Let's Play 20 Questions: Tell Me About Your Organization's Quality Assurance and Testing

by Gary E. Mogyorodi, B.Math., M.B.A.

This article presents 20 questions used to determine and understand how mature the quality assurance and testing environments are within an organization. The questions are ordered to begin with those easiest to answer and become progressively more difficult for organizational compliance. This questionnaire again proves that an organization that has good quality assurance and testing practices tests early and tests often throughout its software development lifecycle.

Currently, I am a consultant specializing in requirements-based testing. I have been in the computing industry since 1973. In 1993, I began specializing in testing and quality assurance. I have been teaching students how to write testable requirements, how to perform requirements-based testing, and how to find ambiguities in their requirements since 1998. Each time I teach a class, I find that I ask students the same questions about quality assurance and the testing environments within their organizations. So, I have derived 20 questions that I use to try to understand how mature the quality assurance and testing environments are for each organization. After all, an organization that has good quality assurance and testing practices tests early and tests often throughout its software development life cycle.

I have ordered these questions so that, in my experience, the beginning questions are the easiest for an organization to comply with, and they become progressively harder to comply with as organizations proceed through the questionnaire.

One point is awarded for each of the first four questions, two points for each of the next set of four questions, three points for the next set, and so on. In my experience, the last four questions are the hardest for organizations to comply with and are worth five points each. A perfect score is 60 points; I would expect very few organizations to score perfectly.

The questionnaire can act as a self-appraisal of the quality assurance and testing activities within an organization. A sample scorecard is included in this article.

The questionnaire does not cover every aspect of software development and testing. The following topics are not included in the questionnaire:

- Project management.
- Risk management.
- Configuration management.
- Change management.
- Nonfunctional testing.

Following are each of the 20 questions along with the benefits of compliance and a score for each.

Question 1: Does your organization *consistently* create an environment for development and testing separate from production?
Benefits: A development and testing environment separate from production allows the organization to avoid making programming changes on the fly in production, and to develop and test software completely before it is promoted into production.
Score: One point.

Question 2: Does your organization *consistently* write functional requirements during the analysis phase of the software development life cycle?

Benefits: Functional requirements provide a description of *what* the expected behavior of the software should be before design begins so that the following occurs:

- The domain experts/users understand what will be built.
- The developers understand what to build.
- The testers understand what has to be tested to prove that the software behaves as expected.

Score: One point.

Question 3: Does your organization *consistently* perform user acceptance testing within its own test environment?
Benefits: User acceptance testing identifies defects before they get into production and gives the user community a chance to *kick the tires* on the system before it goes live. A system should operate in user acceptance testing just as it will operate for the users in production. This is not the time for finding serious functional defects that should have been eliminated in earlier phases of software development and testing.
Score: One point.

Question 4: Does your organization *consistently* perform reviews of the functional requirements for content?
Benefits: A review of functional requirements is a quality assurance step to improve the correctness and completeness of the requirements before software design begins. The review reduces the risk of rework that can occur if requirements are incorrect or missed.
Score: One point.

Question 5: Does your organization *consistently* include a design phase in the software development life cycle?
Benefits: A design phase organizes development and determines how the system will be able to deliver the requirements. A good design reduces ongoing software maintenance costs.
Score: Two points.

Question 6: Does your organization *consistently* perform system testing with an independent test team in a separate test environment?
Benefits: System testing provides an independent perspective (not the developers') on the quality of software development and finds many defects before they get to user acceptance testing. Software is promoted from development to system test to user acceptance test to production.
Score: Two points.

Question 7: Does your organization *consistently* use a defect tracking system?
Benefits: A defect tracking system allows an organization to identify defects in development, assign ownership to each defect, and to track the defects to resolution. A defect tracking system can report on the status of the software development and test effort from the standpoint of defects created, defects resolved, and outstanding defects.
Score: Two points.

Question 8: Does your organization *consistently* create test plans and test specifications for use in the system testing and user acceptance testing phases?
Benefits: A test plan identifies the test schedule, the testing tasks, and the test resources. A test specification identifies the scope of testing, the test strategy, the test environment, the test procedures, the test cases to be executed, and the assumptions made behind the test effort.
Score: Two points.

Question 9: Does your organization *consistently* use a test management system?
Benefits: A test management system manages and tracks the execution of tests against the software, making the testing effort more efficient. A test management system can report on the status of the software development and test effort from the standpoint of tests not executed, tests executed, and the results (pass/fail) for each executed test.
Score: Three points.

Question 10: Does your organization *consistently* perform regression testing during the system-testing phase?
Benefits: Regression testing is a technique to re-execute all tests each time there is a new software build. Regression testing identifies new defects that are introduced: it also identifies defects that were masked by defects corrected in the new release. Regression testing during the system-testing phase protects the asset value of the software.
Score: Three points.

Question 11: Does the organization *consistently* perform unit testing?
Benefit: The programmer of a particular code module(s) typically performs unit testing, which finds defects earlier in the development life cycle than system testing and user acceptance testing.
Score: Three points.

Question 12: Does your organization *consistently* define entry criteria and exit-completion criteria for each phase of the test process?
Benefits: Entry criteria and exit-completion criteria formalize each phase of the test process. Software cannot enter a test phase without meeting the entry criteria for that test phase. Software cannot leave a test phase without meeting the exit-completion criteria for that test phase.
Score: Three points.

Question 13: Does your organization *consistently* perform design reviews?
Benefits: A design review is a quality assurance step that improves the quality of the software effort by finding potential defects within the design phase before coding begins. The design must be robust enough to show how it will deliver the requirements.
Score: Four points.

Question 14: Does your organization *consistently* use a requirements management system?
Benefits: A requirements management system makes requirements management more efficient than paper documents by providing the following:

- Ownership of each requirement to a resource.
- Collaboration among resources to define requirements.

- Traceability among requirements.
- Traceability between requirements and use cases, test cases, test results, and software components.

Score: Four points.

Question 15: Does your organization *consistently* perform ambiguity reviews on requirements?
Benefits: An ambiguity review is a test of the functional requirements to insure that requirements are written in a clear, concise, unambiguous, and testable manner. An ambiguity review is performed after requirements are drafted but before requirements are reviewed and signed off for content: correctness and completeness. An ambiguity review improves the requirements' quality and helps avoid defects from being entered at the analysis phase of the software development life cycle, thereby propagating throughout the remainder of the software development life cycle.
Score: Four points.

Question 16: Does your organization *consistently* perform post-implementation audits and act on the results?
Benefits: A post-implementation audit is a quality assurance step that compares the actual results of each project (objectives, schedule, resources, budget, functionality, and quality) with the project objectives, and then acts on the post-implementation audit results. Post-implementation audits provide the organization with lessons learned about the software development life cycle.
Score: Four points.

Question 17: Does your organization *consistently* apply a formal test methodology?
Benefits: A formal test methodology provides the standards and procedures for performing testing throughout the software development life cycle. A test methodology provides a description of the test effort throughout each phase of testing, i.e., unit, system, and user acceptance, and describes the techniques and tools needed to make testing an efficient and manageable effort.
Score: Five points.

Question 18: Does your organization *consistently* use a rigorous approach to test-case design, e.g., cause-effect graphing?
Benefits: Equivalence class partitioning, boundary analysis, and cause-effect graphing are some of the techniques used for designing test cases. Cause-effect graphing is a technique to design the necessary and sufficient set of test cases that cover 100 percent of the functional requirements. The requirements' authors, domain experts, and developers review these test cases to validate that the right requirements are being documented. This step satisfies the definition of validation in the Capability Maturity Model® Integration℠ (CMMI℠). Each project role gains the same understanding of the expected behavior of the software before it is developed, thereby reducing the risk of rework occurring throughout the software development life cycle. Once the software is ready to be tested, the test cases are then used to verify that the software behaves the way it is expected in the requirements. This step satisfies the definition of verification in the CMMI.
Score: Five points.

Question 19: Does your organization *consistently* perform code reviews?

Benefits: A code review is a quality assurance step that identifies defects in the software before it moves into system testing. Code reviews improve the maintainability of code. Code reviews help share best practices in software development among the development team.
Score: Five points.

Question 20: Does your organization *consistently* use a code coverage monitor?
Benefits: A code coverage monitor measures the amount of code that is executed during a test. A code coverage monitor helps identify lines of code that would go into production without being tested. By executing a code coverage monitor during a test, the risk of promoting untested code to production is minimized. This reduces the risk of a software failure occurring under untested conditions in production.
Score: Five points.

Question Number	Topic	Point Score If You Comply	Your Score
1	**Create a separate development and test environment.**	1	
2	**Write functional requirements.**	1	
3	**Perform user acceptance testing.**	1	
4	**Review functional requirements for content.**	1	
5	**Include a design phase in the SDLC.**	2	
6	**Perform system testing.**	2	
7	**Use a defect tracking system.**	2	
8	**Create test plans and test specifications.**	2	
9	**Use a test management system.**	3	
10	**Perform regression testing.**	3	
11	**Perform unit testing.**	3	
12	**Define entry criteria and exit-completion criteria.**	3	
13	**Perform design reviews.**	4	
14	**Use a requirements management system.**	4	
15	**Review functional requirements for ambiguity.**	4	

16	**Perform post-implementation audits.**	4	
17	**Apply a formal test methodology.**	5	
18	**Perform cause-effect graphing.**	5	
19	**Perform code reviews.**	5	
20	**Use a code coverage monitor.**	5	
	Total final score.	60	

Table 1: *Questionnaire Scorecard*

Results

Now let us review your score. The exact score is not important, but higher scores are better than lower scores. If I were presenting this topic at a conference, I would give away a T-shirt to the organization with the highest score.

More importantly, look at the questions to which you answered *no*. Are you familiar with each topic being discussed? If not, take the time to get acquainted with the issue identified in the question and discover the benefits that your organization may be missing. If you are familiar with the topic, ask yourself what is holding you back from implementing that process step within your organization. Why is it you are not achieving the benefits described, since each question addresses an issue that would improve your quality assurance and testing?

Generally speaking, organizations tend to implement quality assurance and testing process steps at the end of the software development life cycle rather than at the beginning of it. They organize themselves to wait for software to fail, and then go back and rework it. Rework adds time to the project schedule that is often not accounted for in a project plan, so a project with rework often runs longer than expected.

The quality assurance and testing steps that occur earliest in the software development life cycle are the most beneficial to the organization. It is less expensive to eliminate defects early rather than wait to eliminate the same defects in a later phase. By eliminating defects early, an organization can avoid defects from propagating through development, and thereby reduce the risk of rework occurring. If rework can be eliminated, the organization has a better chance of completing its projects on time.

About the Author

Gary E. Mogyorodi is the principal consultant with Software Testing Services , consulting, training, and mentoring in software testing, and specializing in requirements-based testing. He has more than 30 years of experience in the computing industry and has presented at numerous conferences, including the Software Technology Conference, Software Quality Forum, Toronto SPIN, Starwest, and more. Mr. Mogyorodi has a bachelor's degree in mathematics from the University of Waterloo, and a master's degree in business administration from McMaster University.

Phone (905) 813-7937
Email: garym@softestserv.ca

Glossary of Terms

Ambiguity Review: A technique for identifying requirements that are unclear, not concise, ambiguous, and untestable.

Cause-Effect Graphing: A test-case design technique in which test cases are designed by drawing cause-effect graphs. A cause-effect graph is a graphical representation of the inputs or stimuli (causes) with their associated outputs (effects), and the relationships between those causes and effects.

CMMI: The Capability Maturity Model® (CMM®) Integration℠ (CMMI℠) is a framework to provide guidance for improving an organization's processes and the ability to manage the development, acquisition, and maintenance of products and services.

Code Coverage: An analysis method that determines which parts of the software have been executed (covered) by the test case suite and which parts have not been executed and therefore may require additional attention.

Code Review: A formal review of the code to ensure coding standards are followed and to validate that source code functionality has been achieved. Best practices can be shared among coding team members during a code review to improve the maintainability of code.

Defect Tracking System: A tool that tracks the status of defects from their identification to their resolution.

Functional Requirements: The document or repository that describes in detail the characteristics of the product with regard to its intended capability, i.e., *what* the product should do.

Post-Implementation Audit: A review of the project following its completion to assess actual results, including whether the project objectives were met. Other factors that are assessed include the project schedule, the budget, the resources used, the software development effort, and user acceptance of the delivered solution.

Quality Assurance: A planned and systematic approach to the evaluation of the quality of software product standards, processes, and procedures. It is also the evaluation of the adherence to these same product standards, processes, and procedures.

Regression Testing: A technique for retesting previously tested programs following a modification to software to ensure that faults have not been introduced or uncovered as a result of the changes made.

Requirements Management System: A tool that captures functional requirements, shows the relationships among functional requirements, and tracks any changes made to them.

System Testing: The process of testing an integrated system to verify that it meets specified requirements. System testing is performed by an independent testing team.

Test Case: A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Test Management System: A tool that controls the sequence of test cases performed across many program modules. The tool manages the execution of strings of tests, both manual and automated.

Test Methodology: A description of the activities and tasks for the content and placement of software quality steps in the software development life cycle.

Test Plan: A test plan identifies the test schedule, the testing tasks, and the test resources.

Test Specification: A record of the test process detailing the scope of testing, the degree of tester independence, the assumptions behind testing, the test environment, the test case design techniques, test measurement techniques to be used, and the rationale for their choice.

Unit Testing: The testing of individual software components by the developer who created them.

(User) Acceptance Testing: Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component. For organizations that create commercial software, user acceptance testing is called beta testing.