RBCS

TIME TESTED.
TESTING IMPROVED.
www.RBCS-US.com

Fitting Testing within an Organization

Rex Black, President, RBCS, Inc.

Back in the early days of computing, there was no such specific, separately identifiable activity known as "testing." Developers debugged their code, and that was usually intertwined with some unit testing task. That didn't work. My first job was as a programmer, and where I worked, this was exactly how we approached quality assurance. It was a quality disaster.

This approach by itself still doesn't work. It does not work for those throwback, Neanderthal organizations that rely on this approach entirely. It does not work for those cutting-edge companies that think some fancy language or process or tool has solved the software quality problem at last. It does not work for anyone, unless we define "work" as "shift most costs of poor quality onto end users who are too stupid to know better or trapped in a monopolistic market without any real choice." And those organizations that rely on stupid users or a monopoly position had better hope they are right.

With the widespread advent of independent test teams in the late 1980s and early 1990s, we saw improvements. I was working in an independent test lab in the late 1980s and as a test manager in the early 1990s, and we made great strides. However, we also saw the emergence of a new dysfunction, the "hurl it over the wall to the test guys and hold them responsible for delivered quality" mistake. Every now and then, we work with organizations that still suffer from this problem.

Let's be clear. When quality matters—and shouldn't it always—everyone must play a role. Good testing involves a series of quality assurance filters. Each team in the organization typically participates in and owns one or more of these filters.

Some of these filters, especially high-level testing like system test and acceptance test, work best with a high level of independence. Some of these filters, especially low-level testing like unit testing, work best with less independence. Let's survey the degrees of independence so you understand your options.

Self-testing occurs when the developers test their own code. There is no independence here, of course. The author bias problem is significant, and the developers—even if given enough time to do unit testing—often miss the important bugs because they determine that the code works as they intended. Of course, that might not match the actual requirements. The advantages are that the developers can fix any defects they find quite quickly and, being quite technical, understand the software being tested.

Buddy testing occurs when developers test each other's code but not their own. Pair programming, which is a practice in some agile techniques, is a special form of this, where development of code, continuous code review, and development and execution of unit tests by a team of two programmers evolves the code. While the author bias problem is not so acute here, when two people work closely together, it's hard to say there is much independence between the developer and the tester. In addition, there tend to be few if any usable defect metrics captured in this situation without careful cultivation of the proper mindset because when peers test each other's code, they might not want to report defects. Finally, since the average programmer has little training or formal experience with testing, the mindset is usually focused on positive tests. Once again, the advantages include a quick repair of defects and good tester understanding of the software being tested.

Having the tester or testers inside of development occurs when a development team includes one or more testers and these testers are not part of an independent team. This is rather popular these days because many proponents of agile methodologies advocate this approach. There is nothing wrong with it as part of a larger quality assurance process, but by itself it can be dangerous. The main problem is editing and self-editing.

Self-editing means that the tester does not report—or reports only informally to the developer—problems they find, leaving no official trail in a bug tracking system. This is the equivalent of an organization tearing out its eyes and flying blind with respect to quality. Defect metrics are necessary to any balanced, meaningful picture of quality. In addition, an organization that doesn't study its mistakes is unable to learn from them.

Even if the tester does report bugs, editing can happen. The development manager or project manager does not allow the tester to release a clear, balanced, complete set of test results to the broader set of stakeholders. Furthermore, since the development manager or project manager is often focused on short-term goals like getting the product released on time and on budget, the entire mission for the testers is likely to be verifying adherence to requirements. Finally, it is often the case in these arrangements that the tester tasks are assigned to junior developers or factotums of some sort or another, along with a number of other responsibilities. Therefore, the testing is often done hurriedly and without any particular professionalism.

All those disadvantages enumerated, I do see value in having one or more testers—whose permanent positions are inside an independent test team and who are true professional testers—assigned to act as testing resources within development teams. In this role they can create good test cases, build automated test harnesses, create continuous integration build-and-smoke-test facilities, and the like. We have played this role for clients in the past, with great success. However, this too is often not sufficient by itself.

Testing by business, users, and technical support occurs, often in the context of acceptance testing and beta testing. This has the advantage of a truly independent outlook, motivated to report findings truthfully to the stakeholders. What these folks typically care about is the ability to get their job done, plain and simple, and if quality's not there, they'll suffer. This is a great approach for the final levels of testing.

Unfortunately, what we tend to see with organizations that rely on this approach for system testing is that the test team's skills are one-dimensional. They are focused entirely on domain knowledge. Technical skills, if present, are limited. Management disdains professional testing, with the usual refrain being, "Oh, any user can test." In addition, the amateur testers tend to bring a firefighting, patch-it-until-it-works attitude to the testing work.

Test specialists in an independent test group are present in many thoughtful organizations, with the independent test team responsible for system test, system integration test, and, in some cases, component integration test. In this case, we have all the advantages of true, professional testers testing against specific test targets. Unlike the approaches mentioned earlier, we often see test targets beyond functionality, including usability, security, and performance. For all the advantages of an independent test team, it should be kept in mind that the formality usually associated with such teams does tend to slow down the process. It's also possible that reporting structures or poor management can lead to perverse incentives and a lack of focus on quality.

Finally, testing by an external test organization occurs in a number of settings. For example, in certain military contracts, independent verification and validation by a team not in any way associated with the prime contractor is required. As another example, you might hire a test lab to do compatibility testing for an e-commerce website to save the expense of having all the configurations in-house. Here, the maximum level of independence is achieved. Of course, the separation of test and development duties might mean that the knowledge transfer necessary for thorough testing might not occur. To make up for these disconnections, the organizations must put in place very clear requirements and well-defined communication structures.

Furthermore, there's a potential "who guards the guards" problem. I worked with one client once where they were doing testing for a defense contractor. One of the test managers told me that he wasn't interested in skills growth for his test team. He just wanted templates for test plans and test cases so his testers could fill in the blanks. I asked him if it mattered whether they had their brain turned on or off while they were doing that. Surely even author-biased testing by the programmer is better than some unskilled tester doing a haphazard, path-of-least-resistance job of it. Any company engaging an external test organization should plan to audit the quality of that organization regularly, including the skills and professionalism of the team.

Now, notice that I listed advantages and disadvantages for each option. This means that you can use each option for one or more of the quality filters I mentioned earlier. The advantages and disadvantages tend to be mutually correcting, so a mix of degrees of independence, with different degrees for different tasks, is often appropriate.

Reducing matters to black or white, either/or dualities, while simplifying the concept, often loses some of the important nuances. This question of independence is one where too many people have fallen into the black or white, either/or duality trap. People ask, "Should we have developers testing, or users testing, or independent test teams, or buddy testing, or outside test labs?"

The answer is not one of these options to the exclusion of all others but rather each of these options, in some mix with some or all of the other options, to the degree appropriate for the particular project, product, and software development and testing process. Independence is a matter of degree, not an either/or state.

In addition, independence is an attribute of the relationship between those developing and those testing. For any two entities—whether at the level of individual people, teams, or organizations—we can ask, "What is the relationship between them and to what extent are they independent?” The more an entity is free to act as it sees fit, without having to accept direction from the other entity, the more independent that entity is.

Notice that the distinction here is not one of disregarding how one’s actions affect another but rather not having to get approval for those actions. This is important to remember because some independent test teams make the mistake of thinking that they can and should—as a sign of their independence—do whatever they think is right, and to the devil with the consequences to the project and the organization. Independent test teams that fall into this adversarial, “quality cop” mindset often end up being disbanded. Successful independent test teams act in consultation with other project stakeholders, preserving their independence, but with the goal of serving the stakeholders and the best interests of the project and organization in mind.

Increasing independence of testing is not without risks. More independence can result in more isolation. It can reduce the level of insight and understanding of what is going on in the project. It can also lead to a loss of ownership and responsibility for quality on the part of those developing the code. These are not necessary outcomes of independent testing, as some of those who argue a dualistic view of this question suggest, but simply project risks that the manager of an independent test team must mitigate.

Decreasing independence of testing is not without risk either. It can increase insight and understanding of the project—and this is the outcome touted by many of the dualists—but it can and often does introduce conflicting goals. Decreasing independence can lead to blind spots as to what the requirements really are. Decreasing independence also decreases the degree to which testing involves people who specialize in testing and thus have an imperfect, skewed skills mix.

In some cases, the choice of software development models and other project realities can influence the choices here. For example, if you are following an agile life cycle model, then pair programming and testing within the development team might be part of the mix. As another example, if you are seeking to have your product certified as Microsoft compatible, then using Microsoft’s compatibility test lab—obviously completely independent from your organization—will be required.

Again, remember that you can mix all of these independence options that have been discussed. In a few moments, you'll see an example of a very successful organization that does just that.

When you do split up the testing across various entities with various degrees of independence, the usual rules of pervasive testing apply. Make sure you define the responsibilities and expectations for each test level and entity doing testing. A concise, clear test policy document, developed with the participation of all the entities and approved by senior management, can accomplish this. By defining the responsibilities and expectations, you'll be setting up a mix of different filters, deployed at the ideal spot in the life cycle, which can maximize quality within the schedule and budget constraints of the project.

Outsourcing of testing is one form of external, independent testing. This can take a number of shapes. One is hiring an outside testing company to provide co-located testing services (which is sometimes called insourcing). Another is to have the testing done at an external facility located close to the development team. Yet another is to have the testing done at an external facility that is some distance away, perhaps not just miles or kilometers but also in time zones.

Many outsource services companies, RBCS included, provide a mix of all three options. We have testers who work on-site for our clients. We have the ability to provide testing services through local or in-country facilities. We also have the ability to have the testing done through partners around the world. While outsourcing does work, it does present some challenges:

- The outsource testing team might have cultural differences with your test team, your development team, or both
- The project team and local test team might have difficulty providing timely, adequate supervision and direction, particularly on chaotic, constantly changing projects.
- Due to a lack of foresight, significant communications problems can exist between the local project team and the outsource test entity. This can compound the supervision problem.
- Without careful contracting, you can have problems with protection of intellectual property. Even with good contracts, in some countries your legal recourse might be quite limited.

- Again, if insufficient care is taken when the testing is contracted, including especially the selection of the outsource test vendor, the skills of the testers can be questionable.
- Exacerbating this skills problem can be the problem of employee turnover. Again, proper contracting and vendor selection can help reduce this problem.
- Since companies pursuing outsourcing often forget to include their own costs of managing the relationship in the overall budget, outsourcing does not always involve accurate cost estimation.
- Finally, the quality of the work can suffer.

Some of these challenges can increase with distance. Again, I'm not mentioning these as outcomes that necessarily will occur and dog your project but rather as risks that you can and should mitigate. A lot of it comes down to careful outsource test vendor selection and careful contracting.

Figure 1 shows how one of our clients approaches integrating quality assurance tasks, including testing, with various degrees of independence, into the entire development life cycle.

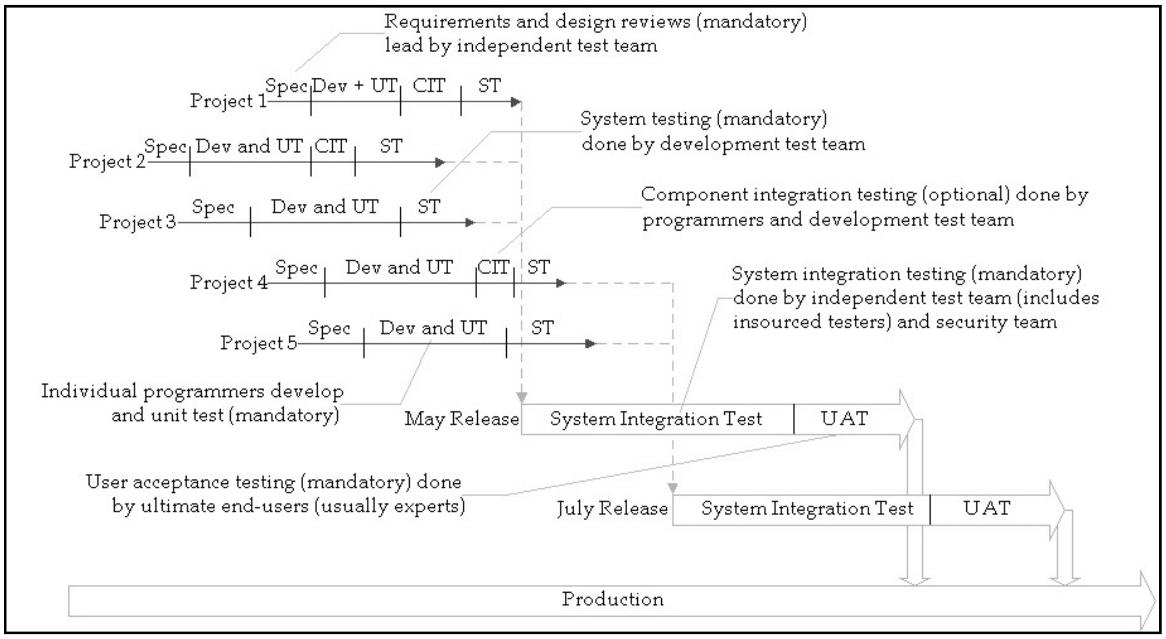
Requirements and design reviews (mandatory) lead by independent test team
Project 1
Spec
Dev + UT
CIT
ST
System testing (mandatory) done by development test team
Project 2
Spec
Dev and UT
CIT
ST
Project 3
Spec
Dev and UT
ST
Component integration testing (optional) done by programmers and development test team
Project 4
Spec
Dev and UT
CIT
ST
System integration testing (mandatory) done by independent test team (includes insourced testers) and security team
Project 5
Spec
Dev and UT
ST
Individual programmers develop and unit test (mandatory)
May Release
System Integration Test
UAT
User acceptance testing (mandatory) done by ultimate end-users (usually experts)
July Release
System Integration Test
UAT
Production

Figure 1: Mixed QA independence options

First, let me review the life cycle. Project teams take on projects for various business stakeholders who authorize those projects. To be deployed, projects will be included in a bundled release that will be tested with other projects and delivered into production. The bundled releases into production occur about every two months. This approach minimizes regression and interoperability risks.

For each project, as it moves through this life cycle, a variety of quality assurance activities take place. Each has differing levels of independence, differing responsible parties, and differing focus.

Each project goes through a requirements and design specification period at the beginning. The independent test team leads mandatory reviews of these specification documents. The participants vary, with business stakeholders and business analysts heavily involved in the requirements reviews and senior programmers, system architects, network administrators, and database administrators heavily involved in the design reviews.

Each project then goes into a development phase. Each programmer unit tests their own code. Programmers review each other's code. Unit tests are subject to approval.

Each project then moves into a pre-bundle testing period. In other words, the code developed will go through separate testing prior to integration with the other projects' code. This period can—but need not necessarily—contain a component integration test level, depending on the number of modules involved. The programmers themselves handle this component integration test, assisted by a test team internal to the development team working on the project. This internal development test team is a transient team, assembled by the project manager from members of the project development team. Thus, it is not independent but does have excellent insight into the project, especially technically.

The pre-bundle test period contains a mandatory system test level. The development test team handles this test level.

At this point, all those projects that have qualified to enter the bundle will do so. When I say, "qualified to enter the bundle," what I mean is that project metrics indicated sufficient quality to be in the bundle.

The first quality filter for the bundled software is system integration testing by the independent test team. This is a mandatory activity. One of the test types during system integration testing, security testing, is done by the security team, another separate group.

The final quality filter is user acceptance testing, done by actual users, ideally expert users who know the most about the business problem to be solved by the system.

So, we have eight quality filters:

- Requirements reviews
- Design reviews
- Code reviews
- Unit test
- Component integration test
- System test
- System integration test
- User acceptance test

These filters have different levels of independence.

So, does this approach work? Well, the defect detection percentage for system integration test and user acceptance test has been consistently over 99 percent for a few years now. This is in part because the independent test team and users find very few bugs in those final filters, only those bugs that couldn't have been found earlier.

So, yes, this approach works. In fact, if what you care about is the highest quality, every time, without compromise, this not only works but indeed is one of the best ways I've ever seen it done.