

## Blurring the Line Between Dev & QA

What's the difference between development and QA?

It's been decades since we began distinguishing between these two project roles, and in most organizations the fact that they are necessary and distinct from each other is taken as an article of faith. But new voices have arisen in recent years. Most of them do not suggest that we go back to the 1960s, but they do raise interesting questions about the Dev/QA dichotomy. How well has the traditional structure worked? What dysfunctions are crying out to be addressed? Can we make our projects more effective by re-thinking these two roles and how they relate to each other?

**Credit where credit is due:** When I spoke at the [Pacific Northwest Software Quality Conference](#) last week they were promoting [an upcoming event in Portland, Oregon](#), which bears the title I have co-opted for this article. The 17 November 2010 session is being presented by the Software Association of Oregon in Portland, as a joint effort of their Developers and Quality Assurance Forums. (I wish I could be in Portland to participate!)

### How Dev & QA Came to Be Distinct

Before the dawn of time (in the late 1800s), we saw the advent of the large corporation (with its need to process large amounts of financial data) as well as surges astronomical and other scientific research (with their need to process large amounts of scientific data). Thus, the computer was born. Large corporations and research organizations had "Computer rooms"—vast seas of desks at which people sat and scratched pencils down to nubs as they filled reams of papers with numbers. Their job title? "Computer."

A few decades later we developed machines that would eventually put those human Computers out of work, but there was no such thing as "software." The circuits in each digital computer were custom wired (by Electrical Engineers—EE's using wires) to perform the necessary functions. This practice was soon replaced by more generalized hardware and the concept of software (machine code) that the EE's used to encode the logic. As we progressed to Assembly language and then to higher-level languages like Fortran, the EE's began to branch and specialize. Some remained hardware engineers, and others transformed into coders of this new software stuff. Thus was born the role of "programmer."

**Fun side note:** The first computer "bug" was discovered in this time period under the direction of then-Lieutenant Grace Murray Hopper. A moth had been mashed by a relay in ENIAC, causing the relay to malfunction and the computer to stop working. When the moth's remains were found, the operators pulled it from the switch with a pair of tweezers, taped it into the logbook, noted dryly that the failure was the "First actual case of [a] bug being found," and reported that they had "debugged" the machine. ([A picture of the log page](#) is available on the Naval History and Heritage website as part of Admiral Hopper's biography.) We've been finding computer bugs ever since.

In the 1950s, it became clear that programming was a different profession from Electrical Engineering. Schools started providing education in this new arena, and ever so slowly, the ranks of programmers were filled by people who were not EE's. These people did all software work. They figured out what was needed,

designed the systems, coded up the logic and data storage mechanisms, tested and "de-bugged" them, put them into production, operated them and kept them running.

From there we have seen a continuing parade of specialization. Operations vs. programming. Designing vs. coding. Developing vs. testing. Technical work vs. project management. And most recently, business analysis vs. system development. Each of these specializations has been driven by the fact that each of these jobs requires specialized capabilities and knowledge requiring different training, education, and experience paths.

It is generally recognized today that conceiving, designing, and building software requires a different set of skills than conceiving, designing, and performing tests of that software.

## Separating QA from Dev

The distinction between developing and testing gave rise to the observation that "independence" of testing resulted in objectivity, which enabled the tester to see the system under test through a different lens than the developer. The tester did not share the developers' presuppositions, blind spots or biases. This made the tester more able to identify the defects that result from those failings. In addition, the tester felt no ownership of the code, and so had no reason to ignore or downplay issues with it.

The obvious advantages of independence gave rise to the idea that QA professionals on a project must be kept from working too closely with the developers. We moved the testers into separate departments and built walls between them and developers (literally and figuratively). We limited the opportunities for them to communicate across those walls, and built protocols for the communications that were allowed.

This degree of separation has been codified by regulators in some domains (for example, the US Food and Drug Administration—FDA), to the point that some regulators or auditors will cite an organization for not maintaining enough separation between developers and testers. And even in the majority of organizations, where this kind of structure is not required, separation is widely accepted as being an important contributor to product quality.

## The Dysfunctions of Separation

While there is clearly value in recognizing testing as a separate specialty, and while the difference between the testers' and developers' views of the system under test enhances the effectiveness of testing, the ways in which we have separated developers and testers have resulted in a lot of dysfunction on projects.

**Antagonism** between testers and developers is common. Operating as two separate groups creates an "us vs. them" relationship. While "us vs. them" is not inherently bad, it can easily degenerate into an adversarial relationship. While some organizations do a good job of keeping this relationship from becoming too adversarial, many suffer the consequences of antagonism, often because of some of the other dysfunctions!

**Misalignment** of objectives results in developers and testers working at cross-purposes. When developers are rewarded for producing lots of code quickly, they are often motivated to minimize their own testing and

other quality-oriented activities (e.g. design and reviews). This results in buggy (and sometimes dead-on-arrival) code being pushed into test, causing frustration for testers.

When testers are rewarded for writing lots of bug reports quickly, they are often motivated to be picky and report as a defect anything that strikes them as being questionable. This results in a flood of questionable defects reports washing back over the developers and frustrating them.

**Respect** is eroded when developers and testers don't appreciate what it takes to do each other's work. It is not uncommon for developers to view testing as beneath them. This idea comes from being unaware of the knowledge and skill required to do the job well. Testers sometimes view developers as not really caring about what they produce. This idea comes from having little (or no) experience grappling with the complexities of the systems that are being built.

**Throwing things over the wall** is the natural mode of operation when a wall separates people. This is a dysfunction to the extent that the thrower is unaware of the impact on the target, and the target doesn't appreciate the value of what is being thrown.

In the end, all of these dysfunctions boil down to the project not operating as a unified whole, but as two factions pushing against each other. Even in projects with well communicated and inspiring vision or mission statements, each faction will embrace and interpret the purpose of the project in a way that maintains, if not reinforces, the antagonism between them.

## Blurring the Line

Much recent experimentation has been done with how we run software projects, and the experiences that are being reported call into question the traditional separation of Dev and QA. The Agile methods are one example of this experimentation.

The Agile methods as originally documented do not even mention the role of "tester." The development team is responsible for delivering what the customer needs in the form of a technically excellent product. This sounds like going back to the 1960s, before "tester" was a recognized role.

But a funny thing happened in the real world of the organizations who began adopting Agility. There were testers! And management (if not the developers) recognized that testers were an important part of the organization, so they asked what testers do on Agile projects. The official answer (based on the Agile books' silence about testers) was, "I don't know." And the experimentation commenced.

The Agile value of "Individuals and Interactions over processes and tools" meant that even if the testers operated as a separate team, there was significantly more communication between developers and testers than on traditional projects. On some projects, a tester or two was actually made a member of the Agile team, ratcheting the communication level upward.

In addition, the developers were now delivering small slivers of product to test at the end of each short iteration—sometimes even more often than that. Even in cases where there was still a wall to throw the product over, the dynamics of the regular volleys reduced the negative impact. And in the cases without

walls (tester on the Agile team), there was a totally new cooperative dynamic that replaced throwing it over the wall.

The net effect reported out of many organizations is that many of the dysfunctions discussed above began to melt away. In working more closely together, testers and developers began to understand each other's work to an extent that improves respect. The complexities and difficulties of each job were better appreciated by the other professionals. Each began to appreciate the effects on the other party of what they do, leading to more cooperation and less antagonism.

But the biggest effect people are reporting is that the two teams operate less as two teams, and more as one team with one objective. We're all on this project together, and making it successful is **our** responsibility together. I cannot be successful unless you are, and vice versa!

But what of independence? How much do we lose when testers are more closely integrated with the development team? The emerging answer seems to be that not only do we not lose quality, but it is improved! The current experience seems to be that the need for separation was a myth. It's the existence of testing professionals on the project that brings the benefit. The separation added only dysfunction.

## So What Can We Learn?

For those who are adopting Agility, the lesson is clear: Testers should be integrated into the Agile team and simply work shoulder-to-shoulder with the developers.

But for those of us who are not embracing Agility (or not yet), there is much we should ponder. What can we do on our projects to break down the walls between developers and testers? How can we align their objectives to avoid cross-purposes and encourage one-project thinking? What opportunities exist for increasing communication between development and testing? How can we build the respect of each for the other?

Whether or not your projects are Agile, the concept of blurring the line between Dev and QA is worth exploring. Let's continue experimenting, figure out what works in each situation, and lay the foundation for moving into the next generation of thinking about roles on our projects.