Reflections on xDD Development Methodologies

Charles E. Matthews
Fifth Generation Systems, Ltd.
Markham, Ontario, Canada
charles.matthews@acm.org

Abstract

The proposal for the SPLASH'12 Workshop **xDD - What Drives Design?** raises interesting questions about software engineering development practices. In order to address these questions, I describe a recent project that I completed for a client in April 2012. After describing the project, I propose answers for some of the questions that the organizers posed in the workshop summary.

TDD Example Project

On March 10, 2012, I began a project for a client that required the modification of the code base for an existing product. This project was an embedded system that used the LPC2194 processor, which is an ARM7 processor from NXP. The product had a number of pressure and temperature sensors that communicated with the external world via a serial port and used the Modbus communication protocol. My task was to modify the existing code base to support the DNP3 serial communication protocol.

The entire project was estimated at five man-weeks of effort. (Unfortunately for me, the estimate was made prior to my involvement on the project by a previous employee who then bid my client, "Adios," and moved to greener pastures.) I estimated the effort at seven man-weeks, but, by that time, the project had already been accepted by my client's customer as a fixed-price bid at five man-weeks. The actual effort expended turned out to be 8 man-weeks.

Of the 8 man-weeks of effort, I spent 16% writing the design specification, 24% designing the new code, 37% implementing the code, and 8% performing the final test and verification tasks. I spent the remaining time on certification tasks for the DNP3 protocol.

The amount of new source code written for the implementation was 5855 lines of code. The unit test code base was 3848 lines of code. Of the total amount of code written for the project, 39.7% was unit test code.

The primary factor for the "success" of this project was the use of an open source unit test framework for the project. I selected the Unity tool (http://throwtheswitch.org/white-papers/unity-intro.html) as the test harness. For the project implementation, I followed a SCRUM story-based approach and used a modified Test Driven Design (TDD) process. During the design phase, I identified a list of stories for the implementation. As I completed each story, I added unit tests to test the story. My modification to the TDD process was to write the unit tests after I implemented the story rather than before.

My implementation ended up with 19 stories and 80 unit tests. After the final code release to the customer, the customer identified only 2 bugs within the first two months after the release. Although the project took three weeks longer than the initial project quote, it was only one week off of my estimate. This was a successful project. I attribute the success to the use of SCRUM stories and the use of unit testing with a test harness that supported automated regression testing.

Workshop Questions

For the purpose of a thoughtful discussion, consider the answers to the following questions.

How do new languages and development environments affect the design process? For example, do scripting languages like Javascript and Ruby promote design or "non-design"? Is Eclipse-based Java development just *another video game?* - One must be careful about designing a language or tool that introduces accidental complexity into the design process. For example, consider the Visual Basic language. Although the designers of Visual Basic may have been motivated by the vision of helping non-programmers create their own software, the limitations of the language caused far too many problems the moment that programmers began using Visual Basic for complex problems. You can't build a safe car out of Legos, and you can't build serious software with Visual Basic. If you use a language that has built-in flexibility such that it can be "A" in one context or "B" in a different context, then at some point you will have incorrect behavior because you "thought" that you were doing "A" but the software was doing "B". At that point your program will fail, and you may be clueless as to why it has failed.

Should we do our design thinking in pictures or words? This question is similar to asking whether programmers who speak English write better software than programmers who speak French. (Substitute any spoken language you wish.) In most cases, this is a matter of personal preference. Any individual who strongly prefers to design in pictures will be counter-balanced with another individual who strongly prefers to design in words.

How should we teach design to junior staff members? I believe that the way that Agile programming is practiced in most organizations actually inhibits junior staff from learning lessons from the more experienced staff. Most books on Agile programming take a very heavy-handed approach to the "we are all equal" stance. The reality is that some people are simply better at this job than others. Until your organization provides a way to value the lessons that the more experienced developers have, every crop of new hires will continue to make the same mistakes over and over and over.

One criticism of many of the xDD design approaches is that they "drive out" much of the creativity, innovation, and fun. What are some design principles and practices that we should consider using to reinforce and reward innovation? In my experience, a design approach drives out creativity and innovation only when it is presented in a cookbook form. If a particular approach is presented as a list of rules to follow with no room for deviation, then it becomes extremely brittle. When a design approach is presented as a conversation that allows one to consider which principles to use under different circumstances, the prospect for adoption improves.

There is an ongoing battle over what is the right volume of design documentation. Can xDD principles help designers find the balance between anarchy and process obsession? Some Agile proponents display strong opposition to almost any documentation. They say that no one ever reads the documentation after it is written. On the contrary, I find that a design document is the only way to maintain the final authority on what design decisions were made. It is the final source of arbitration after the designers have forgotten what they agreed to implement.

Conclusions

A very unfortunate characteristic that I observe with many supporters of the various methodologies is that they leave little room for disagreement with **any** of their positions. However, the trend that troubles me more is the tendency to deviate from logical, structured arguments into emotional arguments for persuasion. I see this characteristic with many of the well known authors, such as Kent Beck, who are regarded as the leaders of the Agile movement. Speaking as someone who has been around this industry for a number of decades, this style of intolerance is a relatively recent characteristic and may be a reflection of an unfortunate growth of intolerance in the general public. As long as people mask propaganda as scientific discussion, there is little hope for agreement on anything.