The Craftsman: 40
Dosage Tracking XVII
Non-trivial trivialities

Robert C. Martin
21 July 2005

...Continued from last month.

In the winter of 1943, and the spring of 1944 the American spy satellites watched as the axis powers assembled a huge armada on the west coast of Africa. The Americans kept very close tabs on this build-up, and the scouting missions that were related to it. The Axis scouts were paying a lot of attention to the Yucatan peninsula. Their intent was obvious. The Axis powers were planning a major invasion of Mexico; probably in the hopes of establishing a base from which to directly assault the heartland of the U.S.

General MacArthur smiled as he perused the endless flow of satellite photos and intelligence reports. The straightforward actions of the enemy told him that they had no knowledge of eyes that were watching from above. And if they didn't know about the orbiting cameras, then they certainly didn't know that Von Braun's missiles could deliver atomic fire anywhere in the world. MacArthur's smile deepened. The options were endless! He didn't know what he would do next, but he would think of something.

---

22 Feb 2002, 0900

"Getting this test to pass shouldn't be to hard." I said. "Let's look at the first failure." I pulled the test up on the screen.

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

"OK, the first line to fail is the `check register suit` line. The registration should fail because 314159 is a duplicate suit. So we need to put the duplication check in to the function that the fixture is calling."

Jasper nodded sagely, and didn't say a word for a change. So I pulled up the `DTrackFixture` to look

at the `registerSuit` method.

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

I stared at this function for a few seconds, trying to piece together what it implied. Finally I said: "OK, `Utilities.manufacturing.registerSuit` is the function that sends the message to Manufacturing to validate that the suit is authentic, and that it was approved for outside maintenance."

"It doesn't seem to have the right name, does it?" asked Jasper.

"No, it doesn't. It should probably be called something like `requestApprovalForRegistration`. I'll make the change." So I quickly changed the name of the function.

"Let's follow that function and see if there are any other name changes to make." suggested Jasper. I agreed so I opened up the `Manufacturing` class.

```
public abstract class Manufacturing {
  public boolean requestApprovalForRegistration(int barCode) {
    SuitRegistrationMessage msg = new SuitRegistrationMessage();
    msg.sender = "Outside Maintenance";
    msg.argument = barCode;
    send(msg);
    return true;
  }

  protected abstract void send(SuitRegistrationMessage msg);
}
```

Jasper pointed to the screen. "Yeah, that `SuitRegistrationMessage` really ought to be `SuitRegistrationApprovalRequest`, shouldn't it?"

"I agree." And I quickly made the change. "Gee, we didn't pick our names well, did we?"

A big grin spread across Jasper's face. "Aw, Alphonse, don't sweat it! You've only been working on this code for a day. Look how quickly we spotted this naming issue, and how easy it was to resolve."

"Yeah, but if we'd thought about it a little more before we chose those names..."

"We wouldn't have as many tests passing as we do! And we wouldn't have been as sure about the names as we are now! C'mon Al! You know this!"

Jasper's grin was wider than ever. His eyes were sparkling. I held up my hand and said: "OK, OK, you're right. Cool down Jasper! And remember, my name is *Alphonse*."

Jasper caught himself, nodded, and then turned back to the screen. "Right. Sorry. OK, let's look inside the `SuitRegistrationApprovalRequest` class."

```
public class SuitRegistrationApprovalRequest {
  public String id;
  public int argument;
  public String sender;
  final static String ID = "Suit Registration";

  public SuitRegistrationApprovalRequest() {
    id = ID;
  }
}
```

I looked over the class and then said: "Yeah, that ID string isn't well named. It should be "`Suit Registration Approval`".

I was about to make the change but Jasper stopped me. "I rather doubt that. I don't think it should be

a string at all."

"What do you mean? What should it be?"

Jasper scratched his head for a second and said. "It could be a unique integer, or an enum, or, I suppose, it could even be a string. It just shouldn't be tied to the name of the class. I don't want to change the ID code just because we decide to change the name of the string."

"Oh yeah, SRP!" I said.

"Right! Single Responsibility Principle. *Good* Alphonse!"

I shot him a glance, and he wiped the growing grin off his face.

"We could still use a string." I said. "How about this." And I grabbed the keyboard.

```
final static String ID = SuitRegistrationApprovalRequest.class.getName();
```

Jasper looked at that for a second and said: "Hmmm. Clever. But it still changes whenever the class name changes."

"Yeah, but *we* don't have to make an explicit change to the source code!" I said enthusiastically.

"True, but since the string changes, it means that a cosmetic change to the source code could break the protocol with `Manufacturing`. The `Manufacutring` system would have to be recompiled and redeployed."

"Hmmm. Yeah, but if we're making changes to our code, won't `Manufacturing` have to recompile their system anyway?"

Jasper froze in his seat. A funny puzzled expression passed over his face. "Wait a darn second! We can't be sending *objects* to the `Manufacturing` system! We must be sending a packet based on XML or something!" Jasper sprang out of his seat and called over to Carole. "Carole, what is the format of the messages that we are sending back and forth to `Manufacturing`?"

Carole was working with Jean on something. She looked up and said "They are XML based. I put a description of the messages on the wiki yesterday. Check there." And then she resumed work with Jean.

Jerry and Avery were busy working on something just across the table from us. Apparently Jerry had heard Jasper's question because he looked up and said: "Yeah, I just read Carole's wiki entry. The XML messages are pretty straightforward Jasper. No attributes or anything; just one tag per field. The packet that's sent between the two systems is just raw text."

Jasper nodded his thanks and sat back down. "OK, that's good news. We can just forget about this issue for the time being."

I was puzzled. "Why can we forget about it? Don't we have to build the XML string?"

"Yes, but not right now. We'll write a translation layer that takes our message objects and converts them to XML later."

I thought about that for a few seconds and then said: "Oh, OK, so we'll implement that abstract `send` method in the `Manufacturing` class to translate the request into XML."

"Yeah, something like that."

I nodded. That made sense. Some derivative of the `Manufacturing` class could worry about XML later. We didn't have to worry about it now.

"OK, so now back to the original problem. How do we get this test to pass?"

Jasper said: "Go back to the original `DTrackFixture` class and look at the `registerSuit` method."

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

Jasper continued: "This fixture should not be calling `requestApprovalForRegistration`, should it? That function shouldn't be called by a fixture at all. Asking for approval is *part* of the registration process. We need some object to handle the *whole* registration process."

"Avery and I created just such an object yesterday afternoon! We called it the `Registrar`."

"Yeah, I know." Jasper blurted. "Jerry told me about it after dinner last night. That's when we decided that you and I should work together today."

I didn't care for the sound of that. These journeymen were talking about Avery and me behind our backs, making plans for us. I suppose that's part of their job, but I didn't like it. If they are going to make plans that concern me, they should *involve* me!

I shook off that thought and filed it for later. "OK, so I have an idea." I grabbed the keyboard and began to type.

```
public class DTrackFixture {
  ...

  public boolean registerSuit(int barcode) {
    return Registrar.attemptToRegisterNewSuit(barcode);
  }

  ...
}
```

```
public class Registrar {
  ...

  public static boolean attemptToRegisterNewSuit(int barcode) {
    return Utilities.manufacturing.requestApprovalForRegistration(barcode);
  }
}
```

Jasper half-smiled and half-smirked. The way he was shifting his eyes and jerking his head I knew that something was itching him. He was trying to keep from saying something. It occurred to me that I needed to get Jasper into a poker game. "Jasper, what are you smiling about?"

He actually burst out with a giggle, looked at me with that impossibly wide grin, and said: "Fonse....Alphonse, how long has it been since you ran a test?"

Ooops. It has been awhile. I'd done all those name changes. But those changes were trivial! I ran the tests.

One of the unit tests failed, and every acceptance test failed.

"Ouch!" I said.

"Ouch indeed!" Jasper said with an even bigger grin.

The code for this article can be located at:

`http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_40_DosageTrackingSystem.zip`

To be continued...
