ObjectWind Software *-Harness the Winds of Change*

Mark Windholtz, www.objectwind.com

Testing Existing Code

The next statement turns conventional wisdom completely on its head. *Complete automated Testing allows new features to be delivered more quickly into production*. Does this sound backwards to you? It is a common misconception to believe that there is a trade-off between: **Quality** and **Time to Market.** So, in too many cases, software projects are delivered late and without adequate testing. Agile Software Processes have shown that increasing quality allows programmers to add features more rapidly. Think about your garden shed, kitchen, or woodshop. It's always easier to make progress when things are put away, clean, and reliable. Certainly, if you only put the dishes away once a year cooking simple dishes would be overwhelming.

A complete automated test suite is the way to prepare your application code for the next round of new features. One of the highest priorities in extending a system is to not break what is currently working. Testing existing production code is the first step in extending it in a controlled manner. Feature extensions can add subtle errors to the existing behavior of a system.

Testing allows your team to add new features much, much faster

Testing current functionality used to be called *Regression Testing*. Agile teams just call it testing. Agile teams run tests sooner and more often. Regression Testing of old was conducted by the Quality Department right before shipping the software. This detects code defects at the last possible minute from escaping into production. Discovering defects at the first possible minute has an immediate payback by allowing programmers to focus more on new features and less on the accumulating defects during code construction. This highlights the compounding benefits of early testing. (1) It finds the defects. (2) It finds them early so that code construction can go faster. (3) In finding defects early they are easier to isolate to a smaller set of recent changes and easier to fix. So, full automatic testing allows your team to add new features much, much faster. Because a suite of automated tests acts as a safety net for preserving the current features of the application.

What kinds of testing should programming teams consider? Agile teams use two levels of testing: **(1) Functional Testing**focuses on end-to-end User-level behavior of the while system. **(2) Unit Testing** focuses on behavior on the class and method level in Object Oriented systems.

Most production software applications started out as carefully designed modules. Over time they were extended and defects were fixed. Often this results in systems that have a large code base of interdependent code modules. For testing purposes small bits of code cannot be tested without a lot of set-up. This leads to very large and complex tests. Large tests are hard to run and difficult to maintain. Agile teams use a technique of test-first programming when building new feature code. Test first is very productive and often results in superior, flexible design. But how can we add Agile style testing to a existing, large, interdependent code base?

Guard & Grow approach is useful when the new features can be built with clear interfaces

I have worked with clients to implement two different approaches that I call: (1) Guard & Grow and (2) Secure & Separate. A **Guard & Grow (G&G)** approach is useful when the new features can be built with clear interfaces into the original system. The first step is to Guard the new feature code from the interdependencies of the old code. The new code is built in a test-first and disciplined fashion. Usually this requires that isolation code be written to guard the new feature code from the existing system. Design Patterns can be used to strictly manage the code dependencies between the new features and the old code. And the interface code itself is usually easy to write and understand. Once the feature has been built test-first with a high quality design, the new code area can be grown as more new features are added to the system. As the system ages the new features and test-first code will continue to grow, and the percentage of the system covered by unit tests will increase. Which in turn will decrease your sustainment costs and increase your ability to add new features.

Secure & Separate first secures the current behavior of the system behavior with function tests, and then Sepa-

The second approach is to move the whole code base forward at the same time. I call this: **Secure & Separate (S&S)**. Secure & Separate first secures the current behavior of the system behavior with function tests, and then Separates the pieces into Modules. The Modules can then get individual treatment by adding unit tests using the techniques of G&G. Secure & Separate is a more difficult to get started than G&G, because S&S is a global approach. An initial step that quickly pays for itself is to implement just the Secure part of S&S. Securing the high-level functional tests reduces the unpleasant surprises associated with adding new features. When the application is Secure only those things change that you want to change and fewer unintended defects creep into the code.

The first step is to begin writing large size functional tests for the system. It is still often easier to avoid testing through the Graphical User Interfaces and start with the highest level of functionality that is accessible through code APIs. One of the best tools for this is the xUnit family of test frameworks. (free at: www.junit.org) Chart you can dependencies in java code with: www.objectwind.com/dependNet.

Of course, G&G and S&S are not mutually exclusive approaches. Even if we focus on extending a smaller part of the system with G&G we would usually still benefit from the added stability in the larger system provided by S&S.

In conclusion, adding automated testing is the best way to allow for future expansion of application features. The two approaches to testing existing code are (1) to secure the current features and separate it into modules. And (2) to guard a new area of the code with well managed interfaces while you grow the cleaner code base using test-first development techniques. While both of these approaches sound daunting at first they are a clear way out of the uncertainties of adding features to code with unmanaged dependencies.

Please send comments or questions to: mailto:info@objectwind.com

Mark Windholtz
ObjectWind Software Ltd.
www.ObjectWind.com