The Craftsman: 34
Dosage Tracking XI
That's Final.

Robert C. Martin
19 January 2005

...Continued from last month.

The Beaches.

Without an eastern front to worry about, Hitler could focus his attention on Britain. He reactivated operation sea-lion; and this time he did not waver. The Luftwaffe paid a high price, but by sheer weight of numbers Goering finally reduced the RAF to tatters. The Royal Navy also forced the Germans to pay dearly for their gains; but in the end fared no better. By the late spring of '42, invasion seemed inevitable.

The United States longed to help, but was in political turmoil. The growing "friendship" between Stalin and Hitler, and the formation of the Axis powers, caused FDR to think again about the accusations against Alger Hiss. He authorized an investigation into the possibility that the Soviets had spies in US government. This investigation yielded terrifying results.

So as the German invasion barges crossed the channel from Le Havre to Brighton, the US was caught in the midst of a deep political purge from which it could not afford to be distracted. The UK was left to face the enemy alone.

They fought on the beaches. They fought on the landing grounds. They fought in the fields and in the streets, and in the hills. But in the summer of '42, after Churchill was killed by a well-aimed artillery shell, Britain finally fell to the Third Reich.

---

21 Feb 2002, 1600

Avery and I gave each other a conspiratorial glance.

"Let's run the test and see what's left to do." I said.

"Roger that." said Avery as he pushed the test button on the FitNesse page. This is what we saw:

Normal suit registration.

```
Import
dtrack.fixtures
```

We assume that today is 2/21/2002.

```
DTrack Context
Today's date
2/21/2002
```

We also assume that there are no suits in inventory.

Suit inventory parameters
Number of suits?
0

We register suit 314159.

Suit Registration Request
bar code
314159

DTrack sends the registration confirmation to Manufacturing.

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

Manufacturing accepts the confirmation.

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

And now the suit is in inventory, and is scheduled for immediate inspection

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

"OK", I said, "it's the last two tables that matter. The `MessageReceivedFromManufacturing` fixture needs to call something that puts the confirmed suit into the inventory; and then the `SuitsInInventory` fixture needs to read out the entire inventory."

"Right." Avery replied. "And suit number 314159 should be the only suit in there."

"Yah. So what does that `MessageReceivedFromManufacturing` fixture look like?"

Avery pulled it up on the screen.

```
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);
  }
}
```

"Ew, yuk, it's another one of those `Utilities` methods that we need to get rid of" complained Avery.

"Yeah, but let's not get rid of it until we make it pass. Can you pull up the `Utilities` class?"

```
public class Utilities {
  ...
  public static SuitGateway suitGateway = new InMemorySuitGateway();
```

```
  ...
  public static void acceptMessageFromManufacuring(Object message) {}
  ...
}
```

"Well! That explains why the suit is missing. The `acceptMessageFromManufacturing` method doesn't do anything."

"So, do you think we should write a unit test for adding a suit to inventory?" Avery asked.

"I think we already have one." I said, as I grabbed the keyboard and brought up one of the unit tests.

```
  public void testOneSuitInInventory() throws Exception {
    Utilities.suitGateway.add(new Suit(1, new Date()));
    assertEquals(1, Utilities.suitGateway.getNumberOfSuits());
  }
```

"Oh yeah, I forgot about that." Said Avery. "So now all we have to do is call that `add` method in `suiteGateway`."

"That's the way I see it too." And so I typed the following.

```
  public static void acceptMessageFromManufacuring(Object message) {
    SuitRegistrationAccepted suitAck = (SuitRegistrationAccepted) message;
    Suit acceptedSuit = new Suit(suitAck.id, getDate());
    suitGateway.add(acceptedSuit);
  }
```

"Whoops, that doesn't compile!" I said. "The `id` field must not be public." So I made the appropriate changes.

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

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

"OK, now it compiles, and the unit tests still pass." I said. "Now let's see what the FitNesse test is doing." And I hit the test button on the FitNesse page; but there was no change.

I was so focused on why the test results hadn't changed, that I forgot about Avery and started mumbling to myself. "Oh! I forgot to change the `SuitsInInventory` fixture." I started scrolling around in the screen without comment.

"Hay, slow down!" Avery said. "Remember me? I'm not real happy about making those variables public."

I found this annoying. We'd already discussed the difference between classes and data structures, and I didn't feel like having the debate again. I just wanted to get this FitNesse test to pass. So I said "Yeah." and just kept right on looking at `SuitsInInventory`.

```
public class SuitsInInventory extends RowFixture {
  public Object[] query() throws Exception {
```

```
    return Utilities.getSuitsInInventory();
  }

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

"Oh drat! It's another one of those `Utilities` functions." I mumbled.

Avery didn't get the hint. "I mean, shouldn't those variables be private?"

"Maybe." I said, and I brought up the `Utilities.getSuitsInInventory()` method.

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

"Oh, no wonder!" I subvocalized. "It's returning a dummy array." I started to change it; but the keyboard was suddenly yanked out from under my fingers. "Hay!"

"Hay, yourself!" said Avery brandishing the keyboard. "I'm part of this too. You can't just bulldoze your way past me. I don't like those public variables."

He was right, of course. I felt dumb. "Sorry." I said. "I just want to figure this out."

"Me too, but we have to work together, don't we?"

"Right, we do." I shook my head to clear it. "OK, the public variables, why don't you like them?"

"There are a lot of reasons that I don't like public variables. Frankly, I'm still reeling from this morning's conversation with Jerry and Jasmine about encapsulation."

"OK, but they made sense, right? I mean there really is a difference between a data structure and an object."

"Yeah, I can sort of see that; and I accept that `SuitRegistrationAccepted` is a data structure and not an object."

"So what's your beef?"

"Well, look at it. This class represents a message. The variables are loaded by the constructor. Once loaded, nobody should have the right to change them. Changing those variables would be like somebody intercepting one of my emails, and changing it before it was delivered."

"Yeah, OK, so we won't change them. Why do we have to make the variables private?"

"Don't you think it would express our intent better if the variables were private, or at least protected?"

"I see what you mean. By making them public we imply that they are changeable; by making them private we are telling other programmers that they shouldn't change them."

"Right, we'd supply a public accessor method like `getArgument()`, and everyone would know that those variables shouldn't be changed.

"Hmmm. I think there's another way to express your intent. Er, may I have the keyboard to show you?"

"Man, what a keyboard hog!" But Avery smiled as he handed the keyboard back to me. I typed:

```
public class SuitRegistrationAccepted {
  public final String id;
  public final int argument;
  public final String sender;
  public final String recipient;
  ...
}
```

Avery looked a the screen for a few seconds and then said: "I hadn't thought of it that way, but you're

right; this is better. It keeps the variables accessible, and yet no one can change them."

"Yeah, this is better than making them private and adding accessor methods. Now can we make this test pass?"

Avery grabbed the keyboard and said: "Sure, but I'm going to do the typing for awhile."

I rolled my eyes, but yielded.

"So," said Avery, "we need to return an array of `Suit` objects from the `getSuitsInInventory` method. That should be pretty easy." And Avery typed the following:

```
public static Suit[] getSuitsInInventory() {
    return suitGateway.getArrayOfSuits();
  }
```

```
public interface SuitGateway {
  public void add(Suit suit);
  int getNumberOfSuits();
  Suit[] getArrayOfSuits();
}
```

```
public class InMemorySuitGateway implements SuitGateway {
  private Map suits = new HashMap();
  ...
  public Suit[] getArrayOfSuits() {
    return (Suit[]) suits.values().toArray(new Suit[0]);
  }
}
```

And with that change, the unit tests, and the FitNesse page all passed.

"Pretty easy." I said.

"Yeah, but we've really got to get rid of that `Utilities` class." replied Avery. "Let's see if we can do that before Jerry tells us to quit for the day."

To be continued...

The source code for this article can be found at:

`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_34_DosageTrackingSystem.zip`