The Agile Testing Pyramid

The Agile Testing Pyramid is a handy way of describing the difference between traditional software testing and testing for iterative development. The differences are important for Agile success and require re-thinking and re-tooling for many organizations.

Let me start this post by stating that there is nothing new in here other than perhaps my choice of words and expressions. The concept of the Agile Testing Pyramid has been described by many people and, I am pretty sure, originated by Mike Cohn. I am going to describe it here using the banter I use in my classes because it incorporates some concepts and tools I have learned over the years and it seems to get the message across pretty well.

Traditional software testing can be described using the pyramid on the left of Figure 1. The great majority of testing is done using test plans to manually exercise the system through the UI. There may be some automation in the middle tier to test services. There may be some unit testing by developers. I am continually amazed by the blank looks I get when I ask the technical people in my client organizations if they do unit testing. When I learned it back in the early 2000's I felt like I was late to the game. Apparently it is still a fairly rare practice.

![Testing Pyramids](image)

Figure 1: Testing Pyramids

We can think of traditional testing as designed to find bugs. That philosophy aligns with the traditional phased waterfall process of code and fix. Developers write code to implement features, ever confident that their code is perfect. Then QA engineers hammer on it to find the problems, logging defects for the developers to fix. The testing phase is as much about coding as testing when many defects are found. As the deadline approaches, out comes the digital duct tape to make quick fixes; technical debt piles up for future repayment.

Agile testing has a different shape in Figure 1. Now we have a pyramid with a solid base instead of the ice cream cone of traditional testing that can tip over at any moment. Agile testing relies more on automation. It requires a much greater contribution from developers. And it has a different basic philosophy to prevent bugs.

Automation requires a clear understanding of what a system must successfully accomplish. In the old days we called these functional specifications. When we know these, we can write tests that prove they have been met. To reduce process and documentation overhead, efficient Agile teams will go directly from User Story acceptance criteria to automated tests. This means that the bulk of testing is done using automated tests at the unit and story (functional) level instead of testing if the functional requirements are working. Automation gives us confidence that the product is right (does what is requested by the business) and is built right (delivers the requested functionality with high quality). We can validate the correctness on a continual basis by running our automated tests many times a day.
The Agile pyramid requires new skills and attitudes in many organizations. Developers must validate their own work with unit tests. The typical tools for this are xUnit implementations for the language in use. My favorite approach to this is Test Driven Development. QA professionals must learn automation tools that work at the story level. Some popular tools are Fitnesse, Concordion, Cucumber, Behat and Robot Framework.

Notice that the top of the pyramid, UI Testing is very small. The reason for this is simply that UI changes often. When it does, any automation at the UI layer will have to change with it. The test maintenance can quickly overcome the value of testing at the UI layer. So we encourage teams to use record and playback type testing as little as possible. Here is some more discussion of this issue: from Elisabeth Hedrickson and Martin Fowler. In successful Agile, automated testing does not mean using Selenium-type tools to run recorded scripts. That is only one small part of an efficient testing strategy.

To accomplish automated testing in the middle tier, a design philosophy change may be required, one that has been known for a long time called The Thin UI or The Humble Dialog Box pattern. The essence of this pattern is that all logic is moved to the logic layer (i.e. business rules, services) beneath the UI, leaving just the UI controls that send and receive messages. By doing this, test harnesses can access the logic to be tested without going through the UI widgets. They just communicate with the logic layer to test everything in and below it.

When developing iteratively, regression testing is essential. The system changes daily. We want to know that the things that used to work still work, so everything needs to be tested continuously. This is very hard to do when the testing is done manually. It is repetitious, boring, error-prone and likely to be incomplete. It also takes longer and longer to the point where there is less and less time for new feature work because testing takes everyone's time. Figure 2 shows how regression test load increases geometrically in iterative development.

![Figure 2: Test Workload - Manual vs. Automated](image)

Automation gives us an added benefit that improves overall team efficiency. In the old days, bug fixes often resulted in new bugs elsewhere in the system. Finding the cause a bug can take a long time. With automated tests and high code coverage we get, for free, a set of trouble indicators that tells us when a change has broken something else and an easier way to find where the breakage is: go to the test that is failing, see what code it tests and make the fix. The automated tests are also a form of safety net that helps us to refactor code with greater confidence. If all tests pass before and after the refactoring, we know that the new implementation did not break something that used to work. This confidence maps to the wide, solid base of the Agile Testing Pyramid. When most of the testing is automated, the amount of test work can track linearly with story development, also shown in Figure 2. Some old tests may have to change as new stories cause changes in old stories, but these will be fairly infrequent. By automating what we know must be true, we have constant validation from regression testing and few surprises. They may still be some unanticipated behaviors from the overall complexity of the systems we tend to build so the time saved by automation can be applied to exploratory testing - targeted manual or scripted testing to find the surprises.

Agile testing done right will give your team a firm foundation for iterative development. We can be clear on what must be working. We can be confident that they are working. And we can more quickly attend to things that stop working as we add functionality.