

Getting The Most Out Of Your Testing Efforts!

1.0 Introduction

No one intentionally makes bad quality software! But, compared to the makers and users of products of other engineering disciplines, confidence levels of developers and users of software on its quality is far too low. A lot has been researched, written, preached and practiced with the intention of ensuring development of quality software. One of the critical practices is testing the software as a conscious and continuous effort.

Yet, even the best-tested software is often found to throw up surprises. This leads to nagging doubts on quality of the software and even the testing efforts that were undertaken to ensure the quality. Some of the common refrains are: "We never thought we need to test that as well", "We never expected a problem there", "It was working very well", "We need to do a complete testing", "I thought, we have good testers", "We should actually completely automate testing", "Automating the testing does not work". These are only partial reflections of the hidden challenge. The real challenge, often, gets the attention only after the damage is done, very much like the insurance policies whose value is realized only when the disaster happens.

2.0 Call for a closer look

For an industry which is increasingly adopting engineering principles, this actually calls for an objective analysis. The very verb 'test' implies an object being tested (which is the software) and an expectation (which the software is expected to fulfill). Expectation varies for every produce, its every possible usages, and its operating environments. For any non-trivial software, its usage evolves, even as it gets used, by way of having to support more users, additional functionality, improving user interface and so on. We almost expect the software used with a specific version of Internet Explorer to work in the same way with another version of the same or even when used with another browser like Netscape coming from another vendor. We almost expect the software used in Windows NT to work in the same way in Windows 2000 or Windows XP. For this to be a reality, the software is to be designed and developed for those expectation; tested to ensure those expectations are met. Testing indeed starts with a closer look on the expectations. These expectations are typically documented as functionality or requirements of the software.

3.0 Perspective differences

We perceive the software functionality as being achieved by execution of a few thousands or millions lines of code we have written explicitly (code that is immediately related to the application). But, these internally rely on many other hidden elements like APIs, system function calls and even hardware interfaces. It is such a loose collaboration that actually helps the smooth functioning of the software, helping it to meet the expectations. Consequently, every new usage, and every new environment, has a potential to throw up surprises (called defect or bug, in software parlance). It is not very uncommon to find otherwise well-functioning software to malfunction, when used in a computer which has some new virus or a different version of a DLL introduced into the computer during installation or upgrading of an apparently unrelated software.

Every (perceived) defect is a manifestation of such difference between the expectations and the

product capabilities seen from user's perspective as he/she may not distinguish between software and the environment it operates in. It could be a showstopper (eg. application crashing reporting a missing DLL or because of using a wrong version of a DLL); an irritant that could be resolved by some workaround (eg. setting classpath to a specific version of JRE) or some minor difference (eg. not printing page no in every page of multi-page reports). Also, what is perceived by one user as an irritant could be a showstopper to another. For example, setting a classpath is an irritant for a technically knowledgeable person while it may prove to be a show stopper for a novice. The latter may not even be aware that he is extending the application beyond the intended environment or usage even when he is actually doing it.

All this boils down to bridging the gap between user's expectations and the product's capabilities. Some of the user's expectations, even as its get evolved, is recorded into requirement documentations while some are often left assumed. All assumptions made during requirements elicitation, design and implementation is validated (rather than analyzed or assumed) with demonstrated capabilities during the testing phase. This implies that depending on the criticality of the work and risks associated, the testing may need to go an extra mile to cover even what is left assumed. This emphasizes the need for undertaking the testing as a controlled effort while retaining creative exploration.

4.0 Towards solution

There are quite a few practices in testing that could actually help compliment other activities in software development. Testing helps in ascertaining what works (eg. a specific functionality, as tested on a specific environment and in a specific sequence) and what does not work (eg. a specific usage that failed). It divides the entire unknown solution space to be addressed by the software to what definitely works, what does not work and what could possibly work. A new usage or environment actually represents the third category so long as it is not tested. If it is critical to business, further testing efforts should be focused towards covering more and more of this category. It is also important to identify and document the environment in which software functionality was found to be achieved so that it could be shared with the user.

It is important to understand the magnitude of the challenge and, yet, not be intimidated by that. It helps to understand priorities and work towards addressing those areas, with planned effort. A closer look reveals that some of the potential usages are truly esoteric, some are futuristic, some are desirable and some are very critical. Also, the impact of not meeting the expectation is also not the same for every software, every usage, every feature, and every environment. However, these vary from product to product, solution to solution, and context to context. It is essential to capture this notion into testing even as we plan for testing and it is important to drive the testing on those lines. Every variation observed during the testing or during the actual usage must be studied against these specifications. Defect reported on an esoteric usage should not be classified along with defect on a critical functionality in an intended usage.

Based on some of our experience in helping customers in testing various projects, we recommend a few generic best practices for more effective testing. Though overnight miracles are far from reality and dependence on experience of the experts would continue, careful planning specific to the project would help in progressively improving the confidence levels of developers and users. Also, a close monitoring of the testing effort against the plan with continuous and incremental improvement based on the test results would help improve the overall testing process.

4.0 Best Practices

1. Plan your testing
 - ✓ Identify critical functional and non-functional requirements that software is expected meet
 - ✓ Identify and avoid potential risks and problems
 - ✓ Identify, document and avoid violations
 - ✓ Estimate based on metrics, and continuously revisit & revise the estimate
 - ✓ Closely & continuously monitor adherence to the plan

2. Establish & verify the objective
 - ✓ Identify the testing techniques & approaches to be adopted (eg. testability of component)
 - ✓ Ensure need, availability, and appropriate use of the resources (eg. skilled manpower)
 - ✓ Establish the effectiveness of the techniques & approaches, with at least a candidate case
 - ✓ Do not introduce changes without assessing impacts, especially during the later stages

3. Test as much as possible
 - ✓ As early as possible
 - ✓ As many aspects as possible
 - ✓ As many levels as possible
 - ✓ As many ways as possible
 - ✓ As frequently as possible
 - ✓ As systematically as possible

4. Manage the defects
 - ✓ Classify the defects
 - ✓ Document the defects
 - ✓ Analyze the impact of the defects
 - ✓ Communicate the defects
 - ✓ Resolve the defects

5. Appropriate level of documentation
 - ✓ Requirements
 - ✓ Plan
 - ✓ Efforts
 - ✓ Results

6. Traceability & impact analysis
 - ✓ Business needs to testing
 - ✓ Requirements to testing
 - ✓ Design to testing
 - ✓ Changes to testing Efforts
 - ✓ Defects to fault in design

