Software Quality Assurance Engineering at NASA

Linda H. Rosenberg, Ph.D.
Chief Scientist for Software Assurance
Office of Systems Safety and Mission Assurance
Goddard Space Flight Center, NASA
Greenbelt, MD 20771 USA
301-286-0087 (fax 301-286-1667)
Linda.Rosenberg@gsfc.nasa.gov

Albert M. Gallo, Jr.
Director, Software Assurance Technology Center
Office of Systems Safety and Mission Assurance
Goddard Space Flight Center, NASA
Greenbelt, MD 20771 USA
301-286-3763 (fax 301-286-1667)
Al.Gallo@gsfc.nasa.gov

Abstract - As our missions on the ground and into space expand to reach new horizons, the complexities and importance of software to mission success also increase. Software has taken on a new, enhanced role and now directly impacts not only mission success, but also the safety of the missions. 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, with specific activities to assure both the processes used and the products developed. It also includes areas such as safety, reliability, IV&V, and metrics. The purpose of this paper is to help the reader understand what software quality assurance entails for both the processes and the products.

TABLE OF CONTENTS

1. INTRODUCTION
2. DEFINITIONS
3. COMPONENTS OF SOFTWARE QUALITY ASSURANCE
4. CONCLUSIONS

1. INTRODUCTION

Within the complex systems developed throughout the aerospace industry, software is playing an increasingly important role in mission success. Yet there is no shortage of evidence indicating that software quality programs are designed and implemented differently from those of other, more tangible, physical system elements. 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. Over 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 critical in the short term, but also linked to mission success in the long term.

Assuring the quality of software requires that engineering knowledge and discipline be applied at all phases of the development lifecycle. And just as with hardware, the final step in developing quality products culminates in rigorous testing prior to release for use. 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. From a technical standpoint, quality engineers should have some knowledge of the coding language(s) being used as well as in-depth knowledge of appropriate standards and technical limitations, for they are a manager’s objective representative at code walkthroughs and formal inspections. During testing, they determine whether a sufficient level of rigor has been employed in uncovering and resolving defects. Ultimately, these specialists are responsible for advising management when or whether a product is reliable and meets quality standards.

All these diverse activities must be brought together effectively via well-crafted and mature processes, which, like all activities, are continuously refined and improved. Over the past years, there has been much debate centered on whether or not software engineering is truly “engineering” or simply a misnomer. The answer seems obvious, however, when considering that software quality assurance is a field of software engineering. The complex, technical and highly disciplined set of activities associated with software quality assurance certainly makes software engineering part of the engineering discipline.

This paper discusses the engineering aspects of software quality assurance as it relates to NASA’s perspective of mission success. It looks at the major components of the software assurance activity as well as the tools and techniques available to evaluate the quality of both the intermediate and final products. One of the more useful tools available in support of software quality is a comprehensive Metrics Program. The Software Assurance Technology Center at NASA’s Goddard Space Flight Center (GSFC) has developed a core set of measurable quality indicators that identify risk areas in the development process. Finally, we discuss the ways in which software safety and reliability are assessed from a quality perspective. Investigation shows these two areas are often more neglected than others despite their critical role in mission success. The impact of COTS integration and object-

1 U.S. Government work not protected by U.S. copyright
oriented development are briefly discussed before summarizing NASA's research program in software assurance.

2. DEFINITIONS

Quality Defined

Prior to defining software quality, we need to define what is meant by "quality." 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] Gilles states "Quality is generally transparent when present, but easily recognized in its absence."[4] Gilles describes quality using the following insights:

- Quality is not absolute; it means different things in different situations.
- Quality is multidimensional, it has many contributing factors, and it is not easily summarized in simple, quantitative ways.
- Some aspects of quality can be measured but the most easily measured criteria are not necessarily the most important.
- Quality is subject to constraints; assessment of quality in most cases cannot be separated from cost. (Cost may be defined as any critical resource.)
- Quality is about acceptable compromises. When quality is constrained and compromises are required, some quality criteria may be sacrificed more acceptably than others.
- Quality criteria are independent, but interact with each other causing conflicts.

Therefore, while we can define quality in theory, in practice and use, its realization can be elusive.

Software Quality Defined

Software quality is defined in the Handbook of Software Quality Assurance in multiple ways but concludes with the 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] Table 1 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

Table 1: McCall Software Quality Criteria

While the criteria above are again appropriate, they must be tailored and enhanced based on the environment. As an example, a large manufacturing company cited by Gilles stated the software quality criteria are shown in Table 2. [4]

- Technical correctness
- User correctness
- Reliability
- Efficiency
- Integrity
- Security
- Understandability
- Flexibility
- Ease of interface
- Portability
- User consultation
- Accuracy
- Timeliness
- Time to use
- Appeal
- User flexibility
- Cost/benefit
- User friendliness

Table 2: Gilles Software Quality Criteria

At NASA, the criteria for evaluation of software quality are also taken from Table 1 depending on the application. For example, flight software that flies on a single mission
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 SQA Handbook, the following definition is given. “Software quality assurance (SQA) 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]

3. SOFTWARE QUALITY ASSURANCE APPLIED

The focus, therefore, of software quality assurance is to monitor processes 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 process compliance to approved plans, procedures, standards and analyses. Product assurance activities focus on the quality of the product 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. SQA also encompasses many aspects of software development, such as safety, reliability, IV&V and metrics. In the remainder of this paper we will first briefly discuss process assurance, then look at SQA life cycle phase activities and conclude with the relationship and responsibility of SQA to safety, reliability, IV&V and metrics.

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 software quality assurance ensures that there is discipline and control in the software development process via independent evaluation. [5] ISO9000 provided a way to gain external accreditation for a quality management system. The application of ISO for software has been used by many companies, but the complaint is that it tends to fossilize procedures rather than encourage process improvement. [4] A range of standards and models has been developed which seek to provide the benefits of quality standards while recognizing different stages of development and the need to improve.

One of the most common software development models is the work of the Software Engineering Institute (SEI) at Carnegie Mellon University, the Capability Maturity Model (CMM), which has recently developed into the Capability Maturity Model Integrated (CMMI). The basic premise underlying the 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. The CMM/CMMI are defined as a five-level framework assessing the maturity of an organization’s software processes, based on specific key process areas. [6]

SPICE is a major international initiative focused in Europe and Australia to develop a Standard for Software Process Assessment. This project is being carried out under the auspices of the International Committee on Software Engineering Standards, ISO/JTC1. The SPICE standards cover software process assessment, improvement and capabilities. [7]

Many commercial standards are also found in common practice for software development. Some of the more common ones are the DOD issued MIL-STD-498, Software Development and Documentation, IEEE-STD1074, IEEE Standard for Developing Software Life Cycle processes, and EIA/IEEE 12207, Information Technology – Software Life Cycle Processes. [3] Many organizations such as NASA have, in the past developed their own standards for software development, but recently have recognized the value of using 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.

Software quality assurance is an ongoing process to ensure that the development is being carried out according to the procedures laid down by the standard or model. It should also have a role in monitoring the effectiveness of the procedures intended to establish a quality culture.

Product Assurance

At NASA’s Goddard Space Flight Center, software quality assurance is carried out by an independent group of people whose function is solely to monitor the implementation of quality. At Goddard Space Flight Center (GSFC), responsibility for SQA is assigned to the Office of Systems Safety and Mission Assurance. The Assurance Management Office recently created a list of tasks that SQA should perform at each phase of the software development life cycle. Below is the list of activities within phases.

Concept Phase Activities

- Generate and/or assist in the generation of concept of operations and operational scenarios.
- Attend concept reviews and facilitate tracking/resolution of issues/concerns/risks, etc.
- Generate and/or assist in the identification of program/project risks and mitigation strategies/techniques.
- Generate and/or assist in the development/review of various program/project plans including but not limited to project management plans, subcontract management plans, etc.
- Generate and/or assist in the generation of and/or review of assurance requirements.

Requirements Phase Activities
- Generate and/or assist in the generation of requirements.
- Attend and/or participate in requirement working group meetings/ Joint Application Development meetings.
- Attend requirements reviews and track/maintain any issues/resolution tracking logs, tools, etc.
- Review and analyze requirements for industry-acceptable and required characteristics (testability, traceability, consistency, clarity, etc – see IEEE stds, etc).
- Execute Automated Requirement Measurement (ARM) tool and resolve any requirements quality related issues.
- Generate and/or review requirements traceability/verification matrices.
- Generate and/or review software development plans, configuration management plans, quality assurance plans, test plans and related artifacts and processes.
- Review and provide guidance on program/project metrics including strengths, weaknesses, limitations, etc.
- Assist in the tracking and facilitation of program/project related items such as problem reports, etc.

Design Phase Activities
- Attend and/or participate in design working group meetings/ Joint Application Development meetings.
- Attend and/or participate in design reviews, and track/maintain any issues/resolution tracking logs, tools, etc.
- Review and/or analyze design for industry-acceptable and required characteristics, etc – maintainability, reusability, IEEE stds, etc).
- Generate and/or review requirements to design traceability/verification matrices
- Review and/or track program/project metrics for trends, risks, potential problems, etc.
- Attend change control and defect review board meetings and participate in the assessment of changes and/or defects (i.e. concur on closeout of S/W discrepancy reports, inspection reports, etc.).
- Observe, witness and/or participate in prototyping efforts. Provide feedback as applicable on prototyping efforts/results.
- Assist in the tracking and facilitation of program/project related items such as RIDS, etc.

Implementation Phase Activities
- Attend code walkthroughs and peer reviews. Participate in the tracking and resolution of any issues, etc.
- Review and/or assess code per organization’s coding standards.
- Generate and/or review requirements to code traceability/verification matrices.
- Review/track program/project metrics for trends, risks, potential problems, etc.
- Attend change control and defect review board meetings and participate in the assessment of changes and/or defects (i.e. concur on closeout of S/W discrepancy reports, inspection reports, etc.).
- Assist in the tracking and facilitation of program/project related items such as RIDS, etc.

Test Phase Activities
- Witness, observe and/or assist in testing activities (integration, system acceptance, operational readiness/launch readiness)
- Generate and/or review test defects/test reports and other test-related artifacts as applicable.
- Generate and/or review requirements traceability/verification matrices.
- Attend change control and defect review board meetings and participate in the assessment of changes and/or defects.
- Attend and/or participate in test readiness, operational readiness and launch readiness reviews.
- Review and/or verify test environment and test tools’ configuration/qualification for use.
- Review and/or verify test data sets.
- Review and/or assess/verify user’s guides, training manuals, etc.

Operations and Maintenance Phase Activities
- Support launch range activities in an oversight capacity.
- Attend change control and defect review board meetings and participate in the assessment of changes and/or defects.
- Generate lessons learned/post mortem reports.
SQA related activities performed throughout all lifecycle phases
- Conduct audits of program/project processes and/or products.
- Review lessons learned database(s) for applicability to program/project.
- Generate lessons learned throughout lifecycle phases.

This list represents 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.

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 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 which run safety critical software, or handles safety critical data.”[8] The goal is for the quality assurance 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]

The System Safety Program Plan should adequately describe interfaces within the Assurance disciplines as well as the other Project disciplines. It is the responsibility of the SQA organization not only to identify the safety critical software components but also to ensure the appropriate processes are correctly followed. All analyses and tasks should be complementary and supportive regardless of which group (development or assurance) has the responsibility. The analyses and tasks may be shared between the groups and, within each discipline, according to the resources and expertise of the project personnel.

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. The inputs to the system determine whether existing faults, if any are encountered.”[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, thus the reliability of associated software must be appropriately reliable to support the expected lifetime of the satellite. The conditions of that software’s use will be on a specified satellite’s system.

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 SQA organization to ensure that reliability is continuously promoted and evaluated throughout the life cycle as specified above. At each life cycle phase, SQA needs to monitor the processes that are being applied, thereby ensuring the greatest number of errors are detected and removed as early as possible within the life cycle. Many techniques and models are used in conjunction with reliability and it is the responsibility of software quality assurance organization to ensure they are applied correctly. 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.

Independent Verification and Validation

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 is often the question “Are we building the product right?”, while Validation asks “Are we building the right product?”

At NASA, Software IV&V is defined as a systems engineering process employing rigorous methodologies for evaluating the correctness and quality of the software product throughout the software life cycle. NASA has a Facility in West Virginia whose primary purpose is the accomplishment of IV&V. This Facility is part of GSFC’s Office of Systems Safety and Mission Assurance, recognizing the relationship and importance of IV&V to SQA. Without SQA, IV&V is expensive and often not as 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.

**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. In 1889, Lord Kelvin stated “… when you can measure what you are speaking about, and express it in numbers, you know something about it; when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind…” 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 is part of the Quality Assurance Directorate. It is in part their responsibility to develop comprehensive SQA metrics programs, and to assist projects with effective implementations.

For each phase of the development life cycle, the SATC has identified relevant metrics that can help projects better evaluate the quality of their products at fixed points within their development. They have also developed focused questions to help projects identify what it is they are trying to determine through the use of metrics. Some of these questions are:

**Requirements**
- Can you test the requirements?
- Are the requirements complete?
- Are you testing each requirement?

**Development (Design & Code)**
- How much testing is necessary?
- How hard is it to change the components?
- Can I reuse any of the components?
- What is the quality and complexity of the code?

**Testing**
- How many errors still remain?
- What is the reliability?
- When can I stop testing?

For each development phase, metrics can be used to help guide the developers, designers and testers as well as managers. For example, SATC developed a tool that derives metric information by analyzing requirement specification documents. Known as ARM, Automated Requirements Measurement, this tool provides indicators of the quality of the requirements set. The objective of the tool 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. [10]

The SATC is also working on measurements specifically for object oriented development. It is recognized that while some metrics associated with traditional functional development, such as size and complexity, are still valid for object oriented design, it is also necessary to use additional metrics evaluating the complexity of the structure or “object orientedness.” Metrics such as weighted methods per class and depth of tree can identify potential problems with the structure. [11] Another aspect of software development, specifically development that incorporates COTS-based systems, requires a slightly different approach with metrics. When utilizing COTS, the emphasis cannot be on the internal complexity of the systems; rather, it must be on the interfaces and “glue” code, which is where most of the errors in this type of system are introduced.

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 interpretation of the data. If a project is not employing metrics, however, then it is the responsibility of SQA to encourage, and perhaps facilitate, their use, or develop an independent metrics program for sufficient insight into the development.

**4. Conclusion**

Software Quality Assurance 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 final definition is usually influenced by the environment of the software usage. 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.

ACKNOWLEDGEMENTS

The list of activities performed by the software quality assurance organization was compiled in cooperation with Mr. Stanley Iarosis in the Assurance Management Office at GSFC NASA.

REFERENCES


Dr. Linda H. Rosenberg serves as the Chief Scientist for Software Assurance for Goddard Space Flight Center (GSFC), NASA in the Office of Systems Safety and Mission Assurance Directorate and is the former division chief of the Software Assurance Technology Office (SATO). Dr. Rosenberg is a recognized International expert in the areas of software assurance, software metrics, requirements and reliability. Dr. Rosenberg is also an adjunct professor at University of Maryland, Baltimore for the Masters/Doctoral Program. Dr. Rosenberg holds a Ph.D. in Computer Science from the University of Maryland, an M.E.S. in Computer Science from Loyola College, and a B.S. in Mathematics from Towson University.

Mr. Albert Gallo is the recently-appointed Head of the Software Assurance Technology Center at the Goddard Space Flight Center (GSFC), NASA and has more than 15 years experience in both software development and Quality Assurance. He also holds Bachelors degrees in Pure Mathematics and Computer Science, as well as a Master of Science in Technical Management from Johns Hopkins University, and is currently completing a Doctorate in Engineering Management at The George Washington University in Washington, DC. Prior to joining NASA, he worked at the Actuarial Division of the US Department of Labor.
This page intentionally left blank.