Fundamentals of **Quality Assurance**

WHITEPAPER

ZEPHYR
Real-Time Test Management

Table of contents:

What Is **Quality?**

As a Quality Assurance Manager or Analyst, our very title tells us what our role is – we "assure quality". It seems to be such a simple concept but one that is often misunderstood or misinterpreted by many in a software development and testing organization. Hiding behind those words is a world of interpretation and nuance that can be clarified and defined further.

Let's start with a very basic concept – what exactly is Quality? Typical definitions include such phrases as "the degree of excellence of something", or "how good or bad something is" or "a high level of value".

When we apply those concepts to building and testing software we can surmise that software with a "high level of value" and "a degree of excellence" would be bug-free, do the job it was designed to do, and have a high degree of stability and value to the user.

Bug free? Is there really such a thing?

There has been much debate over the concept of "bug free" software. Let's take a moment and explore what it means to be "bug free", and how realistic that concept is in today's software world.

#include <stdio.h>

```
main( )
{
      printf("hello, world\n");
}
```

The "Hello World" program above is usually the first program beginning coders write as they are learning a new language. It is hard to argue that the "Hello World" program above is not bug free. It works perfectly every time you run it, and it always prints exactly what it is supposed to – "hello, world". So it meets the definition of "bug free" and "a degree of excellence". However, this is a trivial example and one that fails the "high level of value" test, as it does not produce any real and valuable output (except for introducing new programmers to their first program).

Software in today's world is complex, and that complexity is increasing. As far back as 1970 E.W Dijkstra discussed the idea of complexity and software bugs in a paper titled "Structured Programming". One of the many corollaries established by Dijkstra is that "Program testing can be used to show the presence of bugs, but never their absence"[1]. To summarize Dijkstra's thinking; without testing every possible outcome of a program we cannot be sure that a bug does not exist. And complex programs can consist of thousands or millions of possible code paths and outcomes. Testing each and every outcome of complex software would be prohibitive in terms of cost and time.

It's come to light in recent years that something often perceived as simple, such as the engine management or 'infotainment' software that runs in a modern car, can contain between 1M and 100M lines of code[2].

So it appears that we cannot necessarily test every possible outcome and every possible code path in a piece of software that's been given to us to test, because QA teams are generally charged with testing (the quality component) on a schedule (the time component) and within a budget (the cost component). A testing organization needs to balance these three components, quality, cost, and time, in order to be successful and many organizations struggle to get this balance correct. We can visualize this struggle with a modified version of the traditional "Fast, Good, Cheap" diagram that's popular in project management circles. However our version will be labeled "Fast, Low Cost, Highest Quality"

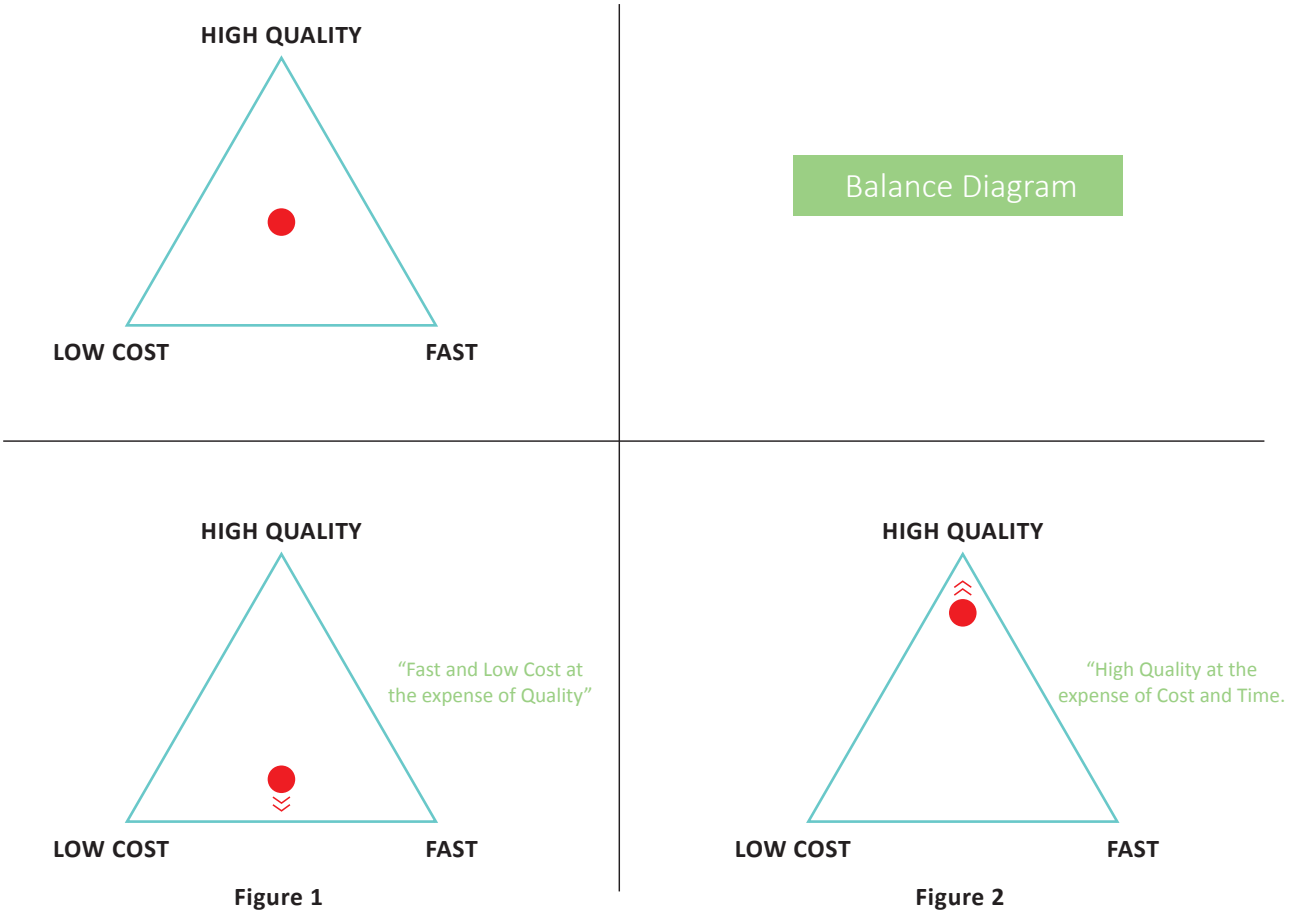


Figure 1 **Figure 2**

In this diagram, the dot (or balance point) close to any one of the three vertices indicates that vertex takes priority over the other two. A dot directly in the center of the triangle indicates a balance between Cost, Speed and Quality with each having equal weight or priority in the project. For example, Figure 1 below indicates an emphasis on Speed and low Cost, with a lowered emphasis on Quality. Figure 2 shows an emphasis on Quality, with a correspondingly increased timeline and cost factor.

In viewing the project balance with this type of diagram, it becomes apparent that being as bug free as is possible means making sacrifices in time and cost (increasing both).

One thing that is critical to note, is that this balance will be different depending on the purpose behind the software. For example, hosted web based applications for sharing photos with friends can be updated quickly and easily if a bug is found. There may be no real downtime involved and the impact of a bug in the code might be a bad user experience, or a broken image link on the web page. While user experience issues cannot be considered trivial for the company, it is not something that would necessarily have a major effect on people's lives or the company's financial future. So our balance in this scenario may lean more towards the "Low Cost" and "Fast" sides of the triangle, with a higher tolerance for bugs as they can be squashed relatively easily.

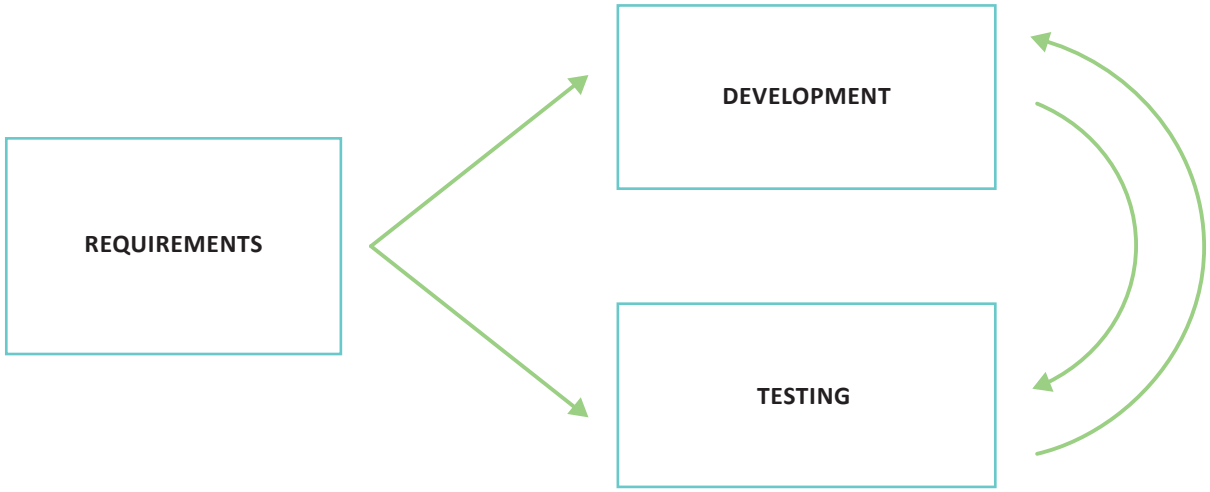
However, consider something like the flight control software for a Boeing 787, or the code that's running in a medical device such as an implanted heart defibrillator/ pacemaker. In both of these examples, bugs in the code which manifest as incorrect behavior can cause more than just a bad user experience – it could endanger lives and cause extreme financial burdens for the company. In addition, delivering software updates is very difficult and expensive (if not impossible, in the case of implanted devices). So in this case the balance point should be much closer to the "Highest Quality" side of the triangle at the expense of "Fast" and "Low Cost".

So let's summarize the attributes of Quality; strive to ensure that the software under test is as bug free as possible, while balancing the needs of the business around Cost and Timelines.

What is **the Scope of our testing?**

In the discussion above, it was determined that the definition of Quality is not absolute. It has a 'fuzzy' component to it based on time and cost factors. But even if time and cost were not factors, how would we know *what* to test? This may seem like an obvious answer; we test the application. But with the complexity of today's applications QA needs to know what's been built (and how), what the client want's the software to do and where their business priorities lie, and plan testing accordingly. This planning effort by the QA team turn will determine the "scope" of the test effort.

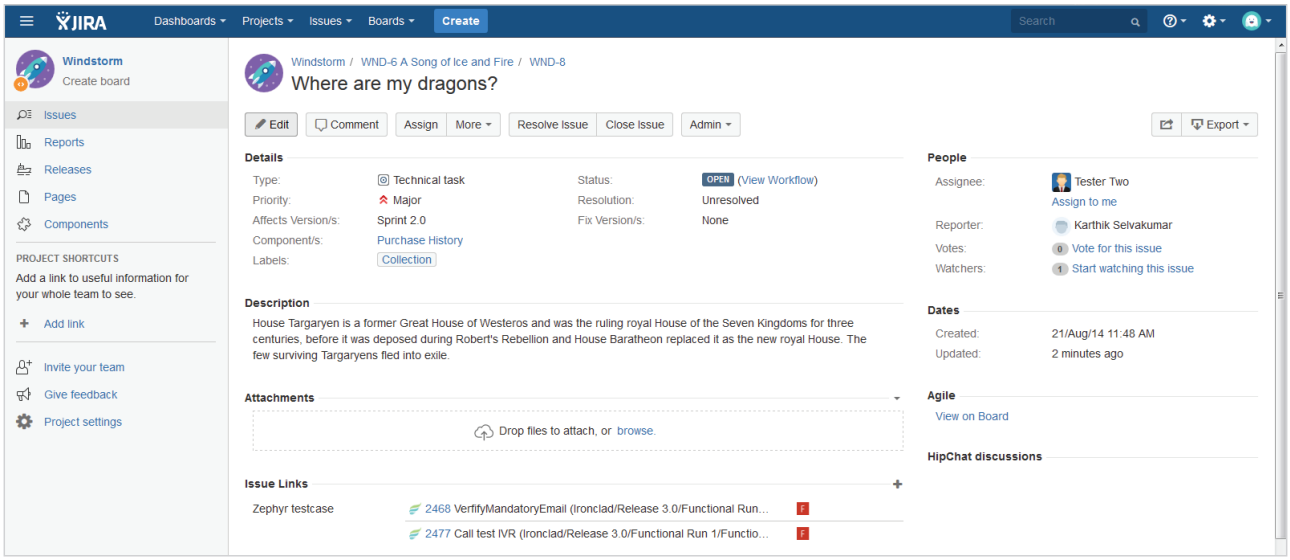
A good test plan begins with a set of requirements – documentation of some sort (formal requirements, user stories, etc.) that describes in detail the functionality that will be built into the software. For the QA team, the requirements documentation is the basis for planning the testing effort and authoring test cases that will ensure that the team is validating the full functionality of the system. The requirements need to be 'specific', meaning they should be written at a level of detail that allows the tester to author test cases that can validate each individual piece of functionality within the software under test, and 'testable' meaning that the end result of executing some set of functionality is deterministic and quantifiable. If the requirements are not specific enough (too high level, for example), the QA analyst may not be able determine all aspects of a piece of functionality and miss some critical edge cases. If the requirements are not 'testable', then the outcomes of the functionality is not clearly stated and thus cannot be fully validated.



Review and Agree on Requirements

Requirements drive the development team as well, indicating to the development team what functionality needs to be built out to satisfy the client or business need. Since requirements drive the work done by both the development team and the QA team, they need to satisfy the needs of both teams. It is prudent that both teams review (and approve, if that is part of the process) the requirements documentation early in the product's design phase, or at the beginning of each sprint. In many cases this review process can lead to discussions around requirements that resolves inherent ambiguity that would have been an issue once

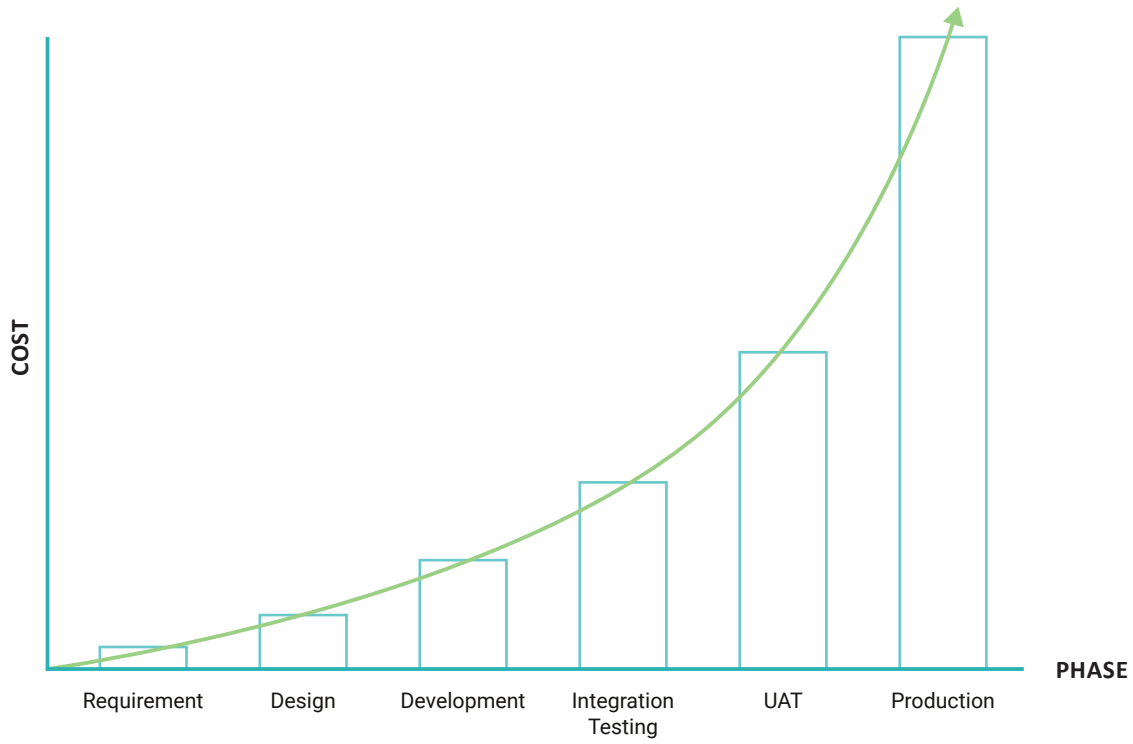
development and testing had started. In my experience, I have found that agreement on the scope and detail of the requirements documentation by the QA and development teams leads to a much smoother development and testing cycles (sprints) through out the project development and testing.



For example the above image illustrates a specific project requirement, showing the specific test cases linked to that requirement from Zephyr Test Manager. So requirements determine the scope of the testing – if you have 100 requirements, then your testing 'scope' is to verify all 100 of those requirements are implemented and functioning correctly (this is a bit of a simplistic example, but let's go with it for illustrative purposes). Having this scope defined allows the team to determine another critical factor in the creation of the product – how to define when we are 'done'. While this may not seem like a major issue, without a defined scope and a formal definition of what it means to be 'done' it is possible for development and testing to become an endless cycle of test/fix/retest as new areas of functionality are 'discovered' as the testing progresses. If, however, you have 250 test cases covering the 100 requirements (that have been defined, reviewed, and agreed upon by SA, development, QA, and the project owner) you can easily track progress to timelines, and declare testing to be completed once the 250 tests have been run successfully with minimal (and accepted) defects remaining.

How much does a bug cost?

This is a well studied topic in computer science education and in industry, with published results back as far as 1981 (Boehm). In a more recent 2004 NASA study titled "Error Cost Escalation Through the Project Life Cycle" (3) (Stecklein, Dabney, et. al.), the authors assigns an arbitrary value of "1 unit" to the amount of time and money it takes to fix a bug found during the requirements phase of a project. He then determines that the cost of fixing the bug rises in an exponential manner as the project continues to move forward through the design phase, build phase, integration and testing phase, and operations or production phase. At the production phase of the project the cost to fix the bug was between 29 and 1500 "units". The conclusion that can be drawn from this study is that the cost of finding and fixing a bug in production can be hundreds of times more expensive than if that bug were caught in requirements, design, development, or testing.



Cost of fixing a bug, by Phase

The QA team can assure that bugs are identified in the early phases of the project in the following ways:

1. The QA team can have a useful and productive role in the design discussions and determinations by having a seat at the table and considering these discussions through a 'testing' lense. Questions and open discussion about the design of the software can lead to changes that prevent bugs that may manifest themselves further downstream in the software development lifecycle.

2. The QA team should be involved in reviewing the requirements to ensure they are 'specific' and 'testable' prior to development starting. This will ensure that the requirements are not ambiguous and are properly testable by the QA team (and specific enough to drive precise development of the feature set).

3. Reviewing the requirements in concert with the development team will lead to a common understanding of the requirements, which in turn prevents different interpretations by development and the testing team. Differing interpretations of the intention of design components frequently shows up as bugs in later phases of the project.

All of these points lead to the discovery of issues much earlier in the product development lifecycle, which in turn saves the organization time and money.

Who owns the overall Quality of the product?

The QA team tests the software and measures the quality against the requirements. But does QA really own the Quality of the product? Who makes the all critical 'go/no go' decision once the product nears the production phase?

While the QA team quantifies the quality of the code through the discovery and classification of issues, it should not hold the responsibility for the decision to release or not to release the software to production. This power to make this decision should be in the hands of the project stakeholder -- generally the client and internal client representative, or the internal stakeholder for the project.

As discussed early on, software is rarely bug-free in today's complex environments. And balancing time, cost, and quality requires a broader view of the project than the QA team generally has. The project stakeholder or client/client representative are in the position to balance time, cost, revenue, client relationship factors, and other business side risks against the current level of issues found in the current state of the project. QA's responsibility is to provide as much accurate and timely information about the current quality state of the project under test so the stakeholder(s) can make as accurate and timely a decision as possible regarding the direction of the project.

Which leads us to…

How does the **QA team provide value to the project**, other than finding bugs?

The first point to consider and a common misconception heard among management and software professionals is that *QA does not fix bugs*. The role of the *individual* QA Analyst is to

1. Validate the functionality of the software under test according to the requirements specification.
2. Open bug tickets for functionality that does not match the specifications.
3. Helping investigate the root cause and providing information to the development team.
4. Determine initial priority and criticality of the issue.
5. Own the issue as it is discussed and worked by the development team.
6. Retest the issue once the development team has fixed and unit tested the issue.

Finding and quantifying issues is the role of QA, fixing issues is the role of development team.

As a *unit*, the QA Team provides a much broader value proposition to the project, project managers, and upper management. As discussed earlier, the team provides requirements and design validation early on in the project. The benefit of this validation is the identification and remediation of issues in the requirements and design phases of the project, allowing for a quicker and less costly development, testing, UAT, and production transition phases.

Data is **Everything**

As the QA team progresses through the testing phase of the project it produces vast amounts of data that can be beneficial as an aid to guide the project's process and predict outcomes regarding timelines, cost factors, and resourcing.

The QA Lead should track and provide information about the test cases and the bugs found. Good test case management system (such as Zephyr's Enterprise Test Management suite) can provide easily configurable reporting functions that can provide regular updates on test case progress, bug status and priority, and trouble spots within the development and test effort. As illustrated below, this data provides a wealth of information about to the project team and management, which is then used to in risk mitigation, expectation setting, and troubleshooting of the development and testing process.

Test Case Data

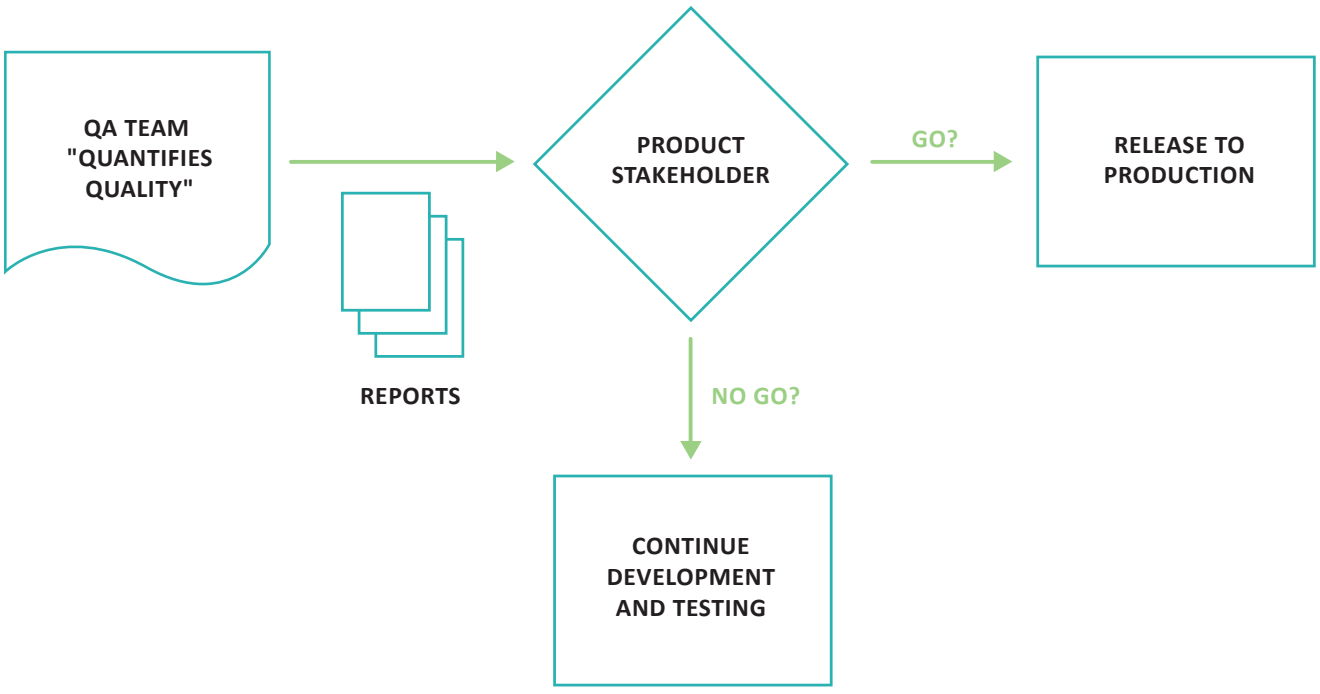
A well planned test effort will be able to provide (either at the project level, or the sprint level) a reasonably accurate indication of the number of tests that the team has authored on each component (or during each sprint, if it's an agile project). Within the QA team this data will provide the QA Lead with a measure by which he/she can manage the workload (shuffling resources among components to meet internal schedules, for example). This information, combined with test execution rates per tester and component, will allow the Lead to efficiently plan and manage tasks and assignments to provide for a successful effort.

As the project moves through it's phases the team tracks progress using the test case completion rates. This information (along with bug information, discussed below) adds to a critical view of the project's health. The person in the project management role uses test case progress and completion information to predict completion dates and manage schedule expectations, as well as monitoring cost and setting appropriate expectations with the project stakeholder.

If the test case management tools provides information on re-run test cases (tests run multiple times due to retests of bug fixes) combined bug retest failure data, the team can create a more in-depth view of the project progress and possible issues.

Bug Data

Bug tracking software (such as Zephyr's Enterprise Test Management , Bugzilla, JIRA, etc.) is an essential tool for a testing effort of any scope. Software of this type allows the project team to capture critical data about functional issues found in the software, including metadata around the component, release, or functional area the bug applies to. Good bug tracking software will also provide a workflow tool for managing bugs throughout their lifecycle, allowing the tester to assign a bug to a developer for comment and fixing, and then the developer can reassign back to the tester for retest once it is complete.

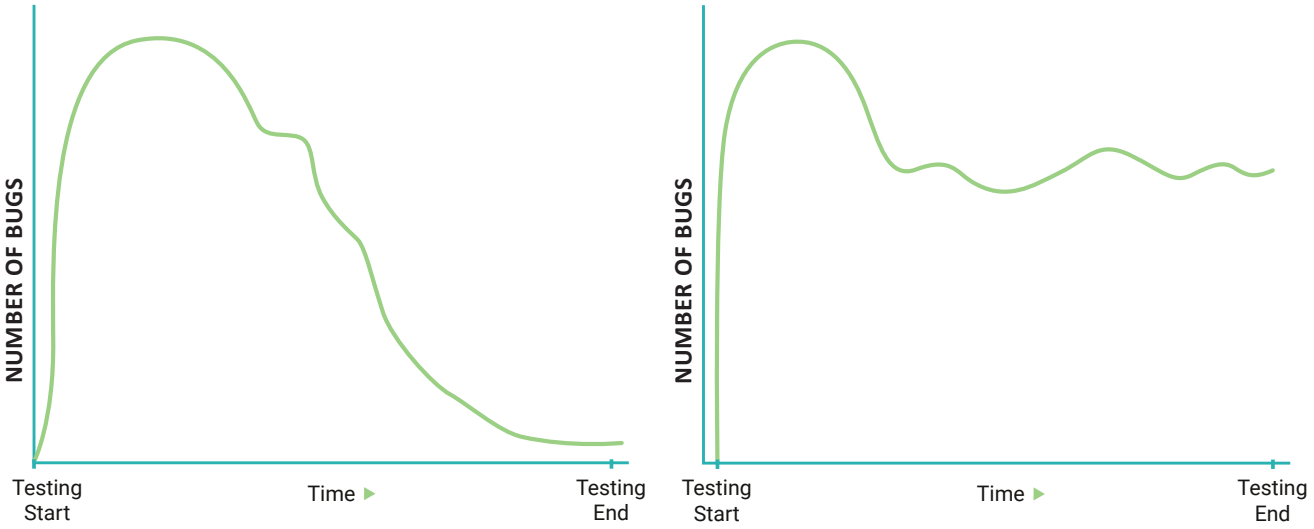


The workflows are generally configurable and ensure that bugs are properly triaged, assigned, and do not get lost in the system.

At a minimum, a good bug tracking system should allow the test team to report on the following attributes:

number of bugs opened	number of bugs closed	bug by priority
bug by severity	bugs re-opened	bug retests that failed

This basic level of information can provide a project level health report during the project lifecycle. As a project is initiated it is normal to see a large number of defects opened, and as the project progresses the number of defects opened should decrease. At the end of the project, the focus should be on fixing the few remaining defects with a very low number of new issues found. For example, in a healthy testing effort we should see a spike in the number and severity of bugs opened earlier in the project (and with every new feature released to QA), but as time progresses the number and severity of the bugs should continue to decline as illustrated below.



Healthy Project Profile

Unhealthy Project Profile

Monitoring bugs in this fashion can alert a project manager or stakeholder to an issue with the project if, for example, the number or severity of bugs is not declining as the project nears the end date.

The charts above illustrate the bug closure rate for a healthy project (left) and an unhealthy project (right). Data like this can trigger intervention by the team Leads to determine why the bug rate is not dropping.

This type of bug open rate profile can be cause by things like:

1. Requirements instability – changes to requirements late in the development and testing of the project
2. Over complex design – an overly complex design can cause a high number of continuing issues because any changes to extremely complex code is likely to cause unexpected side effects
3. Lack of quality unit testing – code that comes to QA without proper unit testing manifests more bugs than properly unit tested code
4. Poor development quality – developers that are under undue pressure, or are being pushed beyond their abilities

Additional reporting information produced by the QA team can help project management and project stakeholders to determine the cause of an unhealthy project. Tracking the re-opening of bugs, and bugs that have failed retest can (when combined with test case data around tests that have failed multiple times) give insights into areas of the software where development is struggling. Having component information associated with these failures can point to a possible design issue that's causing more bugs than average, or even a development team that's skimping on the unit testing properly, leading to bugs in QA. This kind of data can be invaluable when a project is struggling to stay on time and on budget, as it can guide us to design flaws, implementation flaws, or even to teams that are not following process or functioning properly.

Summary

Assuring Quality is a complex and nuanced concept that is often not well understood. Quality is a multi-dimensional measure that must take into consideration the cost and duration in determining the level of quality a team must strive for. The scope of a testing effort is driven by a firm and agreed-upon set of requirements or user stories – with out a firm scope based on requirements risks to the project's quality and schedule increase.

Quality is not owned by QA, but by the project stakeholder(s), and the QA team's responsibility is to quantify the state of the software, providing data to the stakeholder such that they can make decisions about a project's forward progression with full knowledge of the risks involved. In addition to finding bugs, the QA team provides critical information to the project stakeholder(s) and managers that allow guidance and changes in direction based on the current health profile of the project. Providing this information requires a good tool suite which can minimize the overhead necessary 'mine' for these reports.

With the insights and framework described in this white paper, a Quality Assurance manager can ensure that the team's contribution to a software project assures business success.

About Zephyr

Zephyr is a leading provider of quality management solutions, powering intelligent DevTestOps for more than 11,000 global customers across 100 countries. Project teams and enterprises of all sizes use Zephyr's products to enable continuous testing throughout their entire software delivery pipeline to release higher quality software, faster. Zephyr is headquartered in San Jose, CA with offices in King of Prussia, PA, Europe and India. For more information, please visit www.getzephyr.com.

ZEPHYR
Real-Time Test Management

Contact Zephyr Today!
sales@getzephyr.com
www.getzephyr.com
+1-510-400-8656