The Craftsman: 33 Dosage Tracking X Cleanup on Aisle 10.

Robert C. Martin
4 December 2004

...Continued from last month.

The Heisenberg Uncertainty.

Werner Heisenberg loved Germany, but despised the Nazis. In 1939 he joined Neils Bohr's cell of the Contingent. It was gut-wrenching decision for him to leave Germany for the United States; and unfortunately it took him too long to make the decision.

The SS had never really stopped watching Heisenberg after the "White Jew" incident. The SS had also notice that the behavior of many German scientists had changed. When Von Braun went missing, the SS heightened their attention on other scientists. To their horror they found that Von Braun wasn't the only one missing. But they found Heisenberg at a bus stop.

The exodus from Germany had been carefully organized and synchronized. But Heisenberg's patriotic dilemma made him miss a beat. He was supposed to be on a particular bus, but found he could not make himself get aboard. He watched it go by. As he stood there at the bus stop, steeling himself to get on the next bus, the blackshirts found him.

Under the ministrations of experts – and the SS had experts – a man will reveal information without ever saying a word. Body-language reactions to well-timed pictures, or statements, will reveal what the man's tongue holds back. Heisenberg never uttered a word. He resisted with a will. One of his torturers was later heard to say: "The man fought well. His heart was in it."

He may have fought well, but what he revealed indirectly was enough. When Stalin later approached Hitler to extend their alliance, and offered the intelligence he had gathered from Nimbus as an inducement, Hitler could appreciate both the accuracy and value of that information.

In early 1941 the Quadripartite Alliance (The Axis) between Germany, Japan, Italy, and the USSR was formed; and the doom of the Eastern Hemisphere was sealed.

---

21 Feb 2002, 1500

"So," I said intelligently, "let's begin with that mock object. You're right, it shouldn't be building the message." I pulled up the code. It looked like this:

```
public class MockManufacturing implements Manufacturing {
    ...
  public void registerSuit(int barCode) {
    SuitRegistrationMessage msg = new SuitRegistrationMessage();
    msg.id = "Suit Registration";
    msg.sender = "Outside Maintenance";
    msg.argument = barCode;
    lastMessage = msg;
```

```
  }
    ...
}
```

"First of all", I continued, "the id should be set in the `SuitRegistrationMessage` constructor."

"Agreed" said Avery. And he started to type.

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

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

"OK, the tests still pass." Avery said. "Now let's look at the sender field. That shouldn't be set by the mock. Setting that field is a function that the production code should do."

"Agreed." I replied. "But, what part of the production code should do it?"

"Uh....Hmmm."

"Yeah. What are we really trying to accomplish here?"

"Well, the acceptance test is making sure that we built and sent the appropriate message to Manufacturing."

"Right!" Avery said. "So what we need to do is build and send the message in *production* code."

"OK, but we can't really send it to Manufacturing. We're just testing."

"Right, so we create a Mock that overrides just the part that does the sending."

I saw the light. "Ah, OK, you mean Manufacturing is an abstract class that has a `send` method, and our mock overrides that method."

"Yeah, I think so."

"OK, so lets change `MockManufacturing` so that it's in the right form, and then we can move the methods to a base class." I began to type.

```
public class MockManufacturing implements Manufacturing {
  private Object lastMessage;

  public void registerSuit(int barCode) {
    SuitRegistrationMessage msg = new SuitRegistrationMessage();
    msg.sender = "Outside Maintenance";
    msg.argument = barCode;
    send(msg);
  }

  private void send(SuitRegistrationMessage msg) {
    lastMessage = msg;
  }

  public Object getLastMessage() {
    return lastMessage;
  }
}
```

"OK, the tests still pass. Now let's move the `registerSuit` method up into a base class."

"What should the name of that class be?" Avery asked.

"Manufacturing, of course."

Avery shook his head and pointed to the screen.

"Oh!" I said, "There's already a `Manufacturing` interface." OK, let's just make it a class and *then* move the `registerSuit` method up into it".

Avery grabbed the keyboard and started to type.

```
public abstract class Manufacturing {
  public void registerSuit(int barCode) {
    SuitRegistrationMessage msg = new SuitRegistrationMessage();
    msg.sender = "Outside Maintenance";
    msg.argument = barCode;
    send(msg);
  }

  public abstract Object getLastMessage();
  protected abstract void send(SuitRegistrationMessage msg);
}
```

```
public class MockManufacturing extends Manufacturing {
  private Object lastMessage;

  protected void send(SuitRegistrationMessage msg) {
    lastMessage = msg;
  }

  public Object getLastMessage() {
    return lastMessage;
  }
}
```

"That's much better." I said, as I watched the tests pass. "But I don't like that getLastMessage method in `Manufacturing`. It seems to me that's just a methods for the tests to use and should only be in `MockManufacturing`."

"I think I agree with you." Avery said. So he removed the `getLastMessage` method from the `Manufacturing` class.

"Ack! Now it won't compile."

"Yeah, that Utilities class depends on it." I said.

```
public class Utilities {
 ...

  public static Object getLastMessageToManufacturing() {
    SuitRegistrationMessage message =
      (SuitRegistrationMessage) manufacturing.getLastMessage();
    return message;
  }
...
}
```

"And who call that method." Avery asked.

I grabbed the keyboard and did a where-used search. "Just the `UtilitiesTest` unit-test class, and our `MessageSentToManufacturing` fixture. We can fix it, just by casting the `manufacturing` variable to a `MockManufacturing`." So I typed:

```
  public static Object getLastMessageToManufacturing() {
```

```
    SuitRegistrationMessage message = (SuitRegistrationMessage)
      ((MockManufacturing)manufacturing).getLastMessage();
    return message;
  }
```

"OK, that's really ugly." I said, as I watched the tests pass.

"Yeah, let's get that method out of there. We should be able to have both the fixture and the unit test call the manufacturing object directly. Since they both know that the manufacturing object is a `MockManufacturing`, we shouldn't need the cast."

Avery grabbed the keyboard and changed the unit test first.

```
public class UtilitiesTest extends TestCase {
  ...
  public void testRegisterSuitSendsMessageToMfg() throws Exception {
    MockManufacturing mfg = new MockManufacturing();
    mfg.registerSuit(7734);
    SuitRegistrationMessage message =
      (SuitRegistrationMessage) mfg.getLastMessage();
    assertEquals("Suit Registration", message.id);
    assertEquals("Outside Maintenance", message.sender);
    assertEquals(7734, message.argument);
  }
}
```

"OK, that still works." I said. "But it doesn't really make sense to keep it in `UtilitiesTest` does it?"

"No, clearly not." And Avery kept typing. He moved the `testRegisterSuitSendsMessageToMfg` method to a new class named `ManufacturingTest`. All the tests passed.

Avery was on a roll. "Next, we want to change the fixture." He kept typing.

```
public class Utilities {
  ...
  public static MockManufacturing manufacturing = new MockManufacturing();
  ...
}
```

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

"The tests still all pass." He said. "So we should be able to get rid of that `getLastMessageTo-Manufacturing` method." He did a quick where-used to prove that nobody called it, and then deleted it.

"We should be able to get rid of `Utilities.registerSuit` the same way!" I exclaimed, and I grabbed the keyboard. I removed the offensive method, and changed the `SuitRegistrationRequest` fixture to call `registerSuit` through `Utilities.manufacturing`. The tests all still passed.

The `Utilities` class now looked like this.

```
public class Utilities {
  public static Date testDate = null;
  private static SuitGateway suitGateway = new InMemorySuitGateway();
  public static MockManufacturing manufacturing = new MockManufacturing();
```

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

  public static int getNumberOfSuitsInInventory() {
    return suitGateway.getNumberOfSuits();
  }

  public static void acceptMessageFromManufacuring(Object message) {}

  public static Suit[] getSuitsInInventory() {
    return new Suit[0];
  }

  public static void addSuit(Suit suit) {
    suitGateway.add(suit);
  }
}
```

"Heck," I said, "we could get rid of `gutNumberOrSuitsInInventory`, and `addSuit` the same way!" And I kept frantically typing.

"This is *much* better." Avery said. "That `Utilities` class is starting to disappear. It really needs to disappear too."

"Yeah, I agree. But I don't think we can get rid of any more of it just now. So why don't we look at that argument variable in the `SuitRegistrationMessage` class."

But before we could get started Jerry came back.

"Hi guys, how'd it going?"

"Pretty good" we both replied.

"Did you get that test page to pass yet?"

"We'll..." We looked at each other. "We made some progress on that, but then we decided to refactor the code a bit. It was getting pretty ugly."

Jerry raised his eyebrows and said: "You...what?"

"Well, we got rid of most of the `Utilities` class, and...

"You....refactored? Let me take a look.

We showed Jerry our changes. He nodded at each one. Then he said, "Nicely done guys; nicely done. This code is in much better shape than when I left it an hour ago. I think this'll help us make much faster progress now."

Avery and I looked at each other and smiled. I felt pretty good about the cleanup we'd done. It was good to know that Jerry thought so too.

"OK, guys, I'm going to go talk to Carole for a bit. Why don't you get this page to finally pass."

To be continued...

The source code for this article can be found at:

`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_32_DosageTrackingSystem.zip`