

DOMAIN SPECIFIC LANGUAGES FOR FUNCTIONAL TESTING

Chad Wathington
Vice President – Products, ThoughtWorks Studios

February 2008

▶ THE PROMISE OF QUALITY SOFTWARE

Most software professionals believe that testing software is essential to quality. Where people inside the industry differ is how to accomplish that testing - strategies vary by level of the application tested, tools, methodology, amount of automation, and who completes the testing itself. The aspirational desire is to ship high-quality bug-free software no matter how one gets there. However, the devil is in the details.

There are four essential questions that are central to creating a testing strategy that works:

1. How can tests be executed early and often enough to mitigate risk?
2. How can tests be maintained as long-lived, evolving, and reusable assets?
3. How can testing involve all the stakeholders in the software development process such that requirements are fully understood and tested?
4. How can the highest value most essential testing be identified?

Automation is one solution to the risk question. With an automated test suite, an application can be tested as frequently and rapidly as your hardware constraints allow, hypothetically. And, if the automation is started as the application evolves, then the test suite grows with the **system under test** (SUT). However, to date, automation has created additional problems, particularly in addressing the asset and stakeholder questions.

Automated tests are often very difficult to maintain as assets because they can be brittle. As an application changes over time between releases or during the development cycle, the tests must change accordingly. The tool support for changing automated tests frequently and uniformly over time is lacking. Consequently, when tests break because of application changes, they are difficult to repair. Moreover, if automated tests at the GUI level are created by the typical record and playback mechanisms, they need to be re-recorded, especially if the GUI changes substantially.

In terms of stakeholder inclusion, automated testing via traditional means typically falls short as well. Automated tests have a representation problem – tests are generally represented in ways that make sense to computers, not to subject matter experts or business analysts. In some cases, they have no useful representation or specification for non-technical users at all. Some tools try to address this problem by aiming their wares at business users, claiming to have wizard driven “no code required” solutions. These tools suffer from the same issues that many rules engines do – they are not intuitive for technical users, who also need the tests to troubleshoot, and business users often need substantial help to use them effectively.

While holding a lot of promise, automated testing isn't currently delivering the value possible because it makes the asset and stakeholder questions more difficult to answer. However, over the life of a software application an automated test suite at the functional level can:

▶ DOMAIN SPECIFIC LANGUAGES (DSLs)

“I believe that the hardest part of software projects, the most common source of project failure, is communication with the customers and users of that software. By providing a clear yet precise language to deal with domains, a DSL can help improve this communication.” – **Martin Fowler**, *Domain Specific Languages* (working title).

Involving all the stakeholders in the testing process requires that subject matter experts, customers, testers, and developers communicate consistently and effectively. In the past, many software project teams relied on extensive requirements documentation to communicate effectively across these groups. However, requirements in this context require a fair amount of translation. **Subject Matter Experts** (SMEs) and analysts must translate customer desires and needs into requirements. Developers must translate these requirements into code, and testers must independently translate the same requirements into tests. Finally, each one of these translations must match the customers' original expectations, which may change over time. It's similar to the telephone game that children play in primary school, only that the starting phrase is a moving target.

We could effect a substantial improvement to the translation problem if the customers could express and test their own requirements, or acceptance criteria, themselves. At a minimum, customers should be able to simply review any translations that occur in the process and verify that translations match their needs.

One means to ensure that code across a project team is readable by everyone is a DSL. Domain specific languages are mini-computer languages that are designed around a very specific purpose or domain. They are not general programming languages like Java, C# or Ruby but are tailored toward a narrow task. Depending on the implementation, they can also closely resemble written human language. Applied to automated functional testing, creating a DSL to express testing intent solves several problems.

A DSL for an SUT allows a team to share a vocabulary that describes their domain. If a standard vocabulary is already in use, then using a DSL to express acceptance criteria minimizes translation errors. Furthermore, DSLs allow non-technical stakeholders to interact with the testing effort on more level terms. A properly designed DSL will at a minimum allow these stakeholders to read tests. Under the right circumstances, and with the appropriate skill level, it's possible for domain experts and non-technical users to write tests as well.

▶ TOOL SUPPORT

Conceptually, we can address much of the problems with automated testing by creating DSLs that fit application domains and refactoring our tests as the application changes. In practice, this is easier said than done. Many refactorings are impractical to perform manually, and creating a DSL requires a fair amount of developer time.

However, it is possible to create an integrated testing environment, or an IDE for testers, that makes testing oriented refactoring possible. It is also possible to make the DSL creation process simpler along a few parameters. In particular, if testers had enough tool support to

create their own DSLs without substantive help from developers, this could make testing a team level more accessible.

▶ FROM TOOL SUPPORT TO COLLABORATION

Beyond writing a DSL individually or refactoring ones' own tests, the best way to deal with the increasingly complex task of testing modern software is to make it a group effort. Fundamentally, to deal with ambiguity, uncertainty, and translation errors, testing should encompass perspectives from the entire team.

Taking the team involvement a step further, collaboration helps answer the question of what testing is essential. Since determining the essential points for your testing strategy isn't necessarily a straightforward process, frequent information sharing within a team makes these issues clearer. For instance, the choke points of an SUT today may not be the same tomorrow. The parts of the application that seem most complicated to the end users, SMEs, analysts, or customers may be trivial for the developers. Similarly, pieces of the application with lower risk to the business may have higher technical risk. When testing isn't just a function of the QA department, the differing perspectives on what is essential add to a comprehensive picture of the application.

The best way to facilitate this collaboration is to build it right into the testing tool itself. Thus, when the essential parts of the SUT change, everyone on the team can readily see the changes and adjust the testing strategy accordingly.

▶ ANNOUNCING **twist**

Later in 2008 ThoughtWorks will release a testing tool that addresses many of the issues discussed above. It will provide the facility to create DSLs, refactor, and collaborate.

To apply for the Twist Early Access Program, please visit www.thoughtworks.com/twist

▶ ABOUT THOUGHTWORKS STUDIOS

ThoughtWorks Studios builds the next generation of tools for custom software development. Our products are designed to enhance developer productivity and software time-to-market.

ThoughtWorks Studios is the specialist product division of ThoughtWorks Inc, which has been delivering world-class products and services for the past fourteen years to some of the world's largest organizations.