A functional approach to BDD assertions in Javascript

Behaviour Driven Development

Behaviour Driven Development, or BDD, is today one of the standard methods for developing software, particularly under agile methodologies. Using BDD, every required behaviour of the system should have a corresponding test to demonstrate that this requirement has been met. Alongside their use in as demonstrating that the intended behaviours are met, the tests often serve as the sole documentation for the correct behaviour of a system's units.

Application of my dissertation

Client-side web scripting via Javascript is a field which at inception contributed no more than small, frequently gimmicky, dynamic features added to otherwise static webpages. Today the scope and power of client side scripting has increased to the extent that the entire interface for large, complex applications is often programmed in this way. These applications are not limited to running under traditional web browsers but also include mobile apps and desktop software. Given the very fast expansion the web has seen from a technology based around marked-up documents to a full-blown application suite, it is not surprising that the tools have lagged behind in comparison to more mature languages. While TDD and BDD have been common for a long time, Javascript has only recently started to benefit from this more rigorous approach. I believe this lateness is not due to problems inherent with the language; Javascript is very amenable to automated testing. I propose creating BDD assertions for Javascript that are the equal of the best available in any language. Particularly of interest at the moment is QuickCheck, which I plan to draw some inspiration from.

The existing Javascript BDD technology stack

Test runners exist which allow a developer to push tests to several platforms simultaneously, returning results within milliseconds. Of the numerous runners, a Google project, JS Test Driver, JSTD, is emerging as the de facto standard. JSTD comes with a deliberately basic set of assertions. However, tests written using these assertions are more of a unit-testing form than a behavioural testing style and are not easily maintainable by non-programmers. The JSTD web site states that their assertion functions are rudimentary because they would rather focus on creating the client-server aspects of their application. The JSTD project hopes that better assertions will be contributed by the open source community. So far, developers providing BDD frameworks for Javascript have mostly attempted to deliver full-stack applications rather than concentrating on just one part. This approach frequently overreaches and fails to take advantage of a considerable body of prior work, manifesting itself as a rather defuse effort which tackles many areas but excels at none. I propose to focus relatively narrowly, creating only a framework for assertions and for maximum compatibility to do so in a way that is agnostic as to the context in which the assertions are used.

Programming language influences and considerations

Delivering a BDD framework involves writing software which is used in modelling the desired behaviour of software. Because it is code applied to code, the model of a testing framework exists at a higher level of abstraction than developers typically engage. Hence, the design considerations will be quite different from programming a model of the typical application domain. In order to allow a natural expression of bdd-style tests in Javascript, I propose that I will first create a functional programming library on which it will depend. This will require a large degree of metaprogramming. This library will reproduce as closely as possible the necessary features from languages such as Haskell. Although the code under test will almost certainly come with side effects, the assertions should allow tests that are themselves free of side-effects and wholly stateless.

Source code ergonomics and BDD as documentation

With Behavioural tests functioning as documentation, it is desirable that the tests be understandable not just by software developers but also individuals with little understanding of programming. This understandability by non-experts will be gained by providing a framework which allows tests that, as well as functioning as running software, may be expressed in a form very close to grammatically correct English. The ability to program tests in plain English will be a concern throughout my dissertation. For every story delivered, I will be considering developer ergonomics and iteratively developing a thesis of the factors which contribute to a highly usable testing framework.

Self-verification

My BDD framework will itself be developed under the advantages that BDD-style testing brings. With this in mind, the framework will be used to test itself. Because it becomes quite 'meta', testing a BDD framework is a relatively difficult problem domain to apply BDD to. Hence, if my framework is flexible enough to be used in this way I will judge it suitable for application to most problem domains. As well as ensuring that the framework is capable, self-verification allows me to write the BDD assertion framework while myself using BDD, enabling safe refactoring of existing code.

Delivery methodology

My thesis will be developed 'in the open' by committing all code and writing to a public repository hosted on Github. This should allow members of the developer community to contribute comments and suggestions as the work progresses.

I plan to use Kanban to deliver my dissertation, including the written parts. Because Kanban focusses on always having a potentially releasable product, it mitigates problems which could otherwise lead to non-delivery and allows the direction to be changed as necessary. For each unit of work (under Kanban, a card), an entire vertical slice of planning, design, implementation and reflection must be complete before going onto the next card. Alongside each software feature, every written chapter will be expanded and refactored in much the same way as the code. Just as for well designed software, the order of implementation should not be apparent to a user, my plan is that the written work should not feel disjointed for having been written non-sequentially. I plan to manage the Kanban process using paper only, with cards on a physical board.

Timescales

I am registered on the Software Engineering Program until December. While requesting an extension to my time on the course might be sensible to cover unforeseen circumstances, I plan to deliver the dissertation towards the end of 2012.

Summary of deliverables

I propose to create a Javascript BDD-style assertion framework which borrows heavily from functional programming, particularly QuickCheck. Research and analysis of source code ergonomics should result in a framework enabling tests that are easily understandable, maintainable and readable as plain English. The tests will be runnable in all reasonable contexts using existing test runners. To demonstrate correctness, the testing framework will be capable of testing itself.