Critical Issues In Software Quality Assurance: An Exploratory Study

William D. Burg, (E-mail: burg@uab.edu), University of Alabama, Birmingham
Sanjay K. Singh, (E-mail: ssingh@uab.edu), University of Alabama, Birmingham

Abstract

Recent reviews of the Software Quality Assurance (SQA) literature expose that in most organizations, systems development is characterized by recurrent problems, such as poor system quality. The conclusions of these reviews is that SQA efforts have failed due primarily to their failure to adequately address the linkages between the organizational and process aspects of quality assurance in software development projects. As an initial step in identifying and understanding the linkages between the organizational and process aspects of SQA, the authors conducted an exploratory study to elicit a set of issues critical to the assurance of delivering reliable software.

Introduction

In the last decade alone, software failures have wrecked a European satellite launch, delayed the opening of the new Denver airport for a year, destroyed a NASA Mars mission, killed four marines in a helicopter crash, induced a U.S. Navy ship to destroy a civilian airliner killing hundreds of civilians, and shut down ambulance systems in London, leading to as many as 30 deaths. Mizuho Bank of Japan, one of the largest banks in the world in terms of assets, launched a software system without thorough quality assurance. The system crashed shortly after it was launched. As a result, customers were unable to make withdrawals or deposits at approximately 7,000 ATMs for almost a month, and multiple accounts were debited by mistake.

According to a recently released study commissioned by the Department of Commerce's National Institute of Standards and Technology (NIST), software errors cost the U.S. economy an estimated $59.5 billion annually. The NIST study also found that more than a third of these costs, or an estimated $22.2 billion, could be eliminated through earlier and more effective identification and removal of software defects (NIST, p. ES-11).

With the increasing reliance on software for everything from managing personal finances to performing eye surgery, quality in software is no longer optional. In addition, the use of web-based applications linked to legacy systems gives both customers and business partners direct experience with the reliability and quality of an organization's systems. Given the importance of delivering software of sufficient quality, one would expect that software quality assurance (SQA) practices themselves would be not only mature, but also pervasively implemented.

Yet recent reviews of the SQA literature expose that in most organization, systems development is characterized by recurrent problems, such as poor system quality, long development lead times, user dissatisfaction, and high costs (Ravichandran and Arun, p. 382). In assessing the state of SQA, these reviews point out that managing systems development quality involves managing two fundamental but different dimensions. First, a sophisticated technological infrastructure based upon adherence to accepted software engineering standards for the design of high-quality software needs to be implemented and used throughout the software development process. Second, an organizational system for managing all aspects of the overall software development process – one that encourages all stakeholders to engage in quality-oriented behaviors – must be established and utilized throughout the software development process. These reviews conclude that, unfortunately, SQA research has focused primarily on the technical aspects of SQA, to the detriment of the organizational and process aspects.

Given the success of Total Quality Management (TQM) in improving the quality of manufactured products, a number of SQA practitioners have suggested that applying TQM practices to software development offers an effective approach to improving software quality. A review of SQA literature indicates that TQM practices are slowly being adopted by SQA organizations but with mixed results (Fox and Flakes, p. 25; Zultner, p. 79). In a number of organizations, applying TQM to their SQA efforts resulted in a significant improvement in the quality of the delivered systems. However, in an equal number of other initiatives, the results ranged from modest improvements in quality to the complete abandonment of the quality program (Zultner, p. 79). In assessing these mixed results, researchers attribute these partial or complete failures to the piecemeal adoption of selected TQM practices which is a direct violation of the fundamental precept of TQM that organizations should be viewed as systems of interlinked processes. A common example of a piecemeal application of the TQM approach in these partial and complete failures is the application of statistical quality control techniques to the use of software metrics without a corresponding integration of the reward and punishment system under which software developers operate.

Even within SQA research, recent reviews of the literature indicate that a significant discontinuity exists between the tenets of TQM and current academic research (Hall and Fenton, pp. 55-56). One major reason cited for this discontinuity is a lack of relevance in *scope* between the tenets of TQM and academic research (Fenton and Neil, pp. 151-152). As pointed out in these reviews, academic research has focused primarily on developing software metrics that are rigorous, elegant, and based upon formal models, but that, rather than being holistic in nature and scope, cover only one conceptual aspect of the complete systems development life cycle (SDLC) - a direct violation of the fundamental percept of TQM that organizations should be viewed as systems of interlinked processes. For example, a system’s requirements might be captured using use-case scenarios and entity-relationship diagrams, while its design might be described in terms of class, object, collaboration, and activity diagrams. For each type of conceptual model, a unique set of SQA metrics have been developed. The use of single-dimensioned metrics developed in isolation and that applied to specific elements of one aspect of the overall development project fails to meet the basic tenets of TQM that focus on measuring the overall quality of the developed system (i.e. of all the interconnected aspects).

A second major reason cited for this discontinuity is a lack of relevance in *content* between the TQM process and academic research (Fenton and Neil, pp. 151-152). In terms of content, academic research has focused almost exclusively on developing computational-based metrics derived from the details involved in implementing the project. The problem with such metrics is that when used, they tend to prescribe the subsequent development process by causing developers to focus on implementation issues too soon. Instead of providing management with the relevant metrics necessary to enable them to make choices between competing designs (based upon differing conceptual viewpoints with inherent advantages and disadvantages), and improving all aspects of the software development process, these metrics tend to steer the software development process towards the implementation upon which the metrics were defined.

The conclusions of the reviews is that SQA efforts have failed to deliver the desired increase in software quality due primarily to their failure to adequately address the linkages between the organizational and process aspects of quality assurance in software development projects. In order for SQA practices to improve, SQA organizations must be able to effectively manage these linkages. The first step in this effort is to identify and understand these linkages. As an initial step in identifying and understanding the linkages between the organizational and process aspects of SQA, the authors interviewed practicing information systems (IS) project managers regarding SQA practices throughout the SDLC to elicit a set of issues critical to the assurance of delivering reliable software (Eisenhardt, p. 532).

Methodology

In order to identify the issues critical to the assurance of the quality of software delivered by IS development projects, the authors conducted structured interviews of senior IS personnel responsible for filling the role of IS project managers. For some firms, the responsibility of project manager fell to the Chief Information Office (CIO) or Vice-President of Information Technology (VP of IT) depending on the size of the firm and the nature of the specific project. In most cases, the interviewees simply held the title of project manager. The authors selected firms involved in IS development projects located primarily in the Midwest, South, and Southeastern United States. Fifteen firms were chosen, representing a cross-section of industries and project methodologies, and ensuring a broad view of the principle SQA aspects associated with IS project management. All fifteen of the firms agreed to participate knowing the focus of the research project. At some firms, more than one individual was interviewed.

The structured interviews consisted of a series of well-defined, open-ended questions (Eisenhardt, p. 537). The interviewees were asked to discuss the importance of SQA to their organization and, if SQA is important, to discuss how IS development projects needed to be organized, staffed, implemented, and managed to assure the quality of delivered software. Specifically, the interviewees were asked to identify and explain problems that they have experienced due to a lack of software quality and encouraged to explain the underlying reasons and actions that, in their experience, lead to a lack of quality in the delivered software. These initial set of questions were followed by extemporaneous, probing questions based upon the interviewees' responses. The interviews lasted between one and two hours.

The profile of the interviewees was as follows. The interviewees averaged 15 years of experience in IS, 6 years in project management, and 8 years with their current employers. The size of the interviewees' IT departments ranged from 10 to 500 employees, with an average of 50. The managers currently were responsible for projects from $5 million to $20 million in cost, with an average of $1.7 million. The durations of these projects were from six months to four years, with an average of one and a half years. Except for one manager who was responsible for several hundred developers, an average of 16 team members reported directly or indirectly to the managers. Eleven of the subjects managed in-house development projects; the other four were in charge of out-sourced projects. The managers' job titles, industries, and project methodologies appear in Table 1. The profile of the interviewees is shown in Table 2.

Table 1 - Organizational Profile of Interviewees' Companies

Number of Companies	**15**
Industries	Insurance – 4 Banking/Finance – 3 Manufacturing – 2 Healthcare – 2 Local/State Government – 1 Publishing/Media – 2 Educational Institution – 1
Revenue	$100m – $250m = 2 $250m – $1bn = 3 $1bn – $5bn = 6 > $5bn = 4
Location	South South East Midwest
Outsourcing	All of them do partial and selective outsourcing
Methodology/Life Cycle	Most of them adhere to or are thinking about adopting formal software engineering methodologies. Vast majority of the methodologies are vendor specific – e.g., Rational Unified Process. Some have developed their own in partnership with vendors. All the large corporations have a Project Management Office responsible the time/cost/quality of projects. 8 respondents are building BI tools or adopting IT project management software.

Table 2 – Interviewees Profile

Number of Interviewees	**20 (at some corporation, more than 1)**
Average years of experience	15
Job Titles	VP – 5 CIO – 3 Director – 4 Manager – 3
Average IT department size	50 Range from 10 – 500 employees
IT Budget	$ 5 million to $ 200 million

Discussion of Interviewees Responses

The following nine issues were identified by the interviewees as critical to the quality assurance of software delivered by IS development projects.

- Understand that quality must be designed and built into the project from its inception
- Understand the differences in the nature of functional and nonfunctional requirements
- Assign the overall project quality the highest priority and make it everyone's responsibility
- Staff the SQA process with the right personnel and involve them from the beginning
- Design SQA metrics using a goal based framework
- Choose appropriate models and tools to support the SQA process in every project phase
- Use SQA reviews to identify issues in every phase of the software development lifecycle
- Use iterative build cycles to address quality issues that span features and seams
- Realize that SQA is a process unto itself that must be measure and continuously improved

The issues are discussed in a chronological order based upon the SDLC. It is important to note that in the interviewees' experiences no one issue is considered any more important than any other because a failure to successfully address any one issue can result in overall project failure.

Issue #1: Understand that quality must be designed and built into the project from its inception

Building quality software requires recognizing that testing and SQA are not the same thing. Testing is the process of checking to see if the appropriate functionality is already designed and built into the software application, whereas SQA is the process of capturing and understanding user requirements, and designing, building, and verifying the software so that the set of functionality required by all stakeholders is provided. Because testing alone does not provide software developers with all the tools necessary to build quality software, quality can not be build into software using testing alone.

In designing a SQA process, managers must understand that quality must be designed and build into the final product in each of the SDLC phases. Because building software is a process that constantly involves making design or implementation decisions within each phase of the SDLC, every aspect of the SDLC, as well as the concerns of every stakeholder that can impact the quality of the final product, must be addressed by the SQA process. In order to produce quality software, managers must start by understanding that quality must be designed and built into the project, starting with its inception and continuing through each subsequent phase.

Issue #2: Understand the differences in the nature of functional and nonfunctional requirements

By definition, quality software is software that simultaneously delivers all of the features and functions required by all of the stakeholders. As conceptually simplistic as achieving this goal may seem, in reality simultaneously delivering all of the features and functions is very difficult, if not impossible. To understand why this is true, one must understand the difference in the nature of functional and nonfunctional requirements.

Functional requirements are requirements designed to meet a specific business need. An example would be performing a credit check before shipping merchandise to a customer on account. In practice, functional requirements are best discovered by involving senior business users (who understand their specific business requirements) in the requirements discovery. In cases where functional requirements span multiple departments or functional areas (i.e. an ERP implementation), representatives from each functional area as well as owners of the business processes that span the functional areas (i.e. order fulfillment) must be included in the requirements discovery process. This helps ensure that shared responsibilities do not fall through the cracks.

Nonfunctional requirements are far broader and more encompassing than are functional requirements. In addition to spanning all functional requirements, nonfunctional requirements span all users and usage patterns, and encompass security, reliability, and response time issues. Therefore, discovering nonfunctional requirements necessitates using requirements specification techniques that span all aspects of the delivered system simultaneously. Even though nonfunctional requirements, such as security, span all functional requirements simultaneously, nonfunctional requirements have not been traditionally viewed as business processes. Because of this limited view of nonfunctional requirements, traditional user-centered analysis techniques have failed to provide an adequate framework for capturing nonfunctional requirements.

In addition to spanning multiple users and functional requirements, achieving one nonfunctional requirement is often in direct conflict with achieving another nonfunctional requirement. An example of this competitive environment is “security” versus “ease-of-use”. The more secure the system the less easy to use the system becomes to the user. Furthermore, functional and nonfunctional requirements differ in how they are achieved in the delivered system. Because nonfunctional requirements span all functional requirements and all users’ usage patterns, nonfunctional requirements span not only the software modules that provide functional requirements but also the underlying hardware – meaning that nonfunctional requirements emerge as a result of both the overall software and hardware architecture of the delivered system. As system architectures have moved from a monolithic architecture to distributed architectures with multiple and increasingly complex interaction between components, the importance of understanding nonfunctional requirements has increased dramatically. In order to deliver quality software, project leaders must understand the differences in how functional and nonfunctional requirements are captured and implemented.

Issue #3: Assign the overall project quality the highest priority and makes it everyone’s responsibility

Even before starting the development project, management must establish the overall tone or set of ground rules for the development project. They must define what will be valued, what will be measured, and what will be rewarded – in short, they must define a value system upon which the project will operate. The value system they define and the way they implement it is often referred to as the project’s “culture”. In addition to developing an overall project value system, management must show developers that management truly supports the value system. They do this by providing developers with the proper training, technology, supporting tools, and rewards necessary to deliver a quality project.

The most important rule is to establish an SQA function as an integral part of the SDLC and make the overall quality of the delivered software the highest priority for everyone involved in the project. By establishing an environment where the overall quality comes first, developers will be less likely to take short-cuts to meet individual schedules or budgets that could negative impact another part of the project. Instead, the goal of making quality everyone’s responsibility is to encourage everyone to work together to find and fix defects as early as possible. Given that the time and money required to fix defects increases exponentially as the project proceeds, finding and fixing defects as early as possible actually facilitates keeping the overall project on time and on budget.

For software developed using distributed architectural approaches, understanding inter-module dependencies and feature interactions becomes critical in ensuring the overall quality of the delivered software. By developing an appropriate set of overall quality metrics and communicating to the developers that they collectively will be held accountable for the project's overall quality, management will be able to focus the developers' energies on ensuring not only the quality of individual software modules, but also the quality of software functionality that spans multiple software modules. Focusing the developer's attention on the overall system architecture helps ensure that both the enterprise level business process requirements and nonfunctional requirements are properly architected in the delivered software.

Issue #4: Staff the SQA process with the right personnel and involve them from the beginning

For the SQA function to be successful, the SQA function must be staffed by personnel who understand the functional areas, the business processes, and the development approach as well as the interdependencies among all three. Organizations frequently move their newer or lower performing software developers to their SQA function. However, performing reviews and designing effective test cases that help ensure the quality of the overall architecture requires an understanding of complex distributed architectures, significant development experience, and a thorough understanding of business issues. Only senior personnel have the required training and experience.

In addition, successful SQA engineers must have the ability to pay attention to detail, follow an activity through to completion, and communicate effectively with both technical and business-oriented personnel. Given these requirements, only software developers who possess the right level of maturity and people skills can be effective in the SQA function. To ensure a success in the SQA function, organizations must staff the SQA function with the right personnel and involve them from the beginning.

Issue #5: Design SQA metrics using a goal-based framework

In addition to thoroughly understanding the project's functional and nonfunctional requirements, project leaders must also develop a set of overall project goals that capture both the development-oriented as well as business-oriented goals for the project. In general, business oriented goals tend to focus on improving the efficiency or effectiveness of a business process, while development oriented goals tend to focus on managing the productivity and quality issues associated with the software development process. To facilitate achieving these overall projects goals, management must define a set of project metrics that align with and support the project's goals.

Traditionally, software metrics focused exclusively on using metrics like the number of lines of codes (LOC) produced to measure programmer productivity and defects per thousands of LOC (KLOC) to measure program quality. As higher level programming languages and distributed architectures are evolving, software developers are realizing that LOC based metrics represent very crude surrogates for measuring the success of software development projects. Software developers are starting to utilize measurements of software productivity and quality that are intended to be independent of software development languages and architectures.

The major limitation of this approach has been the inability of these correlation-based metrics to link measures of software quality to actual software failures. A correlation between two metric values (such as a module's complexity and the number of defects found in it) does not provide an understanding of if, when, how, and why the delivered software will fail. The reason that correlation-based metrics can not provide adequate failure predictions is because correlation-based metrics do not account for cause and effect linkages between development approaches (i.e. architectures), defects, and failures. To overcome the limitations of using correlation-based metrics in isolation, project leaders must develop a framework for linking actual lower level tasks with overall project goals.

By focusing on the problem's solution first, an overall set of project goals can be established that provides a framework for determining whether the project has satisfied user requirements. Using these goals, project leaders can derive a set of questions whose purpose is to quantify the goals as completely as possible and provide a linkage between project goals and specific SQA characteristics (i.e. reliability, easy-of-use). Questions must cover both business-oriented issues (i.e. aspects of the product) and development-oriented issues (i.e. aspects of the software development process).

The major benefit of focusing on the critical aspects of the project first is that project leaders can develop metrics that provide the actual data needed to answer the questions that support the overall project goals. In addition to providing the obvious advantage of traceability between metrics and project goals, metrics developed using a goal-oriented framework are, by definition, more useful than are metrics developed in isolation. Goal-oriented metrics, therefore, tend to make more sense to software developers, and thus, are more likely to be used. To help ensure ownership of the developed metrics by the software development team, software developers should be involved in the metrics development process, and all project goals, metrics, and the linkages between them should be clearly communicated to the development team.

Issue #6: Choose appropriate models and tools to support the SQA process in every project phase

In order to build the right system, developers must accurately capture the right requirements. For developers, accurately capturing the requirements means asking the right questions. In order to ask the right questions, developers must thoroughly understand both the business processes and how the system's overall architecture will enable and support them. Critical to this understanding is the use of appropriate discovery techniques, models, and support tools for representing different kinds of requirements. Such a level of understanding must precede the writing of the requirements and test cases.

To ensure the accurate discovery of functional requirements, developers must capture and analyze dependencies among multiple stakeholders representing differing viewpoints, developers must determine whether these dependencies are task based, resource based, or goal based. In order to properly architect the nonfunctional requirements, developers must understand the usage patterns of all users across all functional requirements as well as the dependencies that exist among specific nonfunctional requirements (e.g. security versus ease-of-use). Given the differing nature of these requirements, developers must utilize a number of different but appropriate discovery techniques, models, and support tools to facilitate their understanding and specification of the requirements.

Although necessary, discovery techniques, models, and support tools in and of themselves will not ensure a successful SQA effort. However, the right techniques, models, and support tools used appropriately by well trained people can provide invaluable assistance to a successful SQA effort. Given the sheer size of today's development projects, support tools are needed just to manage the substantial amount of data generated during the development process.

SQA support tools can be classified into three categories: 1) planning and control, 2) test preparation and specification, and 3) test execution and analysis. The first category, planning and control, includes applications used for test planning, progress monitoring, configuration and release management, defect tracking and administration, and test management. The second category of SQA support tools includes requirements management, requirements traceability, and test plan link to requirements. The third and largest category, test execution and analysis, includes test capture and playback, load and stress testing, test coverage, simulators, test data generators, application analyzers, and debuggers. The selection of SQA support tools should be guided by the project's goals and development approach to ensure alignment between what the metrics measure and what the project must achieve.

Issue #7: Use SQA reviews to identify issues in every phase of the software development lifecycle

Given the need to design and build quality into the project's overall architecture, instead of waiting until implementation, SQA metrics must be utilized starting with the project's logical design phase. Therefore, SQA reviews must start with a design review in order to find defects as early as possible and to ensure that everyone associated with the project is included. During each of the subsequent SDLC phases, SQA reviews should be conducted to help identify any quality issues that arise as the project progresses.

In each SQA review, interactions among functional requirements must be checked to ensure that feature interactions are not present. Nonfunctional requirements must be checked to ensure that they perform as required under peak load conditions that simulate user usage patterns. By starting with a properly designed set of goal-based metrics, every SDLC phase can be monitored to ensure the overall quality of the product and productivity of the project. By performing SQA reviews in each SDLC phase, SQA reviews become a set of steps chained together to form a complete feedback loop, allowing project leaders to continuously monitor and improve the quality of the delivered software.

Issue #8: Use iterative build cycles to address quality issues that span features and seams

Since quality is an emergent property of the system, managing dependencies among all aspects of the system is critical to the overall success of the project. Only by simultaneously managing all of interdependencies that exist across functional areas, among business processes, and within the development approach can the overall quality of the project be properly managed. Because finding and managing quality issues introduced by interdependencies requires accessing interactions among system elements, actually assembling and testing the assembled system represents the only way to check the quality of the overall system. Therefore, the SQA process should use iterative build cycles to ensure that quality issues spanning architectural seams and features do not fall through the cracks.

Issue #9: Realize that SQA is a process unto itself that must be measured and continuously improved

Ultimately, what project leaders must understand about SQA is that SQA needs to be treated as a process unto itself. It is the process designed to ensure conformance to the project's requirements (i.e. the need to build the right system) as well as adherence to architectural development standards and practices (i.e. the need to build the system right) for all aspects of the delivered solution. To ensure that the SQA process is alive and performing well, management must monitor not only the quality of the delivered solutions, but the SQA process itself. This means that management must set measurable goals and establish feedback loops for the SQA process so it can be continuously measured and improved.

Summary

As companies increasingly rely on information technology for competitive advantage, software quality is becoming critically important. Many large software projects are delivered late, over budget, or without appropriate functionality. In addition, the costs of addressing software defects late in the software lifecycle or after implementation are exponentially greater than the costs of finding and fixing these problems earlier. By focusing on quality from the earliest project activities, most of the rework and associated costs can be avoided.

Implementing a successful SQA process involves choosing the right people, engaging them in the correct activities, and providing appropriate support tools. It involves understanding that quality must be designed and built into solutions, is everyone's responsibility, and is derived from a project's overall design. This will happen only when management commits to making quality the highest priority of the project. It requires involving the SQA team from the project's inception and staffing the SQA function with personnel capable of providing the leadership and skills necessary to ensure the overall project's quality. Personnel comprising the SQA team must be able to add value to design and code review sessions, understand multiple architectures, development, and deployment platforms, be at least as skilled as the senior project development personnel, and have access to the appropriate tools.

A successful SQA process uses reviews in every phase of the SDLC and iterative build cycles to address quality issues that span functionality and seams. By focusing on defect reduction from the start of a project, significant reductions in the total effort can be achieved through the avoidance of the rework/thrashing stage typical of many software development efforts. By providing feedback based upon goal-driven project metrics, the SQA process can be continuously measured and improved.

Suggestions for Future Research

This study contributes to SQA research by identifying critical issues in the organizational and process aspects of delivering quality software. Using these critical issues as a guide, SQA researchers can start to development a model of the SQA process by studying the linkages among issues. As an example, SQA researchers could start by studying the relationships among resource allocation issues, such as staffing and training (Issue 4), and the overall quality of the delivered software. Likewise, other SQA researchers could study the relationships among the design of software metrics (Issue 5), the developer's reward system (Issue 3), and the overall quality of the delivered software. In addition, SQA researchers could focus on the project management aspects of the model as suggested by the use of software reviews (Issue 7), iterative development cycles (Issue 8), and continuous process improvement of the SQA process itself (Issue 9) to determine how these impact the overall quality of the delivered software.

References

1. Eisenhardt, K. M. "Building theories from case study research," *Academy of Management Review*, Vol. 14, No. 4, pp. 532–550, 1989.
2. Fenton, N. E., Neil, M. "Software metrics: Successes, failures, and new directions," *The Journal of Systems and Software*, Vol. 47, No. 2-3, pp. 149-157, 1999.
3. Fox, C., Flakes, W. "The Quality Approach: Is It Delivering," *Communications of the ACM*, Vol. 40, No. 6, pp. 25-29, 1997.
4. Hall, T., Fenton, N. "Implementing effective software metrics programs," *IEEE Software*, Vol. 14, No. 2, pp. 55-64, 1997.
5. "The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards & Technology (NIST), United States Department of Commerce, May 2002.
6. Ravichandran, T., Arun, R., "Quality management in systems development: An organizational system perspective," *MIS Quarterly*, Vol. 24, No. 3, pp. 381-415, 2000.
7. Zultner, R. "TQM for Technical Teams," *Communications of the ACM*, Vol. 36, No. 10, pp. 79-91, 1993.

Notes

Notes