Agile methodologies are approaches to managing software development based on short-term, iterative, and incremental deliveries, enabling continuous feedback and flexible response to change.

Stemming from a rapidly evolving business environment that demands faster product improvements and modifications, the agile methodology promotes the organizational qualities of speed, responsiveness, and adaptability throughout the entire application management process, from defining product requirements to coding, testing, and, finally, release management.

This article explains why agile development cannot be implemented effectively without unit testing—and especially automated unit testing.

The Importance of Code Quality

Developers have known for decades that the further into a project timeline a bug gets discovered from its insertion point, the more costly it is to fix. When a developer finds a bug, it can sometimes take minutes to fix. If it slips through testing and finds its way to the customer, figure 1 shows that mitigation can be exponentially more expensive to fix. [1]

Correcting quality issues can take months of rework, cost millions of dollars, or, if done too late, may even cost lives. Take, for example, the first launch of the Ariane 5 rocket in 1996. Its flight abruptly terminated just thirty-seven seconds after liftoff, taking with it hundreds of millions of dollars in invested effort. Also think of Toyota recalling four hundred thousand vehicles because of a bug in the brake control system, costing an estimated three billion dollars.

While we can't eliminate all bugs, we can fight them by baking quality into the code. There are many ways to define code quality depending on the perspective of the customer or the developer.

The customer expects working software. Customers do not care how the code is written—they just need the software to work. When developers talk about code quality, they talk about code that is easy to maintain, easy to read, and risk-adverse to change. Each perspective takes the cost of bugs into consideration. The customer knows that for each bug, he'll lose precious business hours or days. The developer knows that each returning bug means considerable time spent fixing it instead of working on new features.

Agile methodologies take working software and combine it with early feedback. For example, early releases can get user feedback about how well the software operates. To give the developers confidence that their code works, unit testing gives the fastest available quality feedback.

The earlier defects are found, the cheaper they are to fix. As agile methodologies encourage high code quality, the team should run lots of unit tests. Similarly, automated tests give the developer early feedback on the quality of the software in a repeatable fashion prior to release.

What Is Unit Testing?

Unit testing is a methodology where individual units of software, associated data, and usage procedures are tested to determine whether they operate correctly. The unit is usually a small piece of code—for example, a single function. The unit test is a short function that tests the behavior of the unit that produces a pass/fail result. This is achieved by performing the tested function on a known value with a single correct result. Unit tests often use mock objects to simulate the behavior of dependencies in a predictable way.

The main purpose of unit testing is to allow developers to identify as many problems as possible at the development stage and to do it in an automated, repeatable fashion that can be applied for every code change.

This makes developers directly responsible for producing working code, even before it reaches the quality assurance team.

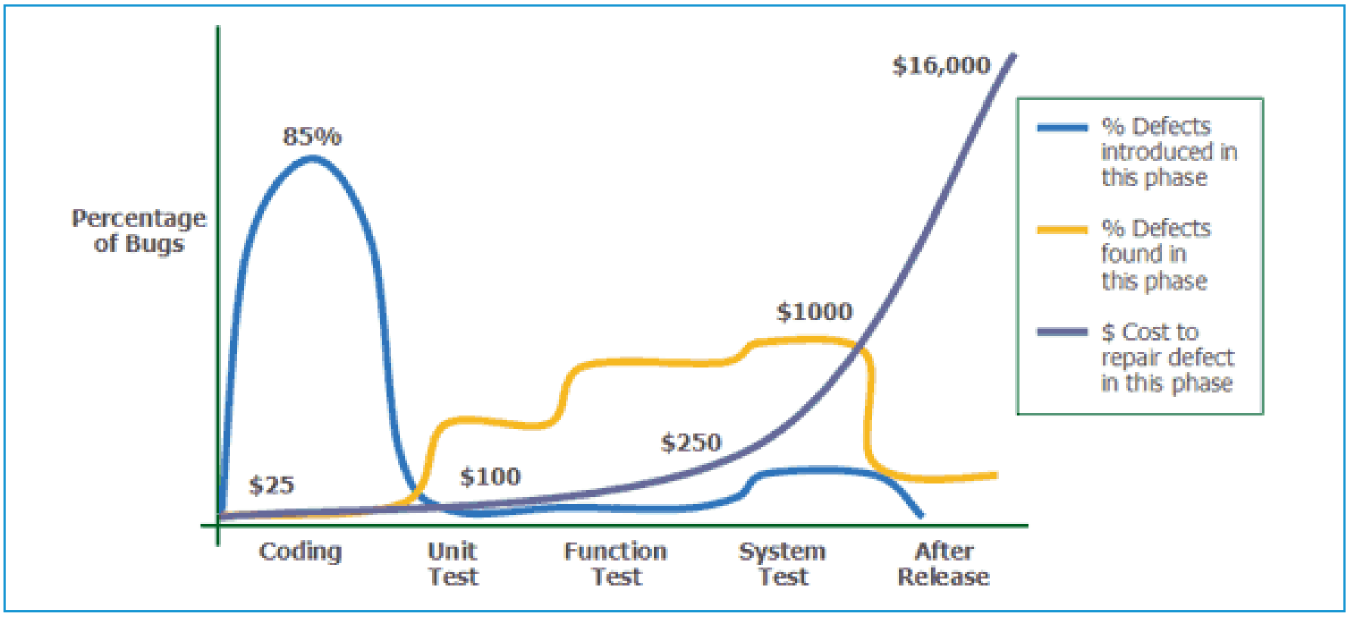

Figure 1: Example of the cost of defect correction during a software project's development lifecycle

What Does Unit Testing Have to Do with Agile Development?

I think the two are closely linked. In fact, I believe you can't be truly agile without implementing automated unit testing as an integral part of the development process. Automated unit testing has several benefits that align closely with agile development principles.

The central benefit of unit testing is that it produces working code faster and with fewer bugs. The ability to automate tests and catch bugs at the development stage reduces a huge amount of overhead that is otherwise spent on releases that are immediately rejected by QA due to basic functionalities being broken. Unit testing increases the chances of a new feature working correctly upon first delivery, as it becomes the developer's responsibility to verify that he is delivering working code.

Another reason that unit tests cut down on development time is that their fine resolution allows them to pinpoint precisely the location of a problem. A failed unit test can direct the developer to the exact location of the problem in the code, allowing him to quickly resolve it. This minimizes or even eliminates the time that would otherwise be spent locating the problem.

Unit testing may not be able to catch all bugs, but it is highly effective in catching regression bugs that are defects that break existing functionality. These bugs hamper progress and waste valuable development and QA resources as code is sent back and forth between the two departments, delaying new versions of existing products and new product releases. Without automated testing, it is virtually impossible to detect bugs during the development phase. This causes sprints to become bogged down as developers need to spend more and more time fixing regression bugs in order to keep producing working software. It becomes impossible to maintain a steady and predictable software delivery schedule while also maintaining quality. When a release date draws near and the product is not working, panic sets in, software is released without enough time to test it, and more bugs are introduced, creating a vicious cycle.

Code that is not properly maintained very quickly becomes legacy code that developers either refuse to change or insist on rewriting themselves. To keep code alive, you need to be able to change it and be confident that your changes won't break anything. Unit testing promotes this confidence. Without it, you end up either refusing to change older code or investing large amounts of time rewriting it every so often. In order to respond quickly to change, you need to be able to modify all parts of your code quickly and confidently. Some tools even allow you to develop unit tests for older code without having to change the code itself.

Agility through Automation

The platform for unit testing is implicit, and we usually omit the word automated before it. In reality, unit testing is a collection of processes, skills, and tools that support agility. For example, writing the tests is an actual skill. I look at tests I wrote five years ago and think, "How would anyone let me write this?" (I'm sure I'll feel the same in five more years about what I'm writing now.)

In addition, using isolation and mock objects correctly is a capability that improves over time. Refactoring of the tested code or changing code design can fill up a three-day workshop, and much like design, it can be improved and lead to maintainable test design.

When we improve our skills, we can move more quickly and change directions as we go with agility. But without automation, we won't be able to use our skills effectively in a repeatable fashion.

Automation is the foundation that gives the power to get quick feedback from running tests. It gives us the ability to cover more code and know we didn't break anything. And it gives us the independence to change our design when we need to without risk and to mold the software the way we want it.

In the end, the Agile Manifesto favors working software. Automated unit tests bring us close to that point quicker than other processes.

Conclusion

The benefits of unit testing are closely aligned with the principles of agile software development. Unit testing allows you to make code changes while remaining confident that they will not break existing functionality and that the major part of new functionality will work on first delivery. This enables frequent, timely delivery of working software, which in turn enables swift response to changes in requirements. Automated unit testing also promotes a transparent view into the code's health by producing reports that allow anyone to see which problems occur and their precise locations in the code. Further, automated unit testing reduces the number of regression bugs, preventing development sprints from becoming bogged down and enabling developers to maintain a constant, sustainable work pace.

Together with the agile methodology, an integrated, automated unit testing tool that works well within your programming environment is a crucial necessity for managing modern software development. **{end}**

gilz@typemock.com

Sticky Notes

Click here to read more at StickyMinds.com.

- References