# The Craftsman: 31
# Dosage Tracking VIII
# Turn off this Force Field

Robert C. Martin
30 October 2004

*...Continued from last month.*

---

*The Nimbus Project was begun in the early months of 1940. By mid-year, as the Germans marched into Paris, and the skies over Britain were darkened by German bombers, General Leslie R. Groves and J. Robert Oppenheimer were establishing a huge facility at Los Alamos, New Mexico.*

*FDR took the war in Europe much more seriously than the threat of a "rock from outer space", but also understood what a gift the Contingent had bestowed upon America. The official charter of the Nimbus project was to develop atomic, electronic, and rocketry technologies necessary to defend the United States against foreign enemies. Indeed, this is all that that the generals and policy-makers really expected. It was just a side benefit that these technologies were exactly those that the Contingent insisted were necessary as a defense against Clyde.*

*Meanwhile Clyde's target ellipse continued to shrink, with Earth remaining near its center. As 1940 drew to a close the odds of a collision were still thousands to one against; yet when Werner Von Braun successfully launched an A4 rocket in the New Mexican desert, the target he had on his mind was not on Earth.*

---

*21 Feb 2002, 1330*

Jerry and I took a quick break. The sweet strains of *Turn off this Force Field* were lilting from the speakers in the break room. The sad melody made me think of Avery in the Woodshed with Jean. I wondered what was going on in there.

Avery and Jean were still in there when we returned from the break. Jerry and I looked at each other as we sat down, but said nothing about what was on both our minds. Instead, Jerry said: "OK, let's make this acceptance test pass. Here is the first table that's failing."

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

"Right." I said. "We can easily make it pass by just changing the appropriate method in the `Utilities` class." So I grabbed the keyboard and typed:

```
public static int getNumberOfSuitsInInventory() {
    return 0;
```

```
}
```

Sure enough the test turned green.

| Suit inventory parameters |
| --- |
| Number of suits? |
| 0 |

But Jerry was shaking his head. "No Alphonse, we don't want to make it pass that way. We want to make it *really* pass."

I pointed to the green cell on the screen and said: "What do you mean? It looks to me like the test is really passing."

"Acceptance tests aren't the same as unit tests. We don't use them for the same purposes. We use unit tests to help us design our classes and methods. We use acceptance tests to make sure that our system behaves as specified. Instead of simply making the acceptance test turn green, what we really want to do is develop the `getNumberofSuiteInInventory` method so that it works the way it should. And for that, we're going to need some unit tests."

"I'm not sure I follow you." I said. "Aren't we supposed to do the simplest thing we can think of to make our tests pass?"

"For unit tests that close to the truth. But we make acceptance tests pass with code that's been well unit tested."

"OK, I think I get it. If I'm tempted to do something ultra simple to get an acceptance test to pass, it means I should really be writing a unit test."

Jerry smiled. "That's a good way to think about it."

"OK, so let's add a test case to `UtilitiesTest.`" I took the keyboard again and wrote:

```
public void testNoSuitsInInventory() throws Exception {
  assertEquals(0, Utilities.getNumberOfSuitsInInventory());
}
```

This test passed immediately, as expected. So then I wrote:

```
public void testOneSuitInInventory() throws Exception {
  Utilities.addSuit(new Suit(1, new Date()));
  assertEquals(1, Utilities.getNumberOfSuitsInInventory());
}
```

This didn't compile because there was no `addSuit` method. So I continued to write:

```
public static void addSuit(Suit suit) {
}
```

And then I stopped. "How should we add the suit?" I asked.

"Good question." Said Jerry. "Where do you think it should go?"

"Well, I guess we need a database of some kind. Should we set one up now?"

"No, it's too early to do that now." Jerry replied.

"Well then I can't finish this method." I said.

"Sure you can." Jerry urged. "Just use an interface."

"An interface to what?" I didn't understand what he was telling me.

Jerry sighed and said: "An interface to a gateway."

The melodic softness of *Turn Off This Force Field* started running through my head again. Jerry's words weren't making any sense, so I just kind of stared at him while humming the melody in my head.

After about fifteen seconds Jerry rolled his eyes, grabbed the keyboard, and said:

"One way to deal with a database is to create an object called a gateway. A gateway is simply an object that knows how to operate on records in a database. It knows how to create them, update them, query them, delete them, and so forth. In this case we are going to create something called a TABLE DATA GATEWAY[1]." A TDG knows how to operate on rows within a particular database table."

Jerry typed the following:

```
package dtrack.gateways;

import dtrack.dto.Suit;

public interface SuitGateway {
  public void add(Suit suit);
}
```

"OK, I think I see." I said. So we should be able to modify the `Utilities` class like this." And I took the keyboard and typed:

```
public class Utilities {
...
  private static SuitGateway suitGateway;

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

Everything compiled, so I hit the test button by habit. We got a `NullPointerException` in `addSuit`, just as you'd expect.

"Now what?" I said.

"Create an implementation of `SuiteGateway` that keeps the suits in RAM." Said Jerry.

"We're not going to *leave* it that way, are we? I mean these suits *do* need to get written to a *real* database don't they?"

Jerry looked at me a little oddly and said: "What are you afraid of, Alphonse? Do you think we'll forget to store the data on disk?"

"No, it's just that this seems...I don't know...out of order somehow."

"Well, it's not out of order, let me tell you. One of the worst ways to design a system is to think of the database first. I know." Jerry glanced over to the woodshed for a moment. "Believe me, I know. What we want to do is push off decisions about the database for as long as possible. Eventually, we'll figure out some way to make sure the suits are stored on disk. I don't know whether we'll use a *real* database (whatever that is) or not. Perhaps we'll just take all the data in RAM and write it to flat files periodically. I just don't know right now, and I don't *want* to know. We'll work out those details as we go along."

I didn't like the sound of that. It seemed to me that you could paint yourself into a corner pretty quickly if you didn't think about the database up front.

"Having the Database Argument?"

It was Carole. She must have overheard us talking. Jerry nodded knowing and said: "Right on schedule."

Carole smiled and said: "Alphonse, remind me to tell you about the first Dosage Tracking System Jerry and I wrote a few years ago."

"Don't you dare!" Jerry retorted with mock fear and anger.

Carole smiled, shook her head, and walked back to her workstation.

---

[1] *Patterns of Enterprise Application Architecture*, Martin Fowler, Addison Wesley, p. 144

"OK, Alphonse, let's write a simple RAM based implementation of `SuitGateway`." So I took the keyboard and typed:

```
public class InMemorySuitGateway implements SuitGateway {
  private Map suits = new HashMap();
  public void add(Suit suit) {
    suits.put(new Integer(suit.barCode()), suit);
  }
}
```

Then I modified the `Utilities` class to create the appropriate instance.

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

And now the test failed with:

```
expected:<1> but was:<0>
```

"OK, Alphonse, you know how to make this pass, don't you?"

I nodded, and added the remaining code.

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

public interface SuitGateway {
  public void add(Suit suit);
  int getNumberOfSuits();
}

public class InMemorySuitGateway implements SuitGateway {
  private Map suits = new HashMap();
  public void add(Suit suit) {
    suits.put(new Integer(suit.barCode()), suit);
  }

  public int getNumberOfSuits() {
    return suits.size();
  }
}
```

And now all the unit tests passed, and so did the `suit Inventory Parameters` table of the acceptance test.

I looked as this code for a few seconds and then I said: "Jerry, I'm not so sure that this `Utilities` class is named very well."

"Really?" He said – though I could tell he was smiling inwardly.

"Yeah, and I'm not so sure that static functions like `getNumberOfSuitsInInventory` ought to be there either."

"What do you think it should look like?"

"Maybe we should rename `Utilities` to be `Gateways`. Maybe it should just have static variables holding all the gateway objects. And maybe people can make all their calls through the gateways." I said.

“Interesting idea.” Jerry said.

“Hi guys.”

It was Avery. Jerry and I quickly turned at the sound of his voice. Both Avery and Jean were standing there waiting to talk with us. I could faintly hear *Turn Off This Force Field* coming from the open door of the break room.

*To be continued...*

---

*The source code for this article can be found at:*
`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_31_DosageTrackingSystem.zip`