The Craftsman: 39
Dosage Tracking XVI
Test Refactoring

Robert C. Martin
7 June 2005

...Continued from last month.

By hook, and by crook, President Wallace poured *money into project Nimbus. He directed the Contingent to develop technologies pursuant to: 1. Saving the human race from Clyde. 2. Defending this effort against the Axis powers.*

The latter of these two imperatives enjoyed more short term success than the former. Technologies for weapons systems were created and deployed in absurdly short time frames. A now famous quote by Edward Teller summarized the period: "When you fight for a desperate cause and have good reasons to fight, you usually win."

By August of 1943, Von Braun's rockets were launching cameras into orbit to spy on Europe and Asia. Canisters of film were ejected from these satellites and recovered by aircraft that snagged them from the sky as they fell on their parachutes.

That's how the U.S. learned that the Axis was planning to invade Mexico.

---

22 Feb 2002, 0830

When I returned from the washroom, Jasper was still sitting at the workstation waiting for me. His smile had been replaced by a hang-dog grimace.

"Alphonse, look." He said with an "aw shucks" tone. "I'm sorry; I guess I was trying too hard. The truth is that I just recently made Journeyman status, and I've never had an apprentice before. Can we start over?"

I had been on the verge of apologizing myself, so I was glad he beat me to it. "Sure, Jasper, we can start over. Just ease off the attitude a bit, OK?"

"I'll try, Alphonse. And if I forget, don't hesitate to remind me. Deal?"

"Deal." I wasn't sure if this would last, but we had a lot of work to get done, and we didn't need personality problems getting in the way. "So where are we?"

Jasper turned to face the screen, and said: "We just got the `SuitRegistrationRejectedBy-Manufacturing` test running. So the next test we should work on is the rejection of a duplicate suit."

"You mean if someone tries to register a suite that's already registered, we should reject it? That makes sense."

"Right. So let me show you this neat new fixture that Rick Mugridge wrote. It's called `DoFixture`, and it really helps writing acceptance tests like this one."

"Rick who?"

"Rick Mugridge. He's an astrogator by training, but has a passion for programming. So lets start by creating the new test page."

Jasper opened the `RegisterSuit` page and made the following change:

```
|^RegisterNormalSuit|''The Happy Path.''|
|^SuitRegistrationRejectedByManufacturing|''Manufacturing rejects the
registration.''|
|^RejectDuplicateRegistration|''You can't register a suit that's already
registered.''|
```

Then he clicked on the `RejectDuplicateRegistration` link and copied the following onto the new page:

```
!path C:\DosageTrackingSystem\classes

!3 DTrack rejects a duplicate suit registration.

!|Import|
|dtrack.fixtures|

''We assume that today is 2/21/2002.''
!|DTrack Context|
|Today's date|
|2/21/2002|
```

I pointed to the first line on the screen and said: "All three of our test pages has that `!path` line. Is there a way to put it in one place?"

Jasper winked and said: "Sure beans, Fon...Alphonse. We can just move it to the parent page." So he made the following change to the `RegisterSuit` page, and removed the `!path` line from the three subpages.

```
!path C:\DosageTrackingSystem\classes

|^RegisterNormalSuit|''The Happy Path.''|
|^SuitRegistrationRejectedByManufacturing|''Manufacturing rejects the
registration.''|
|^RejectDuplicateRegistration|''You can't register a suit that's already
registered.''|
```

Then Jasper typed the following table into the `RejectDuplicateRegistration` page:

```
|Do Fixture|
|start|dtrack.fixtures.DTrackFixture|
|set suit|314159|as registered|
|check|register suit|314159|false|
|check|was a message sent to manufacturing|false|
|check|count of registered suits is|1|
|check|error message|Suit 314159 already registered.|true|
```

"OK, that's different." I said. "Let's see if I understand it. First you force suit 314159 into the registration database. Next you try to register 314159, and expect it to fail because it's a duplicate. You make sure that no message was sent to manufacturing, that the number of suits in the database is still one, and that an appropriate error message was created."

Jasper's smile was back. "Right you are Alphonse old sport!"

Maybe Jasper just couldn't turn his attitude off. I ignored it and asked: "This is pretty simple to understand, but what's that `start` line at the top?"

Jasper raised his eyebrows twice and said: "Good catch! That names the class, `DTrackFixture`, that has all the methods that this fixture will call."

I didn't understand this and must have looked puzzled because Jasper said: "I'll show you. Go ahead and click the test button." I obeyed and saw:

```
Could not find fixture: DoFixture.
```

"Right," said Jasper. "You've got to import the package `fitLibrary`."

So I added `fitLibrary` to the `Import` table as follows:

```
!|Import|
|dtrack.fixtures|
|fitlibrary|
```

Now when I hit the test button I got a whole bunch of error messages. The first was:

```
Unknown class: DTrackFixture
```

Before Jasper could comment I said: "OK, so I need to create a class named `dtrack.fixtures.DTrackFixture`, right?"

"Correctomundo, Al! Uh, Alphonse."

He was clearly making an effort. I was surprised that it was so hard for him, but I made no comment. I just typed:

```
package dtrack.fixtures;

public class DTrackFixture {
}
```

That made the first error message go away. The next message was:

```
Unknown: "setSuitAsRegistered" with 1 argument
```

Again, before Jasper could comment I said: "OK, this must be a method of `DTrackFixture`, right?" I could see him starting to wind up to deliver another syrupy answer so I held up my hands and gave him a meaningful look. He paused, puzzled; then gave me a sardonic smile, and simply said: "Right." "Progress!" I thought to myself. I typed the method:

```
public class DTrackFixture {
  public void setSuitAsRegistered(int barcode) {

  }
}
```

That made the error message go away. Likewise I was able to make all the other error messages go away one by one. Eventually the fixture looked like this:

```
public class DTrackFixture {
  public void setSuitAsRegistered(int barcode) {

  }

  public boolean registerSuit(int barcode) {
    return true;
  }

  public boolean wasAMessageSentToManufacturing() {
    return true;
  }
```

```
  public int countOfRegisteredSuitsIs() {
    return 0;
  }

  public boolean errorMessage(String message) {
    return false;
  }
}
```

Now when I hit the test button, there were no more error messages. Instead, I had red cells in the table that looked like this:

Do Fixture			
start	dtrack.fixtures.DTrackFixture		
set suit	314159	as registered	
check	register suit	314159	false expected true actual
check	was a message sent to manufacturing	false expected	true actual
check	count of registered suits is	1 expected	0 actual
check	error message	Suit 314159 already registered.	true expected false actual

Jasper shook his head as he looked at the screen. "You figured that out pretty fast, Alphonse. Do you think you can wire the fixture up to the application?"

There wasn't even a *hint* of condescension in his demeanor. I think he was actually impressed instead of pretending to be. "Sure, I think so."

The first method of the fixture was easy. All I had to do was jam the specified suit into the database:

```
public void setSuitAsRegistered(int barcode) {
  SuitGateway.add(new Suit(barcode, Utilities.getDate()));
}
```

Then I changed `DTrackFixture.registerSuit` to call `Manufacturing.registerSuit`.

```
public boolean registerSuit(int barcode) {
  return Utilities.manufacturing.registerSuit(barcode);
}
```

The `wasAMessageSentOtManufacturing` method simply checked to see if a message was sent.

```
public boolean wasAMessageSentToManufacturing() {
  return Utilities.manufacturing.getLastMessage() != null;
}
```

The `countOfRegisteredSuitsIs` method was also trivial.

```
public int countOfRegisteredSuitsIs() {
  return SuitGateway.getNumberOfSuits();
}
```

But I didn't know what to make of the `errorMessage` method. So I left it unchanged. When I hit the

test button, I saw:

Do Fixture			
start	dtrack.fixtures.DTrackFixture		
set suit	314159	as registered	
check	register suit	314159	false expected true actual
check	was a message sent to manufacturing	false expected true actual	
check	count of registered suits is	1	
check	error message	Suit 314159 already registered.	true expected false actual

I looked expectantly at Jasper. He said: "Hot ziggedy dog, Alphonse! -- er – Sorry, I mean, uh, good. But what about the `errorMessage` method?"

"I don't know exactly what to do for this method. It looks like you are trying to see if an error message was printed. But I don't know how to check that."

"Oh SURE you do, Alphonse! -- er – I mean, I think we want to make a mock object that represents the screen, and remembers the messages that were sent to it. The `errorMessage` method then simply asks if a particular error message has been sent to the screen."

Of the two Jaspers, I liked the second one better. His idea made sense. So I said: "OK, so we could create an `IConsole` interface that has a method like `display(String message)`. And then we can create a dummy `Console` that implements that interface and saves any message sent there. That sounds easy." So, bit by bit I wrote the following test, and the code that made it pass:

```
public class MockConsoleTest extends TestCase {
  private MockConsole c;

  protected void setUp() throws Exception {
    c = new MockConsole();
  }

  public void testEmptyConsole() throws Exception {
    assertEquals(0, c.numberOfMessages());
  }

  public void testOneMessage() throws Exception {
    c.display("message");
    assertEquals(1, c.numberOfMessages());
    assertTrue(c.hasMessage("message"));
    assertFalse(c.hasMessage("noMessage"));
  }

  public void testTwoMessages() throws Exception {
    c.display("messageOne");
    c.display("messageTwo");
    assertEquals(2, c.numberOfMessages());
    assertTrue(c.hasMessage("messageOne"));
    assertTrue(c.hasMessage("messageTwo"));
    assertFalse(c.hasMessage("no message"));
  }
}
```

```
public interface IConsole {
  public void display(String message);
}
```

```
public class MockConsole implements IConsole  {
  HashSet messages = new HashSet();
  public void display(String message) {
    messages.add(message);
  }

  public int numberOfMessages() {
    return messages.size();
  }

  public boolean hasMessage(String s) {
    return messages.contains(s);
  }
}
```

"OK, that's good." said Jasper. "Now let's see if we can get this acceptance test to pass."

The code for this article can be located at:
http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_39_DosageTrackingSystem.zip

To be continued...