The Craftsman: 30
Dosage Tracking VII
The “Woodshed”

Robert C. Martin
3 September 2004

...Continued from last month.

As 1939 waned, and Europe descended into the chaos of war, the Earth remained centered within Clyde’s narrowing path. Though the probability of a collision was still quite small, it continued to grow. The members of the Stockholm Contingent, communicating through Lise Meitner’s secret network, eventually concluded that their only responsible course was to act as though collision were certain.

The Contingent faced a dilemma. So far they had managed to keep the discovery of Uranium fission within their membership. However, if atomic power were to be used as a defense against Clyde, it would take the resources of a very rich nation to gain the necessary technology in time. In Europe, all such nations were at war, and would certainly use that technology to produce terrible weapons.

The Contingent resolved to go where the war had not yet reached. Leo Szilard, a founding member of the Contingent, convinced Albert Einstein to write a letter to Franklin Roosevelt on behalf of the Contingent. The import of this letter was underscored by the fact that a quorum of the Contingent had somehow managed to arrive on U.S. soil by the time the letter was delivered.

---

21 Feb 2002, 1230

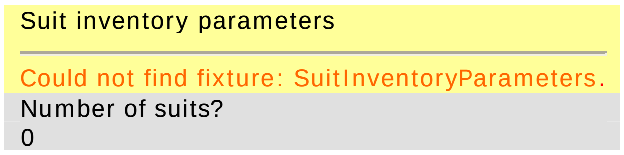
Jean took Avery to a small two-man conference room with glass walls. They were still in there when Jerry and I got back from lunch. The look on Avery’s face made me glad that *I* was not the one in that room with Jean.

Jerry said, “We’d better get back to work. Let’s see how much progress we can make by the time he gets out. “

“Do you think he’ll be back?” I asked?

Jerry glanced over at Avery and Jean with a knowing look on his face. “I’m pretty sure of it, Alphonse. Now let’s get to work.”

We sat at our workstation and Jerry ran the `RegisterNormalSuit` acceptance test. The first two tables showed no errors, but the third look like this:



Suit inventory parameters Could not find fixture: SuitInventoryParameters.
Number of suits?
0

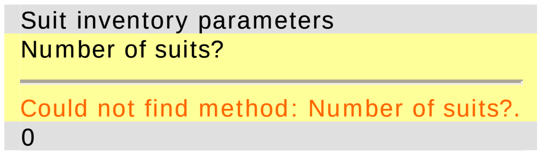
"So we need a fixture named `SuitInventoryParameters`, right?" I asked.

"Right" said Jerry. "Go ahead and see if you remember how to write it."

So I grabbed the keyboard and typed the following:

```
public class SuitInventoryParameters extends ColumnFixture {
}
```

Running the acceptance test now produced this:



Suit inventory parameters
Number of suits?   Could not find method: Number of suits?.
0

"That's not quite what I expected." I said. "The last time we did this[1] it didn't say anything about a method. It wanted a variable."

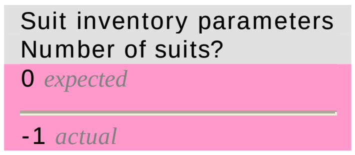
"That's right Alphonse, but this time it wants a method because there is a question mark following the name."

"Oh, right, you mentioned something about that earlier this morning. Question marks mean that the table is asking a question of the application. So we are asking the system how many suits it has in inventory?"

"Right. So go ahead and write that method."

```
public class SuitInventoryParameters extends ColumnFixture {
  public int numberOfSuits() {
    return -1;
  }
}
```

Now the acceptance test looked like this:



Suit inventory parameters
Number of suits?
0 *expected*   -1 *actual*

"Good." Jerry said. "Now connect the fixture to the application."

"What function in the application gets the number of suits in inventory?" I asked.

"Good question." Jerry replied. "What function do you think it should be?"

I looked over the code in the project for a few seconds and said: "We could put it in the `Utilities` class for now."

"We could. I don't think it will stay there long though."

So I modified the code as follows:

```
public class SuitInventoryParameters extends ColumnFixture {
  public int numberOfSuits() {
    return Utilities.getNumberOfSuitsInInventory();
```

[1] September, 2004, "Swiss Wisdom": http://www.sdmagazine.com/documents/s=7764/sdm0409l

```
  }
}

public class Utilities {
  public static Date testDate = null;

  public static Date getDate() {
    return testDate != null ? testDate : new Date();
  }

  public static int getNumberOfSuitsInInventory() {
    return -1;
  }
}
```

There was no change in the acceptance tests results. “Shall I make this table pass now?” I asked Jerry.

“No, let’s connect up the other fixtures first.” He replied.

The next table in the test looked like this:

Suit Registration Request
bar code
314159

Connecting it up to FitNesse[2] was easy.

```
public class SuitRegistrationRequest extends ColumnFixture {
  public int barCode;
  public void execute() {
    Utilities.registerSuit(barCode);
  }
}

public class Utilities {
  …
  public static void registerSuit(int barCode) {
  }
}
```

The next table was a bit trickier. It looked like this:

Message sent to manufacturing		
message id?	message argument?	message sender?
Suit Registration	314159	Outside Maintenance

The basic structure of the fixture was easy enough. The question marks told me that each of the column headers was a method. So I wrote the following.

```
public class MessageSentToManufacturing extends ColumnFixture {
  public String messageId() {
    return null;
  }

  public int messageArgument() {
    return -1;
  }
```

[2] www.fitnesse.org

```
  public String messageSender() {
    return null;
  }
}
```

But then I was stuck. “How do I connect this to the application?” I asked.

“Do you remember what this table is checking?”

“Sure, we’re verifying the contents of the message that DTrack is supposed to send to the manufacturing system.”

“Right. So you need to get that message, and unpack it.”

“How do I do that? None of the methods in this fixture seem to be the appropriate place.”

“I agree. They aren’t. However, FitNesse gives you another choice. The `execute` method in `ColumnFixture` is called before any column header methods are called. So in the `execute` method you could ask DTrack for a copy of the message that was sent, and then the column header methods could unpack that message.”

“OK, I think I get it.” And I changed the code as follows:

```
public class MessageSentToManufacturing extends ColumnFixture {
  private SuitRegistrationMessage message;
  public void execute() throws Exception {
    message = (SuitRegistrationMessage)
              Utilities.getLastMessageToManufacturing();
  }

  public String messageId() {
    return message.id;
  }

  public int messageArgument() {
    return message.argument;
  }

  public String messageSender() {
    return message.sender;
  }
}

public class SuitRegistrationMessage {
  public String id;
  public int argument;
  public String sender;
}

public class Utilities {
  …
  public static Object getLastMessageToManufacturing() {
    return new SuitRegistrationMessage();
  }
}
```

This made the table look like this:

Message sent to manufacturing		
message id?	message argument?	message sender?
Suit Registration *expected* null *actual*	314159 *expected* 0 *actual*	Outside Maintenance *expected* null *actual*

The next table was very simple. It looked like this:

Message received from manufacturing			
message id	message argument	message sender	message recipient
Suit Registration Accepted	314159	Manufacturing	Outside Maintenance

I connected it to DTrack using the following fixture:

```
public class MessageReceivedFromManufacturing extends ColumnFixture {
  public String messageId;
  public int messageArgument;
  public String messageSender;
  public String messageRecipient;
  public void execute() {
    SuitRegistrationAccepted message =
      new SuitRegistrationAccepted(messageId,
                                   messageArgument,
                                   messageSender,
                                   messageRecipient);
    Utilities.acceptMessageFromManufacuring(message);
  }
}

public class SuitRegistrationAccepted {
  String id;
  int argument;
  String sender;
  String recipient;

  public SuitRegistrationAccepted(String id, int argument,
                                  String sender, String recipient) {
    this.id = id;
    this.argument = argument;
    this.sender = sender;
    this.recipient = recipient;
  }
}

public class Utilities {
  …
  public static void acceptMessageFromManufacuring(Object message) {}
}
```

“The Utilities class is collecting an awful lot of cruft.” I complained.

“Yes, it is. We’ll go back and refactor it as soon as we get this acceptance test to pass. But for now let’s get that last table connected.”

I sighed and looked at the last table. It was a bit different:

Suits in inventory	
bar code?	next inspection date?
314159	2/21/2002

“It looks like this table could be asking for more than one suit.” I said.

“Well, the table only expects one suit, but one of the failure modes is that the number of suits is not one. You have to use a different kind of fixture for this. Let me show you.”

Jerry grabbed the keyboard and wrote the following:

```
public class SuitsInInventory extends RowFixture {
  public Object[] query() throws Exception {
    return Utilities.getSuitsInInventory();
  }

  public Class getTargetClass() {
    return Suit.class;
  }
}

public class Suit {
  public Suit(int barCode, Date nextInspectionDate) {
    this.barCode = barCode;
    this.nextInspectionDate = nextInspectionDate;
  }

  private int barCode;
  private Date nextInspectionDate;

  public int barCode() {
    return barCode;
  }
  public Date nextInspectionDate() {
    return nextInspectionDate;
  }
}

public class Utilities {
  …
  public static Suit[] getSuitsInInventory() {
    return new Suit[0];
  }
}
```

When Jerry ran the test, the table looked like this:

Suits in inventory	
bar code?	next inspection date?
314159 *missing*	2/21/2002

I looked at this code for a few minutes and then said: "I think I understand. You overrode the `query()` method of `SuitsInInventory` to return an array of `Suit` objects. You also overrode the `getTargetClass()` method to return `Suit.class`. Any item listed in the table but not present in the array is marked as missing."

"Right." Said Jerry. "Moreover, if the `query()` method had returned more than one `Suit` object, it would have been marked as extra."

"OK, so now we've got the whole test page connected to the DTrack application. Now let's make it pass."

"Right. I'll bet we can do that before Avery get's out of the woodshed."

"The woodshed?"

"That's what we call that little class walled conference room."

I looked over at Avery and Jean in the woodshed. It looked like a pretty heavy, and one-sided conversation.

To be continued...

The source code for this article can be found at:

`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_30_DosageTrackingSystem.zip`