String Calculator Kata

TDD Workflow Basics

Presented by David Andrews

Four steps in this Kata

I’m going to give you some directions for what your solution should do.

In response, you should:

1. Write test(s) to translate my request to code
2. Demonstrate to yourself that those test(s) fail
3. Implement mods to make the tests pass
4. Share your implementation with the team

TDD is Test-Driven Development

1. Implement the simplest code you can imagine to make the tests pass,
2. Once the tests pass, *refactor* the code to simplify it,
3. Let earlier tests stand, they will catch regressions, and
4. Discuss edge cases and obscure aspects with the customer or your team!

We want an emergent design

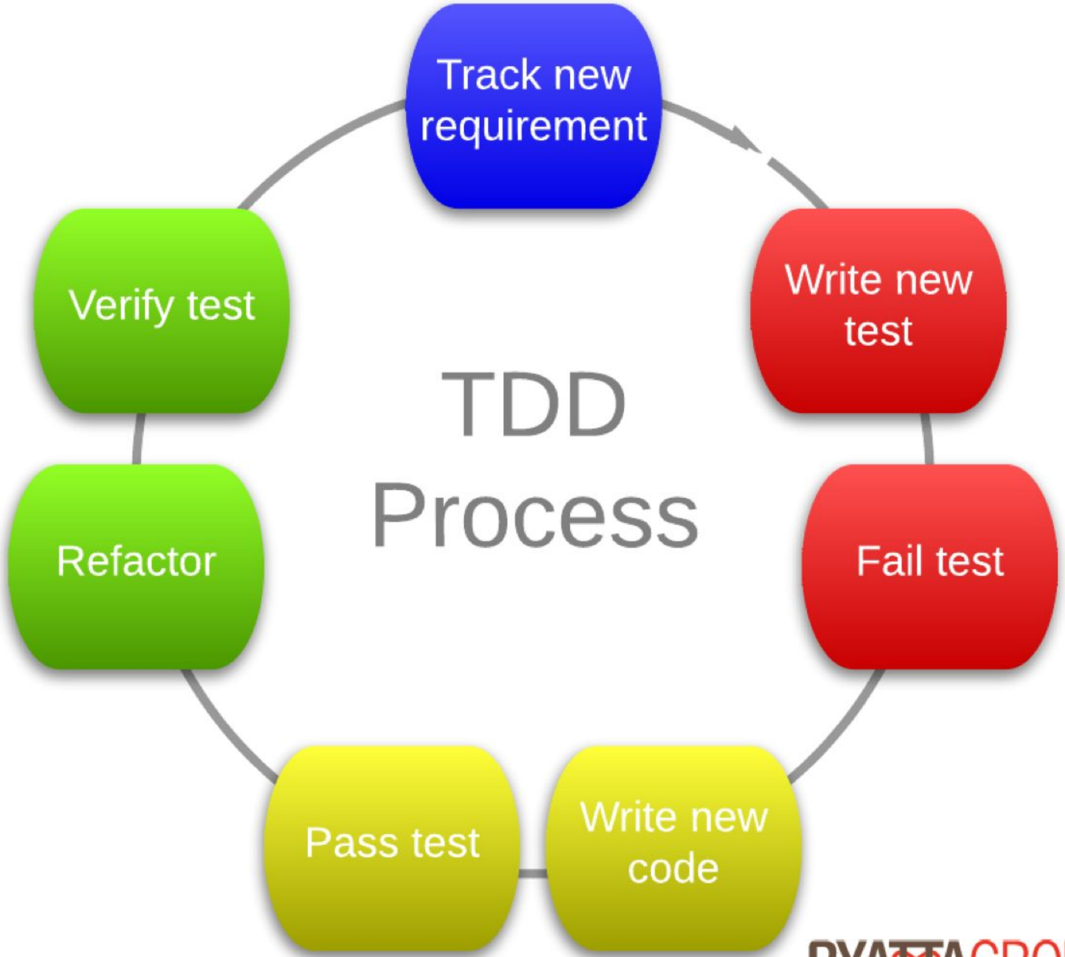
- The more code we have, the bigger the refactoring step will be
- Our design is constantly evolving and under constant review (not predetermined)
- TDD is a way to deliver high-quality code through *Emergent Design*
- Focus on what code does, not what it is

Requirement 1

If I give you an empty string, I want your calculator to answer 0.

More formally

- Today we are practicing the TDD process using RSpec
- TDD is a subset of BDD, a process which adds Cucumber to the mix
- Cucumber is to applications what RSpec is to entities



RYATTAGROUP

Requirement 2

If I give you a single digit, I want your calculator to respond with that number.

For instance:

“0” should produce the value 0, and

“5” should produce the value 5.

Requirement 3

If I give you two single digits separated by a comma, I want your calculator to respond with the sum of those numbers.

For instance:

“1,2” should produce the value 3, and

“7,9” should produce the value 16.

Requirement 4

If I give you two strings of digits separated by a comma, I want your calculator to answer with the sum of those numbers.

For instance:

“12,45” should produce the value 57, and

“42,159” should produce the value 201.

Requirement 5

If I give you any number of strings of digits separated by a comma, I want your calculator to answer with the sum of those numbers.

For instance:

“1558,2,2442” should produce the value 4002,

“15,22,45,79” should produce the value 161.

Requirement 6

I forgot, sometimes I’ll give you a newline character instead of a comma. You should treat it the same as a comma.

For instance:

“1\n2,3” should produce the value 6.

Requirement 7

On second thought, maybe we should make it so delimiters can be set in the input.

Whenever I give you two slashes followed by a character on the first line of the input, use that character as the delimiter.

“//;\n2;5” should produce the value 7.

Requirement 8

Oh damn, I forgot about negative numbers. They’re a problem, so let’s just throw an exception when you see one in the input.

Requirement 9

The exception message would be a good place to watch for problems, so let’s put a list of the negative values in there, but only when there are multiples.

Requirement 10

Sometimes the input gets a little garbled, so let’s ignore any numbers larger than 1000.

Requirement 11

Ok, we’re seeing some strange inputs recently. Can we make it so delimiters can be multiple characters long?

Let’s agree the first line will have square brackets around the delimiter like this:

“//[delimiter string]”

Wrap Up

Ok, so what did we learn here today?

Final Comments

DHH has come out to say that he doesn't follow TDD. His main concerns were with regard to the unintended impacts on system design that TDD were creating. His preferred approach is to focus on System Testing.

Stay tuned to the Ryatta blog for an informative post about this very topic (how timely!)