance objectives of a software system, the purpose and use of the system must be taken into account. In the real work of software development, criteria for quality are identified and applied to differing extents as a result of trade-off decisions.

Software Quality Assurance Defined

Again referencing IEEE, quality assurance is defined as "a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements [1]." This definition needs to be adapted to software taking into account that, unlike hardware systems, software is not subject to wear or physical breakage; consequently, its usefulness over time remains unchanged from its condition at delivery. Software quality assurance must be a systematic effort to improve the delivery condition. In the *Handbook of Software Quality Assurance*, the following definition is given: "Software quality assurance is the set of systematic activities providing evidence of the ability of the software process to produce a software product that is fit to use [5]." These activities are evaluated in part against the above criteria and measured as described in a later section of this article.

Software Quality Assurance Applied

The focus, therefore, of SQA is to monitor continuously throughout the software development life cycle to ensure the quality of the delivered product. This requires monitoring both the processes and the products. In process assurance, SQA provides management with objective feedback regarding compliance to approved plans, procedures, standards, and analyses. Product assurance activities focus on the changing level of product quality within each phase of the life cycle, such as the requirements, design, code, and test plan. The objective is to identify and eliminate defects throughout the life cycle as early as possible, thus reducing test and maintenance costs.

Process Assurance

It has been proven that the use of standards and process models has a positive impact on the quality of the final software. The purpose of standardization of SQA ensures that there is discipline and control in the software development process via independent evaluation [5]. ISO 9000 provided a way to gain external accreditation for a quality management system. Many companies have used the application of ISO to software, but the complaint is that it tends to fossilize procedures rather than encourage process improvement [4].

One of the most common software development models is the Software Engineering Institute's Capability Maturity Model® (CMM®), which has recently developed into the CMM Integration℠ (CMMI℠). The basic premise underlying both CMM and CMMI is that the quality of the software product is largely determined by the quality of the software development and maintenance processes used to build it [6].

"It has been proven that the use of standards and process models has a positive impact on the quality of the final software."

Many commercial standards are also found in common practice for software development. Many organizations such as The Department of Defense and NASA have, in the past, developed their own standards for software development, but recently have embraced the use of commercial standards instead. It is now NASA's policy to use commercial standards whenever possible, thus encouraging more standardization not only across NASA but within industry also.

Product Assurance

At NASA's Goddard Space Flight Center (GSFC), software quality assurance is carried out by an independent group of people whose sole function is to monitor quality implementation. The Assurance Management Office recently created a list of tasks that SQA should perform during each phase of the software development life cycle. This list is comprehensive and starts in the concept phase of a proposed project and concludes with the operations and maintenance phase. For example, in the concept phase, SQA should generate and/or assist in the development/review of various program/project plans, including but not limited to project management plans, subcontract management, etc. In the requirements phase, SQA should obviously generate and/or assist in the generation of requirements, but it should also do activities such as observing, witnessing and/or participating in prototyping activities.

To accomplish all of these tasks would be an ideal set of SQA activities on a project, but projects rarely have sufficient funds or need to perform them all. For most projects, the amount of SQA to be applied is negotiated based on the purpose, degree of mission risk, and the funding level of the project. This negotiation is critical to the success of SQA. In the following sections, I will discuss four activities in which SQA must participate during all phases: metrics, IV&V, safety, and reliability.

Metrics

Software metrics are often ignored during the early software development life-cycle phases and are not an activity generally associated with SQA – but should be. For SQA practitioners, with their responsibility for assuring both the processes and products of the software development, measurement is critical. Throughout each of the life-cycle phases, metrics can be used to help in the evaluation.

The Software Assurance Technology Center (SATC) at GSFC has identified relevant metrics that can help projects better evaluate the quality of their products at fixed points within their development. For example, SATC developed a tool that derives metric information by analyzing requirement specification documents. Known as Automated Requirements Measurement,[1] this tool provides indicators of the quality of the requirements set. The tool's objective is to identify terms within the text that may cause requirements to be ambiguous and hence difficult to test and to identify any requirements that are incomplete [7].

It is up to the SQA organization to be cognizant of available and relevant metrics that help evaluate and assure products. When projects consistently use software metrics as part of their development, the SQA team needs only to validate the metrics and ensure the correct data interpretation. If a project is not employing metrics, however, then it is the responsibility of SQA to encourage, and perhaps facilitate, their use or to develop an independent metrics program for sufficient insight into the development.

Independent Verification and Validation

IV&V is defined by three components; it must be independent technically, managerially, and financially. IV&V must prioritize its own efforts, identifying where to focus its activities. It must have a clear reporting route to the program management, and the budget for these efforts must be allocated and controlled by the program. Control must occur at a level that is independent of the development organization such that the effectiveness of the IV&V activity is not compromised.

Verification is defined as the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase, i.e., whether or not it is internally complete, consistent, and correct enough to support the next phase. Validation is the process of evaluating software throughout its development process to ensure compliance with software requirements. Verification often asks the question, "Are we building the product right?" Validation asks, "Are we building the right product?"

NASA has a facility in West Virginia whose primary purpose is the accomplishment of IV&V. Without SQA, IV&V is expensive and often less effective. Where SQA is a broad blanket across the project, overseeing all process and product activities, including software, IV&V focuses on only those processes and products determined to have the highest risk and does an in-depth evaluation of them.

Safety

Safety is a team effort and is everyone's responsibility. Software is a vital part of the system. Project managers, systems engineers, software leads and engineers, software assurance or quality assurance (QA), and system safety personnel all play a part in creating a safe system. Safety-critical software is defined by the NASA Software Safety Standard as "Software that directly, or indirectly, contributes to the occurrence of a hazardous system state, controls or monitors safety critical functions, runs on the same system as safety critical software or impacts systems that run safety critical software, or handles safety critical data [8]." The goal is for the QA activity to ensure that software contributes to the safety and functionality of the whole system.

When a device or system could possibly lead to injury, death, or the loss of vital (and expensive) equipment, system safety is always involved. Often hardware devices are used to mitigate the hazard potential or to provide a *fail-safe* mechanism should the worst happen. As software becomes a larger part of electromechanical systems, hardware hazard controls are being replaced or backed up by software controls. Software has the ability not only to detect certain types of error conditions more quickly than hardware but also to respond more intelligently, thereby avoiding a potentially hazardous state. The increased reliance on software means that the safety and reliability of the software become vital components in a safe system [8].

"When projects consistently use software metrics as part of their development, the Software Quality Assurance team needs only to validate the metrics and ensure the correct data interpretation."

Reliability

IEEE defines software reliability as "The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software [9]." Using this definition, expectations of reliability must be based on how the system is to be used and for what length of time. At NASA, many of our satellites fly for multiple years; the reliability of their software must support the expected lifetime. The conditions of that software's use will be specified by the satellite's mission.

IEEE continues to define software reliability management as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault protection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule, and performance [9]." This definition puts the burden of reliability not just on the testing phase, but on the entire life cycle to ensure errors are prevented starting in the requirements phase determining the quality of such attributes as phrasing, completeness, and clarity. Throughout the life cycle, errors should be detected and removed using such techniques as code walkthroughs and inspections. Relevant measurements should be used at all phases to ensure the effectiveness of all assurance activities. In the testing phase, reliability can be evaluated using one of the many reliability models. These models, however, must be applied with very strict rigor to ensure accuracy.

It is the responsibility of the SQA organization to ensure that reliability is continuously promoted and evaluated throughout the life cycle as specified above. Quality cannot be tested in at the end of a project; it must be built in as the software is being developed. Reliability also impacts safety – a system cannot be deemed safe if it is not reliable.

Conclusion

SQA is faced with many challenges starting with the method of defining quality for software. There needs to be a common understanding as to what is high-quality software, but the software usage environment usually influences the final definition. There are many aspects of SQA, from those within the phases of the software development life cycle to those that span multiple phases, i.e., safety, reliability, and IV&V. SQA is a very complex area that is critical to the ultimate success of a project; it is also one that requires a rather diverse set of skills. New knowledge areas such as software safety and reliability are now being added to the core set of required skills. Finally, SQA must be independent from development organizations to be successful.◆

References

1. IEEE Std 610.12-1990. Glossary of Software Engineering Terminology. Institute of Electrical and Electronics

Engineers, Inc., 1990.
2. ISO 9003-3-1991. Quality Management and Quality Assurance Standards, Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software. International Standards Organization, 1991.
3. Kitchenham, Barbara, and Shari Lawrence Pfleeger. "Software Quality: The Elusive Target." IEEE Software 13, 1, Jan. 1996: 12-21.
4. Gillies, Alan C. Software Quality, Theory and Management. International Thomson Computer Press, 1997.
5. Schulmeyer, G. Gordon, and James I. McManus. Handbook of Software Quality Assurance, 3rd ed. Prentice Hall PRT, 1998.
6. Software Engineering Institute. Capability Maturity Model. Carnegie Mellon University, 1991.
7. Wilson, W., L. Rosenberg, and L. Hyatt. "Automated Quality Analysis of Natural Language Requirement Specifications." Proceedings of the 14th Annual Pacific Northwest Software Quality Conference, Portland, Ore., 1996.
8. NASA-STD-8719.13A. NASA Software Safety Standard. NASA, 2001.
9. IEEE Std 982.2-1988. Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software. Institute of Electrical and Electronics Engineers, Inc., 1988.

Note

1. Available on the SATC Web site at no cost, see <http://satc.gsfc.nasa.gov>.

About the Author

Linda H. Rosenberg, Ph.D., serves as the chief scientist for Software Assurance for Goddard Space Flight Center, NASA. She is a recognized international expert in the areas of software assurance, software metrics, requirements, and reliability. Dr. Rosenberg has a doctorate degree in computer science, a master's of engineering science degree in computer science, and a bachelor's of science degree in mathematics.

Office of Systems Safety and Mission Assurance
Goddard Space Flight Center, NASA
Building 6 Code 300
Greenbelt, MD 20771
Phone: (301) 286-0087
Fax: (301) 286-1667
E-mail: linda.h.rosenberg@gsfc.nasa.gov