**CS 528 : Object Oriented Programming and Design** **Summer 2014**

# Problem Set 7

Dan McQuillan *Handed In: June 30, 2014*

**Part 0** Click the links above and read the original blog post, and the response from Kent Beck. Then, answer the following briefly (2-3 sentences).

(a) What are the major complaints that David brings against TDD?

That TDD was hurting the architecture/design of his applications. TDD will lead to overly complex applications and unnecessary abstraction. He also states that we should be focusing more on the design of an application rather than the structure of our unit tests.

(b) What does he propose as being the solution?

His proposed solution is that we should put less emphasis on unit tests and more emphasis on system tests. He proposes that we should move away from test first and move more in the direction of testing the responses after the fact.

(c) Pick two of the things Kent Beck would miss without using TDD. Can you accomplish those things without the use of TDD?

i. Logic errors
This could be accomplished through rigorous testing of the service, but it would be easier to plan out these utilities via unit testing and the test first pattern.

ii. Anxiety
This can be accomplished also through rigorous e2e tests, which can be very time consuming.

**Part 1** (a) What is a Mock Object? Give an example of the use for a mock.

A mock object can be used to simulate an example instantiation of an object/class. An example usage could be when interacting with a ReST service. For example if you have a user authentication service and you would like to test responses from the service to make sure that the correct flow is taken for a response from the service.

(b) At one point, Kent says the following:

Even if I don't know how to implement something I can almost always figure out how to write a test for it. And if I can't figure out how to write a test, I have no business writing it in the first place.

Do you agree with this sentiment? Why or why not?

I don't fully agree with this sentiment. However, I do agree with Kent's opinion of you should have no business writing it yet. I say yet because I believe it's more of an issue of you not understanding the problem at hand. Not knowing how to write the unit tests is just an example of a causality of not understanding the problem.

(c) What is the "red/green/refactor" loop?

Red is after writing the tests they will failure. Then green is working on it until it works. Once it's working it may not be in a "perfect" state so you should then refactor to clean up the code to make it more readable/reusable. It is noted as a loop since this process should be repeated until the desired results have been reached.

(d) Which way of arriving at a test suite is better, in your opinion: "going through the tests" (test-first) or "going to the tests" (test-after)? Does it matter at all? Does this depend on the situation? (If so, in what way/ways?)

I think it depends on the situation and the task at hand. If for example you're writing a service that acts as a utility and you fully know what needs to be implemented and all the use cases. Then, test first may be the better option. However, if you're working on a task that is more dependent on a third party, such as an api, where all the use cases are not clear. Then, a mix of the two might be better. Test first to create the necessary set up and helpers and test after to do integration and performance testing.

(e) In what cases is TDD not (easily) applicable? Give at least one example. What can you do in these cases?

An example where TDD is not easily applicable would be when using live data. In these cases integration, service, and usability testing can be helpful. You can also use mock objects in those cases although all the cases may not be caught and tested.

(f) In what ways can an overzealous demand for isolation in unit testing create problems? Is this inherent in unit-testing or TDD?

By having a demand for isolation you are making sure that each component works when isolated from the rest of the components. However you are not testing these components when they are communicating with each other which can cause some edge cases to arise. This is inherent mainly in unit testing as TDD will allows you to thoroughly analyze the problem before beginning on development. However, it can also appear in TDD in edge cases that a developer may not expect.

(g) Why should you have an automated unit test suite? Give at least two reasons.

As the code base grows on larger projects it helps to have to test suite to make sure changes to the code do not break the core functional aspects of the application. This can help by integrating the unit tests in the build of the application. Another reason is because it helps to document the functionality of the component you are unit testing.

**Part 2** (a) What is an MVC application? Where did this design pattern originate?

(b) Can an application have a worse design because of an attempt to make it more testable? Or is it the case that something that is more testable is always a better design?

(c) How does TDD influence the design of code? Give a positive and a negative example.

(d) David correlates the size of a codebase with how easy it is to change it. Does this correlation ring true, in your experience?

(e) Compare the situation David brings up of ?test-induced design damage? coming from a desire for isolation in TDD with the concept of ?speculative generality?. Are the similar? the same? why/why not?

(f) What is cohesion? What is coupling? In what ways can more of one mean less of the other?

(g) Kent says,

Difficulty testing is a symptom of poor design.

Do you agree?

(h) Kent says he is optimistic in that he believes that there is always some design insight that exists that will result in a design that is simultaneously more easily testable as well as better structured.

Do you share his optimism?

**Part 3** (a) Martin identifies three important forms of feedback for software development. What were they, and how well is each one served by having a unit test suite?

(b) How has Quality Assurance (QA) for programming changed over time? In particular, what role did TDD and self-testing code play in this change?

(c) What does David mean when he refers to ?criticality?? How should the ?criticality? of your software influence your testing?

(d) Describe how it's possible for code with 100% test coverage with all tests passing can still have bugs.

**Part 4** (a) Kent mentions a good measure for understanding how important a given test is for a codebase. What is it? How could you determine its value for a given test? Using this measure, how could you numerically define ?overtesting??

(b) Describe Martin?s trick used to identify if a given line of code is tested by the current test suite.

(c) How is the ratio of functional code to test code influenced by coupling in the system?

(d) What are the symptoms of overtested code? undertested code?

(e) How does the ?audience? of a codebase change its testing tradeoffs?

(f) Describe the difference between placing unit tests as the ?authority? of the system differs from placing the functional code as the ?authority? of the system. Which places a larger focus on refactoring?

(g) What is one situation where you would be more happy to lose the tests and keep the code? What is one situation where you would be more happy to lose the code and keep the tests? What is different about these two situations?

(h) David thinks the testing battle has been won, and Martin isn?t so convinced. Do you believe that we?ve ?won? the battle to get people to unit test their code? Why or why not?