The Craftsman: 24 Dosage Tracking I OH NO!

Robert C. Martin
30 March 2004

...Continued from last month. You can download last month's code from:
www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_23_Raggedy.zip

---

20 Feb 2002,18:00

Whew! What an afternoon! I just got out of the strangest meeting I've ever been in. I had to write this down before dinner or else I'd forget the details. It started as Avery and I were on our way to the rim to play LandCraft in the game room. We had just gotten SMCRemote to do it's first remote compile, and we wanted to celebrate. We didn't get very far, though, because Jean pinged us just as we were leaving the lab.

"Hello my dears!" she said to both of us though our lapel coms. I hope it's not too much of a bother, but I wonder if you could join me up in the .36g lounge in alpha shaft. I've got something new for you two to work on, and…well…why don't you just come up here and I'll explain everything." Avery and I looked at each other, grimaced, shrugged, and headed for the alpha shaft turbo.

Jean must have pre-coded the doors, because they opened for us before we could knock. When we stepped in we saw Jasmine, Jerry, Jean, and two people I'd never met before. One was a tall middle-aged woman with striking red hair, and the other was a thin, bespectacled, and pleasantly smiling fellow just a few years older than I. Jasmine was standing and gathering her belongings. She appeared to be getting ready to leave.

Avery took one look at Jasmine, and his buggy eyes got even buggier. Apparently he'd never seen her before. It was good to see someone else's first reaction to her– it put things in perspective.

Jean beamed at us and said: "You boys made it down here in good time! I'm always so impressed by the vigor of young boys, aren't you Jasmine dear? Oh, but there I go gabbing away again and distracting everyone from the point. I just don't know why I do that."

She seemed to gather herself, and in that fraction of a second I could see a determination on her face that I'd somehow missed before.

"Alphonse, Avery, this is Carole and Jasper. Jasper is going to be working with you and Jerry on a new project. Carole is the sponsor for this new project. She'll be working very closely with you too, helping to define the requirements -- oh, you call them stories nowadays don't you. Goodness, I'm sometimes so forgetful. You'll be starting on this project first thing in the morning. Carole, why don't you summarize dear."

"Excuse me, Jean." I blurted. "But what about our SMCRemote project? We just got the very first compile to run and we – er, I – was looking forward to getting back to it." I looked at Avery for support, but didn't get it. He still had his eyes locked on Jasmine.

"Don't you worry about that, Aphonse dear. Jasmine will be taking that project over from you. She'll be working with Adelade. SMC remote is a wonderful learning project for new apprentices, but you boys have bigger fish to fry right now."

"That's right Hot Shot" Jasmine's eyes nailed me – again. "Don't you worry about SMCRemote. I'm going to go gather up Adelade now, and she and I will pick up the pieces. Good work on getting that compile running." She looked around at everyone in the room, tossing her midnight hair with each buoyant turn of her head. "See you all later." She said, and then strode purposefully out of the lounge.

I could feel the atmosphere in the room change in her wake. Avery's gaze followed her until she was gone. An odd smile grew on Jean's face, and then vanished. She turned to Carole and said: Carole…"

Carole stood and gestured for us to sit down. Jerry and Jasper were already seated. There were two chairs next to them arranged so that all four of us were facing Carole. As soon as we were seated she began. From her very first utterance you could feel ambition and energy exuding from her.

"Avery and Alphonse, I'm glad to meet you." Her voice was loud, and her tone direct. "Jean has been telling me some very impressive things about you. Jasper and Jerry, it will be a pleasure to work with you again. Last time we had a lot of fun, didn't we?"

Jerry fidgeted uneasily, but they nodded congenially at each other, and I could feel a kind of anticipation building.

"So, let me tell you what's been going on."

Jerry and Jasper fixed their full attention on her. Avery and I glanced at each other and followed suite.

"Three days ago the Hazard Avoidance guys detected a fluctuation in the starlight along our vector. The stars directly in front of us appear to be dimming and reddening. Spectro-analysis shows significant H_2 absorption lines in the dimmed stars. Apparently we are approaching a cloud of molecular hydrogen."

She let that sink in for a second. It didn't mean much to me, but Avery seemed to have recovered himself, and was very focused on her words.

"This dimmed patch of stars is growing perceptibly, meaning that it is close. The astronometry team says we're likely to enter this cloud in about two months."

Avery had his gaze fixed on Carole, and was holding his chin with his right hand. He moved that hand, pointing it at Carole and said:

"That shouldn't be a problem for us. Our Ice Shield will protect us from it. Molecular clouds aren't usually very dense after all."

Carole gave Avery an appraising glance and said: "That's true Avery. Those five cubic miles of ice running ahead of us will certainly deflect most of the H_2 particles away from the ship. Oh there will be some erosion, but astronometry doesn't think this cloud is deep or dense enough to make much of a dent in our shield. However, there does appear to be a risk of diffraction."

Avery nodded, trying to look knowledgeable, but it was clear that he was puzzled. Carole continued:

"From the point of view of the shield the cloud will appear to be a beam of neutral H_2 molecules all moving directly towards us at nearly light speed. The shield is a disk, and the beam will be diffracted around the edges of the disk. Some of those diffracted H_2 molecules will strike us. This won't cause a problem for the ship, or anyone on board. The skin of the ship is designed to absorb such particles. However, it does pose a slight risk for the folks doing maintenance outside."

"You mean radiation risk?" I said?

Jerry's head jerked in my direction, an unpleasant awareness beginning to dawn on his face.

"Precisely!" Carole beamed. Avery looked annoyed, perhaps jealous that I had pleased her. He said:

"So what do you need us for?"

Jerry's fists clenched.

"We need a new dosage tracking system." She said.

Jerry groaned. "Oh No, not again!"

Carole's smile nearly split her face in half. "Ah, I see you remember our last adventure with Dosage Tracking."

Jerry looked miserable. He hung his head and said: "I hoped I'd never see it again. Last time it was

the cleanup from a fuel spill, now it's diffracting molecular hydrogen. God, when will this application die?"

Jean walked over to Jerry and put her hand on his shoulder. "There, there, Jerry dear. We aren't going to make you work on that old abomination you wrote when *you* were an apprentice. This time you get a team, and the opportunity to do it*right*."

I wasn't sure what *right* meant, but by the sternly emphatic way she said it, it must mean something pretty significant. Jerry just sat there shaking his head breathing deeply. Carole looked almost triumphant. Jean looked kindly determined. The rest of us had no idea what was going on.

Carole continued. "First thing in the morning – 0800 -- we'll meet in your lab up on 44. At that time we'll go over the stories for the system, and plan the first iterations. We'll also sketch out some of the acceptance tests we'll be using – We *will* write those acceptance tests this time, won't we Jerry?"

Jerry looked more miserable than ever. He nodded while staring at the floor, while Jean smiled benignly next to him.

The meeting broke up pretty quickly after that. Avery and I were the last to leave the lounge.

"Do you have any idea what a Dosage Tracking system is?" I asked Avery.

"Of course I do." Avery said with a tone of annoyed superiority.

"What is it?" I asked.

Avery looked at me with almost a sneer on his face. "Go look it up, Hot Shot." And he strode off without a glance back.

Clearly something is bugging him, but I can't figure out what it could be.

To be continued...

The Craftsman: 25 Dosage Tracking II Register Suit

Robert C. Martin
28 April 2004

...Continued from last month.

---

21 Feb 2002, 0800

The impactor that set us on our journey was 22km in diameter, and moving at 53km/sec when it slammed into the Pacific. We had launched two months earlier and were in a parking orbit 60° ahead of the Earth, waiting for the inevitable. We knew that the impact would kick up a lot of debris so we didn't want to be in near-Earth space. I guess they figured 1AU was safe.

I've seen recordings of the impact. I don't want to discuss them. Communication with parts of the Earth continued for a few weeks, but steadily diminished and then ceased. I guess it wasn't much fun down there.

That was 43 subjective years ago, in 1959. Since then we've been star-hopping; looking for a suitable home. The first ten systems we've visited haven't been very promising. There are plenty of planets, but other than the ammonia breathing dribins of a Centauri 5, we haven't seen anything even close to a biosphere.

Now we're about to plow through a cloud of molecular hydrogen at nearly C, and to protect our ouside maintenance engineers we need to rewrite an old Dosage Tracking system that Jerry and Jasper were involved with some years back.

I walked into the lab on 44 just before 0800. Carole and Jerry were already there. Jerry looked resigned and was chatting with Carole about something that he clearly found embarrassing. Their conversation broke up before I could get close enough to join. A few minutes later, Jasper, Avery, and Jean walked in together. Avery gave me a nod and a smile as though nothing were wrong. Perhaps nothing was.

I wanted to talk to him but Carole convened the meeting before I could reach him. We all started moving towards the conference table at the end of the lab. I grabbed a chair, and Avery took the one next to me. He gave me a conspiratorial nod, just as Carole started talking.

"Jean and I have written up some initial stories for the new Dosage Tracking System. By the way, we're calling it 'DTrack'. I'll walk through the stories with you, and you can ask all the questions you like. The first story is 'Register Suit'."

Carol placed an index card on the table with the words "Register Suit" written on it.

"Our system tracks the radiation dosage received by outside maintenance workers by integrating the dosages received by the space suits that they wear. Each suit has a dosimeter integrated into it's systems. When a suit is checked out for use, it's dosimeter is read before the suit is released to the worker. When the suit is checked back in the dosimeter is read again. The difference is added to the total dose for that worker.

So the first thing we need is an inventory of suits. A suit is introduced to the system with this story."

Jerry spoke up. "I supposed we're using the bar code patches that are sewn onto the suit to identify them?"

"That's right, Jerry dear." Said Jean. "As I'm sure you remember the bar code contains a six character alphanumeric string that uniquely identifies each suit."

Jerry grabbed the card and wrote 'Bar Code Patch: X(6) on it.

Jasper poked Jerry in the ribs and said: "Stop writing COBOL on the cards, Jerry."

"I can't help it." He responded. "It's just the way I think."

I asked: "So when a new suit is produced, it is given a new bar code patch and then registered with the system? How does that registration take place?"

Jerry said: "Yeah, that's right Alphonse. The new suit is tagged and then entered into the system using a bar code reader."

Carole added: "The new suit is carried to the Outside Maintenance suit-up room by hand. The clerk there selects the "Register New Suit" function on the screen and then scans the bar code. This registers the new suit with the DTrack system, and sends a message back to manufacturing telling them that the new suit is accounted for."

Jasper grabbed the card and wrote: 'Register New Suit function: on screen. Send confirmation to mfg.'

"I've never understood why we don't create a suit clearing house." Jerry said. "It doesn't make any sense for maintenance to talk to manufacturing like that."

"Later, Jerry, later." Said Carole. "I know how you feel about this issue, and I agree. But the H_2 cloud is two months away and we *have* to be ready for it. Otherwise you'll be spending your shifts patching up the database corruption in that old COBOL system of yours, and I will too – and that's not the way *I* want to spend the next six to eight months."

Jerry grimaced, but nodded acceptance.

"Is that how big the cloud is; eight light-months?" asked Avery.

"That's astronometry's best guess as of last night." Replied Carole.

"Why does manufacturing need to know that the suit has been registered with DTrack?" I asked.

Jean replied: "Aphonse, dear, we track every suit from manufacturing through usage to decommissioning. When one department releases a suit another department registers it and the two departments exchange a message so that each knows what happened to the suit. That way we always know where the suits are. Can you imagine how awful it would be for the poor folks who have to use the suits if they didn't know the history of the suit, how old it was, what repairs have been done to it, what radiation exposure it has taken? Oh, I just don't want to think about it."

"What if production doesn't know about the suit that's being registered?" Avery asked.

"Production will send a denial message, and DTrack will not accept the registration." Carole replied.

I grabbed the card and wrote "Reject registration on denial" on it. Nobody seemed to mind.

"Will production send an acceptance message if they recognize the suit?"

"Yes dear." Jean responded.

Avery blurted: "What if there is no response from production?"

"Wait 10 seconds and then deny registration." Carole said.

Avery grabbed the card and wrote: '10s time out & reject'.

"What if the suit is already registered?" I asked.

"Reject the registration and do not send the confirmation message to production." Replied Carole.

I reached for the card but Avery was already writing: 'If already reg'd, reject reg & don't send conf to prod.'

"OK, one last thing." Said Carole. "Once registered, the suit should be scheduled for an inspection. This is just a flag in the database record that will prevent the suit from being checked out for use until it has been inspected."

Avery still held the card, almost as if he owned it. He scribbled: 'Sched for inspection' on it and continued to hold on to it with a proprietary demeanor.

"Are there any more questions about this story?" asked Carole. There was silence.

Jerry said: "OK, let's estimate it. Jasper, Alphonse, Avery, this story has a few complications, but it's pretty simple overall. I suggest we estimate it at four."

Jasper nodded, but I was confused. "Four what?" I asked. "Man hours?"

"No, just four." Replied Jerry. We don't assign units to these estimates; we just use them to compare one story to another. So a story that's twice as hard would be an eight. One half as hard would be a two."

Jasper reached for the card, and there was an awkward moment where it appeared that Avery would not relinquish it. But then, with a discernable grimace, he handed the card to Jasper. In the upper right corner of the card Jasper wrote the number four and drew a circle around it. Then he placed the card back on the table. Avery made a move towards it, but then thought better of it and backed off.

Carole said: "OK, now how do we test this?"

"Test it? What do you mean?" I asked.

Carole looked meaningfully at Jerry and said: "Would you care to answer your apprentice, Jerry?"

Jerry sighed and looked up at me with an air of inevitability. "Alphonse, Avery, so far you've been working on very simple systems that were designed more for your training than for actual use. DTrack is a production system, and the rules are a bit different. In a production system every requirement is specified in terms of executable acceptance tests. When the acceptance tests pass, then the requirement is done."

"Are acceptance tests like unit tests?" I asked.

"No, not at all. Acceptance tests look just like requirements. They can be read by all the stakeholders and officers. Most can write them too. Even Carole can write them." Jerry shot an evil glance towards Carole who responded by sticking out her tongue. "They are written in a system called FitNesse, which allows anyone with appropriate access to read them, write them, change them, and even execute them."

"How can they look just like requirements?" Avery asked, genuinely interested.

Carole stepped over and said: "You're about to find out. Jerry, let's write up an acceptance tests for Register Suit". And the two of them sat down at a terminal and began to type.

To be continued...

The Craftsman: 26
Dosage Tracking III
A Tabled Requirement.

Robert C. Martin
11 May 2004

...Continued from last month.

---

21 Feb 2002, 0900

Our ship, the *Dyson*, can accelerate at 1G for 60 months without refueling. During the first year of an interstellar trip we accelerate to nearly C. During the last year we decelerate to the reference frame of our destination. At speeds close to C the rest of the universe is subjectively so small that we can go just about anywhere in a few weeks or months. Theoretically that means we could go just about anywhere in the universe in two subjective years.

In the last 43 years we've stopped at ten stellar systems. We spend a year or so at each one, exploring, refueling and refitting. Then it's off to the next – in search of a home for the million frozen embryos on board. Usually we stop accelerating at about .999C, and coast for several months as a way to conserve fuel. There have been two stellar systems that did not provide us with enough Uranium to adequately refuel, and the captain isn't the kind of person to trade time for risk.

Right now we are at peak velocity, and have been coasting for a few months. We've got another eight months to go before we've crossed the 20 light-years of this jump, and start to decelerate. So we're going to plow through that molecular hydrogen cloud in front of us at a screaming clip.

Avery and I stood directly behind Jerry and Carole as they began to work on the acceptance test for the *Register Suit* story. Jerry opened a network browser and went to a site named FitNesse[1]. He did some things on the screen that I didn't follow but quickly wound up at a page whose name appeared to be *RegisterNormalSuit*.

"OK." He said. "Let's describe what normally happens when we add a suit.".

"What do you mean by normal?" asked Avery.

Carole said: "Nothing goes wrong. Everything works as it is supposed to."

"Right." Said Jerry. "So first we want to assert that the current suit inventory is empty."

"What's normal about that?" Avery sneered.

"Avery dear," Jean said soothingly, "we start by making the simplest assumptions. Don't worry, we'll also specify the cases where the database is not empty. Right now, though, it's just easier to assume that there aren't any suits in the database."

Jerry typed something and the following table appeared on the screen.

Suit inventory parameters

---

[1] http://fitnesse.org

Number of suits?
0

"What's that?" I asked.

"It's an assertion." Jerry responded. "I am asserting that there are no suits in the database. Next I want to assert that there is a request to register a new suit."

He continued to type, and another table appeared right below the first:

Suite Registration Request
bar code
314159

"I'm confused." I said. "Why is there a question mark in the first table, but not in the second?"

Jasper walked over and gave me a big toothy grin. "Woah, Jerry, you've got a bright one here. Good eye, Alphonse!"

Out of the corner of my eye I could see Avery stiffen. He didn't like Jasper complimenting me.

"The reason" said Jerry, "is that the first table is a query, whereas the second is a statement. In the first table we are asking the system how many suits are in the database. In the second table we are telling the system that suit number 314159 is being registered."

"That's right." blurted Avery quickly, "you see, Alphonse, the first table can be checked, but the second table is just a fact."

Jasper's eyebrows shot up. "Very *good* Avery! Gosh, Jerry, I think we've got two keepers here."

Avery stood next to me, smiling, but I could tell he was feeling superior.

"Wait." I said. "What do you mean the first table can be checked? That doesn't make a lot of sense to me."

Avery started to answer but couldn't seem to find the words. "Uh, well, the uh…"

Jerry stepped in and rescued him. "FitNesse is going to execute these tables." He said. "When it executes the first table it will check to be sure that the number of suits is zero. That's what the question mark tells FitNesse to do. If the number of suits is not zero, then that cell in the table will turn red, otherwise it will turn green."

Avery blinked, but didn't say anything. I, on the other hand, forged right on ahead. "You mean the cells change color?"

"Right." Said Jerry. He pointed to the screen that held the two tables. "Do you see the 'Test' button on the screen? When I push it, FitNesse will read the tables one by one. For each table it will pass some data into the DTrack system, and read some other data out. The question mark is for data that comes *out* of DTrack. If the data coming out matches the data in the table, then FitNesse turns the cell green. Otherwise it turns it red."

Avery muscled in again. "I see! So the first table *asks* DTrack how many rows are in the suit database, and will turn red if any number other than zero is returned. The second table *tells* DTrack to register suit 314159."

"That's about it." Said Jasper Jovially.

"Let's finish this test." Said Carole impatiently.

Jerry turned back to the console. "OK, so next we want to make sure that the appropriate message gets sent to manufacturing."

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

Avery looked puzzled. "So when FitNesse executes this table, will it ask Manufacturing what messages it received?"

"No," said Jasper. "We'll catch the message before it goes." Then he looked at Carole mischievously, and said "It would sure burn Courtney's breeches if we brought her system down by sending wild messages every time we tested, eh Carole?"

Carole rolled her eyes and said "Let's try to focus here. What's next?"

I said: "I guess we need to assert that manufacturing sends back an acceptance message."

"Right you are, Alphonse." Said Jerry as he typed in the appropriate table.

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

"You forgot the question marks." Avery said. You could tell he was pleased to catch Jerry in an error.

"Did I?" Jerry replied.

Avery looked uncomfortable. I thought I knew why Jerry had left the question marks off, but I held my peace.

Carole said: "No question marks are necessary because we are telling the DTrack system that Manufacturing sent this message. We aren't asking." Carole's patience was clearly wearing thin. She wanted to get this done. "What's next Jerry?"

"OK, having received that message, the DTrack system should enter the suit into the database. So now the database should have the suit in it."

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

"Why did you put today's date as the inspection date?" I asked.

Avery said: "Because, Alphonse, according to Carole's story, newly registered suits have to be scheduled for inspection."

"Yes, I remember that;" I replied, "but why *today's* date? The test won't work if we run it tomorrow. Are we going to have to change that date every day that we run this test?"

Jasper quipped "That's your job, Jerry. We want you to come in every morning and change the date."

"No thanks" Said Jerry. "No, we need to specify 'today's' date in the test. So let's do that as the first table."

DTrack Context
Today's date
2/21/2002

"OK." Said Carole; still trying to move things along. "I think that's the story. Now let's dress it up a little." She took the keyboard and began to write words around the tables. When she was done, the page looked like this:

Normal suit registration.

- *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	2/21/2002

"Great!" Said Carole. "A very nice requirement."

I had to admit, it was pretty clear. But there was something I still didn't understand.

"How do you get FitNesse to execute those tables?" I asked.

Jerry looked up and said: "Sit down, Alphonse; you too Avery; lets make this requirement turn red!"

To be continued...

The Craftsman: 27
Dosage Tracking IV
Carole's way, or the HighWay.

Robert C. Martin
28 May 2004

...Continued from last month.

---

21 Feb 2002, 1000

In the first decade of the 20th century, Percival Lowell predicted the existence of a ninth planet by studying the motion of Uranus. Though he searched, he died before he could complete the effort. In 1929 a young man named Clyde Tombaugh came to Lowell Observatory and resumed Lowell's search for planet X. The procedure was both delicate and tedious. Two photographic plates, taken of the same stretch of sky, but on different nights, were put into a device known as a "blinker". The blinker displayed the two plates one at a time, quickly alternating between them. Any object on one plate that was in a different position on the other would appear to blink. Clyde Tombaugh found Pluto on the 18th of February, 1930. In the summer of 1935 he found the angel of death.

The newspapers named the rock "Clyde", and had a few days poking fun at the notion that Clyde might actually hit the Earth in 1959. But the world was headed for war, and even a possible doomsday rock couldn't hold the papers' attention for long. Besides, as the astronomers kept saying, the odds of a collision were millions to one against.

Carole and Jasper grabbed another workstation to work on the next acceptance test, while Avery, Jerry, and I started working on making the *Register Normal Suit* test run.

"OK, the first thing we need to do is write the fixture for the first table." Said Jerry.

"What's a fixture?" I asked.

"A fixture is a Java class that binds the table to the DTrack application." Jerry replied.

"But there is no DTrack application." Avery complained.

"True." Said Jerry. "We write the tests and fixtures first to make sure that the application is designed to be testable."

"Oh, sort of like writing unit tests first." I said.

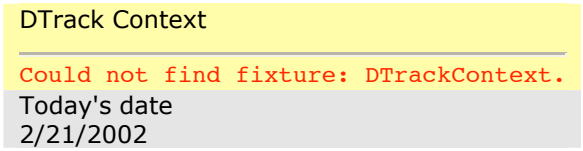
"Yes, it's a bit like that", replied Jerry, "except we write the whole test and fixture before writing any of the application. Indeed, we write many tests and fixtures before writing the application."

"What does a fixture look like?" Avery asked.

Jerry pulled up the test that he and Carole had just finished. "Look at the first table." He said.

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

"Now watch what happens to that table when I hit the *Test* button."



"Does that mean we have to write a class named DTrackContext?" I asked.

"That's exactly what it means." Jerry said, passing me the keyboard. "Put it in the dtrack.fixtures package.

So I wrote:

```
package dtrack.fixtures;

public class DtrackContext {
}
```

"Great." Said Jerry. "Now compile that and run the test again."

It compiled without a problem, of course; but when I hit the *Test* button, I got the same error as before.

"Do you know why?" Jerry asked.

I thought I did, and I started to answer, but Avery beat me to it by saying: "It's a classpath issue; FitNesse doesn't know where the `DTrackContext.class` file is."

"Right you are, Avery." Jerry beamed. I could see a smug little smile flicker on Avery's face.

Jerry took the keyboard and made the following changes to the test page:

```
!path C:\MyProjects\DosageTrackingSystem\classes

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

"This tells FitNesse what the classpath of the fixtures is." Said Jerry.

"But it still fails." Complained Avery, who had just hit the *Test* button.

Jerry looked expectantly at us.

"Oh!" I said. "The name of the class is `dtrack.fixtures.DTrackContext`, not just `DTrackContext`."

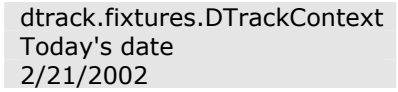
"Right again!" Said Jerry as Avery Scowled. "Why don't you make that change?"

So I changed the page to look like this:

```
!path C:\MyProjects\DosageTrackingSystem\classes

!|dtrack.fixtures.DTrackContext|
|Today's date|
|2/21/2002|
```

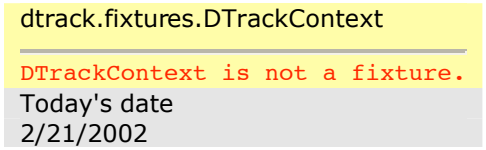
And it displayed like this:



"I can sort of see the syntax of this." I said. The strokes are table cell separators. But what is the bang (!) at the beginning?"

"Don't worry about that for now." Said Jerry. "You can read up on FitNesse in your spare time. The syntax is pretty easy to get used to. For now, let's just concentrate on getting this fixture written. So run the test."

I pushed the Test button, and saw a different kind of failure.



"Good!" Said Jerry. "It found the fixture class."

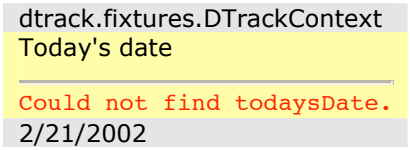
"Yeah, but it says it isn't a fixture." Avery whined.

"I can fix that." Jerry said, as he took the keyboard. He made the following changes to the fixture class.

```
package dtrack.fixtures;
import fit.ColumnFixture;

public class DTrackContext extends ColumnFixture {
}
```

"That's better!" Jerry said as he pushed the *Test* button.



"Not much!" Said Avery with a smirk.

"What is it looking for?" I asked? "A variable?"

"Bingo!" Said Jerry, who continued typing.

```
package dtrack.fixtures;
import fit.ColumnFixture;
import java.util.Date;

public class DTrackContext extends ColumnFixture {
  public Date todaysDate;
}
```

"Ick!" Cried Avery. "A public variable! That's not a very OO construct!"

Jerry looked calmly over at Avery and said: "So what?"

"It breaks encapsulation!" Avery sputtered with righteous indignation.

"Fixture classes aren't encapsulated in the usual manner." Jerry explained. "Public variables are one of the ways we communicate with them. Anyway this isn't the time for a lesson on the true principles of OO. Right now we want to get this fixture done." So he pushed the *Test* button as Avery rolled his eyes impatiently.

dtrack.fixtures.DTrackContext
Today's date
2/21/2002

Avery burst out with a huge guffaw: "Oh great! All that work to make it look normal again!".

"Right!" Said Jerry. "Now we know that the fixture is being found and that the data is getting into it as expected."

"It doesn't look as nice as it did before." I said. "That package name dirties things up a bit. I like the way the variable name uses spaces and punctuation. Can't the fixture name do the same?"

"Indeed it can!" Said Jerry, as he opened the test page and made the following changes:

```
!path C:\MyProjects\DosageTrackingSystem\classes

!3 Normal suit registration.

!|Import|
|dtrack.fixtures|

'' * We assume that today is 2/21/2002.''
!|DTrack Context|
|Today's date|
|2/21/2002|
```

This displayed as:

classpath: C:\MyProjects\DosageTrackingSystem\classes

Normal suit registration.

Import
dtrack.fixtures

- *We assume that today is 2/21/2002.*

DTrack Context
Today's date
2/21/2002

"OK, that's much nicer, I said. Now what do we do with that date?"

"Good point." Said Jerry. "We need to put that date somewhere that the rest of the DTrack system can get it from."

Avery assumed a superior air and said: "Wouldn't it be easier if DTrack just used the regular `Date` class to get today's date? Why do we have to invent a whole new mechanism for something as simple as today's date?"

"Because the tests need to control the date in order to make sure the application manages it correctly." Jerry replied, a little annoyed.

"Yeah, but that's just extra work that slows us down! We need to have this done in two months!" Avery had raised his voice enough for Carole to overhear. She quickly came over, looked Avery in the eye, and said:

"No, Avery, it's not extra work, and it doesn't slow us down. We go much faster when we write these

tests and build systems that are testable. You are right, Avery, we've only got two months. And the only way we'll make it is if we write the tests and follow our disciplines. We've done it both ways -- haven't we Jerry? (he grimaced, but nodded) -- and I can tell you that as long as *I'm* the customer on this project, *we're* going to be writing acceptance tests, and *you* will be doing it the way *Jerry* leads." And she strode back over to Jasper in a huff.

Avery had paled under Carole's tongue lashing. Now he looked at Jerry and me and said: "Whoa!"

"Yeah, she can be a little intense at times." Jerry said calmly. "The point is that we've decided to do things this way, and if you want to be part of the team, you'll have to go along."

"I still think it's a waste of time." Avery said under his breath.

"Suspend your disbelief for awhile." Said Jerry. "Trust me, we're not fools. This is the best way for us to proceed."

Avery shrugged, but nodded. He looked at me and rolled his eyes. I just stayed out of it.

"OK." I said. "Now what about that date? How does our fixture communicate it to the rest of the system, and how should the rest of the system gain access to the date?"

Jerry looked at the two of us, then glanced over his shoulder at Carole who had busied herself with Jasper, and then sheepishly he said: "I think it's time for a break. Let's get out of here for a few minutes and we can talk about why public variables are sometimes appropriate."

So the three of us left the lab and headed for the nearest lounge.

To be continued...

The Craftsman: 28
Dosage Tracking V
An Encapsulation Break

Robert C. Martin
29 June 2004

...Continued from last month.

---

21 Feb 2002, 1100

As tensions in Europe rose and eventually blossomed into war, Clyde was all but forgotten by the public -- but not by Tombaugh. He continued to track Clyde, computing and re-computing its orbit. His results were published in the appropriate journals, and were noted with growing concern by scientists on both sides of the growing political divide. The odds for a collision with Earth, though still very low, continued to rise with every new measurement.

By early 1939, when it was clear to most scientists that they would soon be unable to communicate with their colleagues on the other side of the Atlantic, Tombaugh decided to hold an astronomical conference in Stockholm. The conference was very well attended, and not just by astronomers. It seemed that scientists of many disciplines felt that this conference could be their last chance for an international gathering.

Clyde was not a major topic of the formal conference, but around the bar, and in the lounges, the talk often drifted to: "what if?" These discussions weren't serious so much as they were entertaining; but they served to generate some interesting ideas. Some were about destroying Clyde, others were about deflecting it, and still others were about escaping to Venus, or Mars.

Despite the range of ideas, everyone agreed that the fundamental problem was energy. All these solutions required energies that were beyond our abilities to generate; and so they were all considered to be bar-room fantasies.

Then, on the last night of the conference, Leo Szilard, Neils Bohr, and Lise Meitner broke the news that their investigations into nuclear reactions indicated that such energies just might be within reach. In view of the political climate, it was clear to everyone there that this news was not all that good. A few stayed very late that night to discuss…options. They whimsically named themselves: The Stockholm Contingent.

Jerry, Avery, and I went to the observation lounge at the high-gee end of the gamma arm. We each got a cup of coffee and sat at a table watching the starbow whirl under our feet.

Jerry looked over the table at Avery and said: "So, Avery, you complained about my use of public variables in the test fixture we were writing."

Avery grimaced and said: "Well, it's not a very OO thing to do."

"Can you define OO for me?" Asked Jerry? "Why are public variables not OO?"

"Because OO is about encapsulation, and public variables aren't encapsulated."

"Do you think that OO requires all variables to be encapsulated?"

"Of course!" replied Avery.

Jasmine was at a nearby table with Adelade. She seemed interested in this conversation.

Jerry asked: "What bad thing happens if you have an un-encapsulated variable?"

Avery made a disdainful face and said: "Other programs could change your variables!"

"What if that is what you intend? What if you *want* other programs to change certain variables?"

"Well then there's something wrong with your design!" Avery proclaimed with righteous indignation.

Jasmine loomed over our table and said: "Oh pooh, Avery, that's just silly."

Avery hadn't noticed Jasmine until that moment. When he saw her, the expression on his faced changed to a mix between embarrassment and panic. He sputtered as though to answer her, but Jasmine kept right on talking.

"Look you meatballs, it's not the rules, it's the reasons. The aim of OO is to help you manage dependencies. Most of the time making your variables private helps you to do that. But sometimes private variables just get in the way."

Jerry butted in: "Now wait a minute, Jasmine. I agree that OO languages help you to manage dependencies, but the more important benefit is expressivity. It is easier to express concepts using an OO language."

"Yeah, yeah, sure, sure." Replied Jasmine. But it's not lack of expressivity that turns software systems into a tangled mass of interconnected modules; it's mismanaged dependencies that do that. I can create very expressive systems that are tangled all to hell. I agree that expressivity is important, but keeping the coupling low is much more important!"

"How can you say that? You're approach is so sterile! Where is the art? Where is the finesse? All you care about is that the dependencies go in the right direction."

"What is artful about nicely named modules that are coupled into a tangled mess? The art is in the structure, Jerry. The art is in the careful management of dependencies between modules."

"Structure is important, sure, but…"

It seemed to me that this was likely to go on for awhile, and I really wanted to know about the public variables, because they bothered me too. So I interrupted them. "Excuse me, Jasmine, Jerry, but what does this have to do with public variables?"

The two of them looked at Avery and me as though they just remembered we were there. Jerry said: "Oh, uh, sorry about that. This is an old argument between Jasmine and I. Look, we make variables private to tell other programmers to ignore them. It's a way of expressing to others that the variables aren't their problem, and that we want them to mind their own business."

Jasmine rolled her eyes. "Oh, here we go again! Look boys, (I didn't like being called a boy – especially by Jasmine. I saw Avery flinch too.) we make variables private to reduce coupling. Private variables are a firewall that dependencies cannot cross. No other module can couple to a private variable."

"That's such an ice-cold view" said Jerry. "Don't you see how dehumanizing it is?"

I wanted to stop this before it got going again, so I blurted out my next question: "OK, but when is it OK for a variable to be public?"

Avery seemed to recover himself at that. He looked at me and proclaimed: "Never!"

That stopped Jasmine and Jerry in mid-stride. They both turned to Avery and said: "Nonsense – Ridiculous".

"Look, Avery" said Jasmine "sometimes you *want* to couple to a variable."

"Right" Said Jerry. "Sometimes a public variable is the best way to communicate your intent.

"When is that?"

"Well, like in a test fixture." Jerry replied. "The data used by the fixture comes from one source, but the test is triggered by another. One party has to load the data, and the other party has to operate on it. We want to keep them separate."

"Right" said Jasmine. "It's all about the coupling. The important thing is that the source of the data is

not coupled to the application being tested. The fact that the variables in the fixture are public is harmless."

"But what if someone uses them?" complained Avery?

"Who would?" Replied Jerry. "They are variables in a test fixture. Everybody on this project knows that those variables are under the control of FitNesse[1] and aren't for anyone else to use."

"But they might!"

Jasmine rolled her eyes and said: "Yeah, and they might take one of your private variables and make it public. What's the difference?"

Avery spluttered some more, but couldn't seem to face her down. So I asked the obvious question: "But couldn't you use setters and getters? Why make the variables public?"

"Exactly!" Avery splurted.

"Why bother?" Asked Jerry.

"Right," Said Jasmine. "It's just extra code that doesn't buy you anything."

"But I thought getters and setters were the right way to provide access to variables?" I replied.

"Getters and setters can be useful for variables that you specifically want to encapsulate." Said Jerry, but you don't have to use them for all variables. Indeed, using them for all variables is a nasty code smell."

"Right!" responded Jasmine with a grin. "PeeYew! There's nothing worse than a class that has a getter and setter for each variable, and no other methods!"

"But isn't that how you are supposed to implement a data structure?" I asked.

"No, not at all." Replied Jasmine. "A class can play two roles in Java. The first is as an object, in which case we usually make the variables private and provide a few getters and setters for the most important variables. The other role a class can play is as a data structure, in which case we make the variables public and provide no methods. Test fixtures are primarily data structures with a few methods for moving the data back and forth between FitNesse and the application under test."

"Right." Said Jerry. "A class that has getters and setters but no other methods isn't an object at all. It's just a data structure that someone has wasted a lot of time and effort trying to encapsulate. That encapsulation buys nothing in the end. Data structures, by definition, aren't encapsulated."

"Agreed." Said Jasmine. "Objects are encapsulated because they contain methods that change their variables. They don't have a lot of getters and setters because the data in an object remains hidden for the most part."

"Right." Said Jerry.

"Right." Said Jasmine.

And then the two of them smiled at each other in a way that made my stomach turn. Avery looked like he had just bitten into a lemon.

To be continued...

---

[1] http://fitnesse.org

The Craftsman: 29
Dosage Tracking VI
Move the Date

Robert C. Martin
6 August 2004

...Continued from last month.

---

21 Feb 2002, 1130

Between April and July of 1939 the political and astronomical news continued to worsen. War seemed inevitable, and the Earth remained at the center of Clyde's most probable path. The Stockholm Contingent changed from a small gathering of scientists in the bar, to a small and secret network of scientists around the world. Though the group had no official leader, it was Lise Meitner who provided vision and direction. Based in Stockholm, she remained able to communicate with most of her colleagues around the world. Slowly and carefully she began to recruit them.

By September, when the fits and starts of hostilities finally erupted into full-scale war in Europe, the Stockholm Contingent had members in neutral and hostile nations alike. This group did not know exactly what it was going to do, but they were convinced that the world should be paying more attention to Clyde than to fighting. They were also convinced that the newly discovered Uranium fission reactions were the key defense against Clyde, and too terrible to be used as a weapon of war. They also knew that the warring states would consider their views to be treasonous.

Jerry and Jasmine walked ahead of us on our way back to the lab. The whole way they talked and laughed, and acted all buddy-buddy. Avery and I looked disgustedly at each other, but kept quiet.

Back in the lab, we sat down and started working on the test fixture again.

"So, what are we going to do with that date?" asked Jerry.

"We need to save it so that the application can access it when it needs to know today's date." I replied.

Avery still didn't like this, and made his opinion known by rolling his eyes; but he stayed mute.

"Right." Said Jerry. "Now let me show you how we do that."

He typed the following code.

```
public class UtilitiesTest extends TestCase {
  public void testDateGetsTodayWhenNoTestDateIsSpecified() throws Exception {
    Date now = new Date();
    Utilities.testDate = null;
    assertEquals(now, Utilities.getDate());
  }
}
```

Jerry looked over at Avery and asked: “Can you make that pass Avery?”

Avery sighed disdainfully as he took the keyboard. He said: “I suppose you’ll want me to use a *public* variable.” And then he typed the following without waiting for an answer:

```
public class Utilities {
  public static Date testDate = null;

  public static Date getDate() {
    return new Date();
  }
}
```

He clicked the test button and got a green bar.

“Great!” Said Jerry.

But Avery looked impatiently at Jerry and said:

“Yeah, but watch this.”

Avery started clicking the test button repeatedly. He got three or four green bars, and then the test failed with the message:

```
junit.framework.AssertionFailedError:
  expected:<Tue Feb 21 11:33:16 2000 (subjective)>
  but was: <Tue Feb 21 11:33:16 2000 (subjective)>
```

Before Jerry could react, Avery said: “You wrote the test wrong. Sometimes the two dates won’t be exactly the same. The difference is probably as small as a millisecond, but it’s enough to fail the test. I’ll fix that by writing the test a bit more intelligently.” And he changed the test as follows:

```
  public void testDateGetsTodayWhenNoTestDateIsSpecified() throws Exception {
    GregorianCalendar now = new GregorianCalendar();
    GregorianCalendar systemDate = new GregorianCalendar();
    Utilities.testDate = null;
    now.setGregorianChange(new Date());
    systemDate.setGregorianChange(Utilities.getDate());
    long difference = now.getTimeInMillis() - systemDate.getTimeInMillis();
    assertTrue(Math.abs(difference) <= 1);
  }
```

And then Avery turned towards Jerry and hit the test key repeatedly. He never once looked at the screen, but the test passed every time.

Jerry looked back at Avery coldly and said. “I concede the point, Avery. Thank you. However, that function is a bit cluttered now.” And Jerry took the keyboard back and made the following changes:

```
  public void testDateGetsTodayWhenNoTestDateIsSpecified() throws Exception {
    Utilities.testDate = null;
    assertTrue(DatesAreVeryClose(new Date(), Utilities.getDate()));
  }

  private boolean DatesAreVeryClose(Date date1, Date date2) {
    GregorianCalendar c1 = new GregorianCalendar();
    GregorianCalendar c2 = new GregorianCalendar();
    c1.setGregorianChange(date1);
    c2.setGregorianChange(date2);
    long differenceInMS = c1.getTimeInMillis() - c2.getTimeInMillis();
    return Math.abs(differenceInMS) <= 1;
  }
```

"OK, I think that expresses our intent a bit better." Jerry said as he watched the test repeatedly pass.

Avery just sniffed.

I was feeling a bit like I was in a war zone. I didn't know why Avery and Jerry were showing such animosity towards each other, but I was concerned that it was distracting them from the task at hand. So I grabbed the keyboard and said:

"OK, I think I know what the next test is." And I wrote:

```
public void testThatTestDateOverridesNormalDate() throws Exception {
  Date t0 = new GregorianCalendar(1959, 11, 5).getTime();
  Utilities.testDate = t0;
  assertEquals(t0, Utilities.getDate());
}
```

Jerry watched the test fail and said: "Right Alphonse. Now make it pass."

But Avery grabbed the keyboard and said: "I will." And he wrote the obvious code:

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

And as the test passed, he muttered: "And that's one hell of a lot of time wasted to get one simple and obvious line of code working."

"It wasn't *wasted*…." Jerry started. But then he stopped and shook his head. He took a deep breath and said: "OK, now we have to get the `DtrackContext` fixture to set the `testDate`. So he wrote the following test:

```
public class DTrackContextTest extends TestCase {
  public void testExecuteSetsTestDate() throws Exception {
    Date t0 = new GregorianCalendar(1959, 11, 5).getTime();
    DTrackContext c = new DTrackContext();
    c.todaysDate = t0;
    Utilities.testDate = null;
    c.execute();
    assertEquals(t0, Utilities.testDate);
  }
}
```

As I watched the test fail, I said: "OK Jerry, I see what you've done. You've created an instance of the fixture and jammed the `t0` date into `todaysDate` just like FitNesse[1] would. Then you expect t0 to somehow get set into the `Utilties.testDate` field. I suppose that it's the `execute` method of the fixture that does this?"

"Right. Remember that the FitNesse table we are working on looks like this:" and he pulled the table up on the screen."

DTrack Context
Today's date
2/21/2002

"As FitNesse processes this table it will jam the 2/21/2002 date into the `todaysDate` field of the `DTrackContext` fixture, and then it will call `execute()` on that fixture."

"OK, then I can make this test pass by doing this:" and I grabbed the keyboard and typed:

---

[1] www.fitnesse.org

```
public class DTrackContext extends ColumnFixture {
  public Date todaysDate;

  public void execute() throws Exception {
    Utilities.testDate = todaysDate;
  }
}
```

The tests passed, and I understood a bit more about how FitNesse worked. Then Jerry said: "You should run the acceptance test too." So I went to the `RegisterNormalSuit` page on the FitNesse server and clicked the test button. There was no change.

DTrack Context
Today's date
2/21/2002

"Good." Said Jerry. "We haven't broken anything."

"We haven't accomplished anything either." Said Avery.

Jerry glared at Avery, looked sadly over at me, then sighed and got out of his chair and walked over to Jean. The two of them started talking quietly. I looked at Avery and said: "Avery, what's wrong, why are you complaining so much?"

"He a dufus!" Said Avery. "He doesn't know *anything* about object oriented design, and all this testing nonsense is just make-work for morons. We've been working on this suit registration requirement for nearly four hours, and the only production code we've got to show for it is this:", and he pulled up the `Utilities` class on the screen.

```
public class Utilities {
  public static Date testDate = null;

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

"And even this class is nothing but a hack to allow testing. Four hours, three people, that's twelve man hours, and we've done jack cheese! I think this whole group is wacked out. I think Mr. C must be an idiot. How can he think he'll get this DTrack project done in two months when we squander hours and hours doing nothing but these stupid tests? If I were the captain I'd put Mr. C. out the airlock."

As Avery ranted, his voice grew louder and his eyes started to bug out. He didn't notice Jerry and Jean walking up behind him.

"That will be quite enough of that, young man!" Jean said sternly.

Avery snapped around, and his face went from anger to dismay. Jean was no longer the kindly grandmother we had come to know. She was something else entirely. It was clear from her tone of voice and the expression on her face that we were to remain silent and attentive, and that we were not to move.

"Speaking of the captain and his chief software craftsman that way is not acceptable behavior in this department. I won't tolerate it. Avery, yesterday you insulted Jason so badly he refused to work with you anymore. So I put you to work with Alphonse to see if any of his good manners might rub off on you. You two seemed to be getting along, and I dared hope that you had learned your lesson. But now you've been hostile to Jerry, and to the whole team, department, and even the *ship*. What shall we do with you?"

To be continued...

The source code for this article can be found at:
`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_29_DosageTrackingSystem.zip`

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

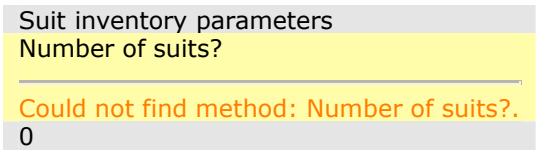
"So we need a fixture named `SuitInventoryParameters`, right?" I asked.

"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:



"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."

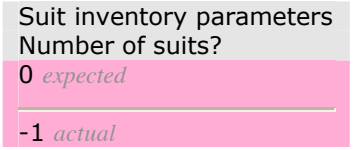
"That's right Alphonse, but this time it wants a method because there is a question mark following the name."

"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:



"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*	314159 *expected*	Outside Maintenance *expected*
null *actual*	0 *actual*	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`

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`

The Craftsman: 32
Dosage Tracking IX
Indubitably!

Robert C. Martin
16 November 2004

...Continued from last month.

In August of 1939, Hitler and Stalin signed a non-aggression pact. Though the pact between such ideological enemies was very unpopular with European and American Communists, the practical result was that by the end of 1940 the USSR had seized the eastern half of Poland and annexed the Baltic States; vastly expanding it's empire and creating a long shared border with Germany.

Over the years Stalin had deeply insinuated an espionage network within the highest levels of the United States government. One member of this network, Whittaker Chambers, was so outraged by the growing "friendliness" between Stalin and Hitler that he defected to the U.S. in 1938. Chambers provided the Roosevelt administration with details on dozens of other communist spies. One of these spies, Alger Hiss, was a top State Department official and a trusted friend of FDR. FDR refused to believe the accusations against Hiss and denounced Chambers instead – allowing Stalin's network to continue operating through 1940.

So Stalin knew all about Nimbus, including the Atomic Bomb research and Von Braun's rocketry successes. Stalin also knew he could not defend against the growing US threat alone. So in the early months of 1941 he negotiated a deep alliance with Hitler and the Axis powers, offering some of the Nimbus intelligence as part of the bargain.

The US had a monopoly on the great minds, but their secrets had been sold, and they had lost the element of surprise. Now a great Eurasian power was rapidly growing, and aligning against them.

---

21 Feb 2002, 1430

Jean didn't give us a minute to respond. "Avery and I have had a nice little chat, and I think we've got everything worked out. Don't you agree Avery? I know you're a fine young man, just a bit high spirited really. You wouldn't make such a fuss if you didn't *care*, would you? Of course you wouldn't. Anyway, Jerry and Alphonse, I'll leave Avery in your fine care and go on about the rest of my day. My goodness, but don't I have a lot of little details to take care of! It never ends, boys, it never ends." And she picked up her knitting basket and shambled off to chat with Carole.

Avery just stood there looking sheepish while Jerry and I put our hands in our pockets and looked at the floor. Finally Jerry said: "I'm glad you're out of there Avery. It's no fun in there – I know. Look, I have to go work with Jasper for awhile. Alphonse, can you and Avery finish up this test page?"

He was going to leave me alone with Avery after the woodshed! What would I say to him? "Sure, Jerry, we can work on it."

"Great, thanks. I'll be back in about an hour or so." And Jerry walked away.

"You were in there a long time." I said to Avery. "Was it bad? What did she say? Did she yell at you? I felt awful watching you in there."

"Yeah, well, Jerry's right about it not being fun. She didn't actually yell, but... Look, I probably shouldn't talk all that much about it. She made me realize a few things about myself that I don't like. I'm going to think about them for awhile. Perhaps I'll be able to tell you more later, but now I'd just like to get back to work; if you don't mind."

"Oh, yeah, sure. No problem. I'll show you what we've been doing."

"...and Alphonse, it means a lot that you were concerned about me. Thanks."

"Er, sure... uh, so here's the thing. Jerry and I have been working on the Register Normal Suit page. We just got the first failing table to pass. You and I should get the second failing table to pass."

I pointed to the page and Avery studied it.

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 expected   314159 expected     Outside Maintenance expected
null actual                  0 actual            null actual
```

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, so we want to get that Message sent to manufacturing table working?"

"Right." I said. We want to make sure that when we register the suit in the Suit Registration Request table, a message gets sent to manufacturing with the registration information."

OK, well it looks real easy to get that to pass. All we have to do is modify the `getLastMessageTo-Manufacturing()` method as follows:

```
public class Utilities {
  ...
  public static Object getLastMessageToManufacturing() {
    SuitRegistrationMessage message = new SuitRegistrationMessage();
    message.sender = "Outside Maintenance";
    message.id = "Suit Registration";
    message.argument = 314159;
    return message;
  }
  ...
}
```

Sure enough the table turned green.

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

"Yeah, that's what I thought too. But Jerry told me not to do that. He said that it was OK to do the simplest thing to get a *unit* test to pass; but not an acceptance test. He said that if you are tempted to do something too simple to get an acceptance test to pass, you should write some unit tests instead."

Avery looked confused. "What kind of unit test should we write?"

"Well, we want to make sure that we send a message to manufacturing when a suit is registered."

"Yeah, but that's what the acceptance test checks."

He had a point. "You have a point."

"Yeah, do we want to write a unit test that checks the exact same thing as the acceptance test?"

He had another point. "You have another point."

"So what do we do?" Avery was clearly trying to fight his incredulity. If he burst out again so soon after the woodshed, Jean might just leave him in there permanently.

"Look, writing a unit test is no big deal. If it happens to overlap with the acceptance test, then so be it. Let's just write the unit test and then show Jerry when he gets back."

"OK, whatever you say Alphonse, but I'll write it if you don't mind."

"Sure, go ahead."

So Avery grabbed the keyboard and wrote:

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

This failed for the following reason:

```
expected:<7734> but was:<314159>
```

"OK, now if we follow Jerry's rule and make this fail the simplest way possible we have to change `getLastMessageFromManufacturing` again."

```
  public static Object getLastMessageToManufacturing() {
    SuitRegistrationMessage message = new SuitRegistrationMessage();
    message.sender = "Outside Maintenance";
    message.id = "Suit Registration";
    message.argument = 7734;
    return message;
  }
```

I clicked the unit test and the acceptance test. "OK, now the unit test passes, but the acceptance test fails."

"Hmmm." said Avery.

"Yes!" I responded.

"My point exactly!" said Avery.

"Indeed!" I responded.

We looked at each other and laughed for a second.

"OK, I think I'm seeing Jerry's logic." Avery said. "To get both tests to pass, we have to do somethign intelligent. We can't just keep doing the simplest thing."

"Do you have something in mind?"

"Yeah, watch this." And Avery started typing again.

```
public class Utilities {
  ...
  private static Manufacturing manufacturing = new MockManufacturing();
  ...
  public static void registerSuit(int barCode) {
    manufacturing.registerSuit(barCode);
  }

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

```
package dtrack.external;

public interface Manufacturing {
  public void registerSuit(int barCode);
  public Object getLastMessage();
}
```

```
package dtrack.mocks;

import dtrack.external.Manufacturing;
import dtrack.messages.SuitRegistrationMessage;

public class MockManufacturing implements Manufacturing {
  private Object lastMessage;

  public void registerSuit(int barCode) {
    SuitRegistrationMessage msg = new SuitRegistrationMessage();
    msg.id = "Suit Registration";
```

```
    msg.sender = "Outside Maintenance";
    msg.argument = barCode;
    lastMessage = msg;
  }

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

"Wow." I said in admiration as I pushed the test buttons. The unit tests pass now, and so does the Message to manufacturing table."

"Yeah." Said Avery. But his brow was furrowed. "I don't like this. The Mock object shouldn't be building the message. That should be done by the real Manufacturing object. I also don't like that argument variable in the message, or the fact that this Utilities class is turning in to one big hodge podge. This code is a mess. It really needs to be cleaned up."

"Yeah, I agree. I said the same thing to Jerry just before you came back."

"Should we clean it up before he gets back here?"

I was a little worried about that. I didn't want to get Jerry mad by changing things that he wasn't ready to change. On the other hand, the code *was* a mess, and if we didn't clean it he'd be just as likely to be mad about that.

"OK." I said. "Let's see how much damage we can do before Jerry gets back here!"

"Indeed!"

"Indubitably!"

To be continued...

The source code for this article can be found at:

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

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`

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`

The Craftsman: 35
Dosage Tracking XII
What a day.

Robert C. Martin
22 February 2005

...Continued from last month.

Surprise, surprise!

In the early Summer of 1942, while britons were hopelessly battling Hitler's armies in the south and east of England, and the United States was frantically routing out the soviet spies from its upper echelons, the Empire of Japan struck at Pearl Harbor with a massive surprise attack.

Admiral Yamamoto had wanted to strike six months earlier, but the negotiations with Hitler and Stalin had caused delays. Even so, the U.S. was in such political turmoil that he felt confident he was about to eliminate the American threat and then run wild in the Pacific.

But that's not the way things happened. Not at all.

At 0740 on June 6th, Saburo Sakei was flying his zero south towards Oahu in a group of two dozen fighters and torpedo bombers. He kept a constant vigil over the sky and sea, looking left and right, up and down, hunting for planes and ships that could threaten him. When he was just 8 miles from Pearl he saw one of the zeros in his group suddenly explode without warning. He instinctively yanked on his stick to break out of formation, and a half second later saw another companion blow up.

Later, he told his captors that it was like being in a pop-corn popper. Plane after plane exploded before the pilots could even begin to take evasive action. Those few that managed to break out of formation fared little better. Sakei could just see the white vapor trails ripping northwards into the flight of zeros and following (!) *those who managed to break formation.*

Of course he didn't see the one that was following him.

---

21 Feb 2002, 1630

"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."

"That sounds like a plan." I replied.

"Indeed!"

I pulled up the `Utilities` class on the screen.

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

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

```
  }

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

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

We both stared at it for about 30 seconds. I said: "I suggest we get rid of that last method, `getSuitsInInventory`, I'd wager that it's quite easy to obliterate."

"I think I might agree with you." Avery invoked the 'find usages' command. The result showed that it was only used in one place: a fixture class named `SuitsInInventory`. He brought it up on the screen.

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

Avery studied the screen for a few seconds and then said: "I think obliteration is imminent." He clicked on the `getSuitsInInventory` function and invoked the 'inline' operation. The call to the function was automatically replaced with its implementation...

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

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

...and the function was automatically removed from `Utilities`.

We looked at each other with evil grins and we both said: "Obliterated!"

All the unit tests still passed, and the FitNesse page still passed. Nothing was broken.

"On to the next function." I said.

We both stared at the acceptMessageFromManufacturing

"Hmmm." Said Avery. "I presume you agree that obliteration is not indicated in this case."

"Indeed." I said. "I think relocation is a more appropriate option."

"Quite, quite. But where?"

This was a good question. This function actually had a bit of policy logic in it. This was the function that accepted the acknowledgement of transfer from manufacturing. This was the function that stored the suit in our inventory.

"What do you call something that finalizes a registration?" I asked.

"A secretary? A clerk? Uh, a ... Registraar!"

We both looked at each other with a grin and repeated: "A Registraar!"

I grabbed the keyboard and clicked on the method. I invoked the 'move' operation and typed `dtrack.policy.Registraar`. After confirming that I did indeed want to create the `policy` package

and the `Registraar` class, the method was automatically moved.

```
public class Registraar {
  public static void acceptMessageFromManufacturing(Object message) {
    SuitRegistrationAccepted suitAck = (SuitRegistrationAccepted) message;
    Suit acceptedSuit = new Suit(suitAck.argument, Utilities.getDate());
    Utilities.suitGateway.add(acceptedSuit);
  }
}
```

"Relocated!" we both shouted.

Now the `Utilities` class was looking very anemic.

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

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

"Death is coming to this class." I said with a sneer.

"Without a doubt. And I think I see our next objective. What would you say to removing that `suiteGateway` variable?"

"I think I'd enjoy it very much." I replied.

Avery grabbed the keyboard, clicked on the doomed variable, and invoked the 'move' operation. He selected the `SuiteGateway` class as the target. And the variable was automatically moved.

```
public interface SuitGateway {
  SuitGateway suitGateway = new InMemorySuitGateway();

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

"Interesting." I said. "The initialization moved too. I'm not at all certain that I like that. It seems to me that `SuitGateway` should not know about its derivatives."

"I'm afraid I must agree with you, Alphonse. The initialization needs to go somewhere else. I wonder where?"

This was taking some thought, and we were beginning to slip out of our formal banter.

"We need some kind of initialization functions somewhere... But who would call it?"

"Yeah, I don't know. Let's put a *todo* next to this one and ask Jerry about it when he comes back. I don't want this problem to derail us from obliterating the `Utilities` class."

"Agreed! Obliteration is our objective!"

And Avery annotated the initialization with a *todo* comment.

```
public interface SuitGateway {
  // todo move initialization somewhere else.
  SuitGateway suitGateway = new InMemorySuitGateway();

  public void add(Suit suit);
  int getNumberOfSuits();
  Suit[] getArrayOfSuits();
```

```
}
```

"OK, now let's find out who uses that variable." I said. I grabbed the keyboard and did a 'find usages' on the variable. There were 6 usages. They all looked about like this:

```
SuitGateway.suitGateway.getNumberOfSuits()
```

"I'd say that's somewhat redundant." I bantered.

"I'd agree. I propose we change the name of the variable." I clicked on the variable and invoked the 'rename' command, changing the variable's name to `instance`. Now all the usages looked about like this:

```
SuitGateway.instance.getNumberOfSuits()
```

Avery looked at the screen for a moment and said: "That's certainly an improvement. But I think we can do better." Avery used the 'rename' function to change the name of `SuitGateway` to `ISuitGateway`. Next he moved the `instance` variable to a new class named `SuitGateway`. Next he found all the usages of the `instance` variable and altered the first by eliminating the variable from the expression as shown below.

```
SuitGateway.getNumberOfSuits()
```

This caused the statement to turn red, showing that it would not compile. Avery clicked on the red lightbulb next to that line of code and selected the 'create method `getNumberOfSuits`' option. This automatically created the method in the `SuiteGateway` class. Finally, Avery modified that method to delegate through the `instance` variable. All the tests still passed.

Avery changed all the other usages of the `instance` variable in similar fashion. Now `SuitGateway` looked like this:

```
public class SuitGateway {
  // todo move initialization somewhere else.
  public static final ISuitGateway instance = new InMemorySuitGateway();

  public static int getNumberOfSuits() {
    return instance.getNumberOfSuits();
  }

  public static Object[] getArrayOfSuits() {
    return instance.getArrayOfSuits();
  }

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

"Avery, you are a genius!" I said. I was duly impressed.

"I know." He said with a smirk. And we both laughed.

"This is really pretty." I went on. The `SuitGateway` is really simple to use, and yet it's also polymorphic. We can put any derivative if `ISuitGateway` into that `instance` variable that we want."

"Yes, but we still have to figure out how to initialize it."

"Yeah. So what does our Utilities class look like now?"

I grabbed the keyboard and brought it up on the screen.

```
public class Utilities {
  public static Date testDate = null;
```

```
  public static MockManufacturing manufacturing = new MockManufacturing();

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

"Wow!" I said. "Nothing left but the date stuff, and that odd `MockManufacturing` variable."

"Yeah, and I'll bet you that we can get rid of that variable the same way we got rid of the `SuitGateway` variable."

"Yeah! A `Manufacturing` class, holding a reference to an `IManufacturing` object. Neat!"

Just then, Jerry stepped up. "Hay guys, it's time for dinner."

"Huh!" Avery and I looked at each other in surprise. Where had the time gone?

Jerry stared at the `Utilities` class. "Gee, it looks like you guys have been busy cleaning things up, eh? Good. Tell me all about it at dinner. Let's go."

So we left the lab for the day...and what a day it had been!

To be continued...

The source code for this article can be found at:

`www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_35_DosageTrackingSystem.zip`

The Craftsman: 36
Dosage Tracking XIII
The Color Purple

Robert C. Martin
20 March 2005

...Continued from last month.

The readout indicated **0.97**. *Turing was not prepared for* this. *His heart fell. His eyes glazed. And, not for the first time, suicide taunted him. He looked around at the huge device that so many of his associates had helped him to build. Flowers, Von Neumann, Eckert, all had played their part. But the machine was* his. *The Automatic Computing Engine was his brain-child. He designed it. He programmed it. And now... Now he must deal with what it told him.*

He felt as if the 30,000 vacuum tubes had betrayed him. As he stared at the number, **0.97**, *he realized just how much he had hoped that ACE would declare their salvation. But* **0.97** *was a virtual death sentence.*

Two months ago, the Japanese fleet in the Pacific had been routed, in no small part by the work that he and his fellows in the contingent had contributed to project Nimbus. How joyful those days had been! How they had buoyed his hopes. But ACE had computed the orbital elements of Clyde with more precision and speed than had ever before been possible. And the answer it gave left only three chances out of a hundred that humanity might survive. "We are lost," he murmured, "We are lost."

Yet even as Turing turned away from the number that spelled humanity's doom, an artificial sun blossomed over the Nevada desert.

---

21 Feb 2002, 1700

"Alphonse, can I ask you a question?"

"I think you just did."

Avery gave me a good-natured punch in the arm and said: "I'm being serious."

I ogled him and said: "You? Serious?" Then I notice the look on his face and got serious myself. "Sure, go ahead."

We were on our way from the lab to the general mess hall at the rim. I liked watching the starbow through the floor ports in the mess hall. Jerry, Jean, and the others were walking several yards ahead of us. They would probably go to the Journeyman's lounge for dinner. Jean favored the lower g. Avery kept his voice low. Apparently he didn't want the others to hear this conversation.

"How much do you think we accomplished today?"

"Avery! Not this again! You just got reamed through for this two hours ago."

"I'm not complaining, Alphonse. I'm just concerned. I know that this is the way Jean and Carole want to run the project, and I know that Jerry and Jasmine don't question it anymore. But I have my doubts. I'm willing to set them aside and work with the team, but I want to know your opinion. So how much do you think we got done today?"

"It seems to me that we finished the Register Suit requirement."

"Did we finish it? Do you think that we could really register a suit now?"

I thought about this for a second, as we stepped into the turbo lift. "Well, no. We don't have the bar code scanners hooked up to the system yet, and we don't have a real database or anything, and we really weren't sending messages over sockets. But we did get the logic of the requirement working."

We rode in silence as the lift hauled us 50 meters from the inner rim on 44 to the outer rim at 60. We could feel our weight increase, as we unconsciously braced ourselves against the mild anti-spinward coriolis force.

Avery didn't speak again until we were half way between the lift and the mess hall. When he did speak he rushed his words together as though speaking them had released a pent-up frustration. "I think we could have written a lot more code in one day. We hardly wrote anything code at all. That one little requirement should have been coded in less than an hour. We should have gotten five or ten requirements like that coded today."

Avery's face was reddening, and his eyes were starting to bug out.

"I agree with you Avery." I said. "We *could* have gotten a lot more *coded* today. But would it have been *working*? Would it have been *done*? That Suit Registration story is *done*. It's done in a way that I don't remember anything from school ever being done before. It's coded, it executes, its been tested against unit tests and acceptance tests. It's *done*!"

We turned into the mess hall, picked out some trays, and stood in the cafeteria line. I prepared two hot dogs loaded with mustard, onions, sauerkraut, and hot-peppers! My mouth started to water. Yum! Avery absently made himself a peanut butter and jelly sandwich. We both got a tall glass of lemonade. Then we seated ourselves at a table that had a clear view through one of the floor ports. The starbow shimmered there as always.

Avery took a long pull of his lemonade and said: "OK, you are right, that tiny bit of the Register Suit feature is done. But don't you think we could have done a lot more today?"

I took a huge bite out of a hotdog. Mustard dribbled out of the bun, and got all over my face. A hot pepper burst in my mouth mixing with the powerful bite of the mustard. Heaven! "Avery, perhaps you... are remembering what it was like to write... projects for school."

"Yeah, sort of. I mean we got a *lot* done in a short period of time!"

I wiped the mustard from my mouth with the back of my hand. "Perhaps it seemed that way to you. Perhaps your classes were more productive than mine. What I remember was frantic bursts of coding followed by long droughts of debugging, only to deliver a project that was buggy, and had taken lots of long midnight hours to get working. Sure the coding went fast. It was everything else that took so long."

"Well, you make a good point, but..." There was a dab of grape jelly on his cheek.

"Avery," I said around another dribbling bite of hotdog, "how much debugging did... we do today? We got Carole's acceptance... test passing without any debugging at all! It's done Avery! Tomorrow we're going to start working on the next requirement. The day after that we'll work on the next. And Friday we'll work on yet another. If we keep up this pace, just you and I will get four stories done. Jerry and Jasper will get three of their own done. We'll have seven stories done! And the week after that we should have ten more done. Gosh, if we knew how many stories Carole was going to write, we could just divide by ten to figure out how many weeks this project is going to take!"

Avery was chewing a big bite of his sandwich, and trying to talk at the same time. The result wasn't pretty. "Yeah.... OK.... but.... that assumes that all the stories are... the same size. Some are... going to be a lot harder... than that."

I picked up my second hot dog. Mustard laden sauerkraut splayed around my fingers. "Yeah..." I took a bite. "you're probably right.... But.... That's OK because... we can estimate which ones are.... bigger and.... so.... we'll still be able to tell... how many stories we can... get done in a week, and... how many weeks until we can deliver the... project."

Avery's sandwich was too full of jelly. A big glob dripped out into his hands as he took his next bite. "You could be right... I mean, maybe working this way....is more predictable. I suppose that Carole... would find that... comforting. But... What about QA?... Won't that take... a long time?"

I crammed the soakstall – the butt end of my hotdog – into my mouth, licking the mustard and pepper juices from my fingers while chewing and talking. "QA?... Do you think... we'll need much?... I mean

with all... the testing... we are doing. Don't... you think...

Suddenly I couldn't breathe. The soakstall had gone down the wrong way. I tried to cough it out, but it wouldn't come. Avery saw my distress and started whacking me on the back. Two whacks and the soakstall came flying out of my mouth. I took a deep grateful breath, and then apologized to the people at the next table.

"You'd better be more careful when you eat those." said Avery.

"Yeah, I guess I should. C'mon, I'm still hungry. Let's see if there's any pizza." I started walking towards the cafeteria line again.

Avery followed me. "Hay, Alphonse?"

"Yeah?"

"Did you know that you have two purple handprints on the back of your shirt?"

To be continued...

The Craftsman: 37
Dosage Tracking XIV
Handling Rejection

Robert C. Martin
27 April, 2005

...Continued from last month.

Through the last quarter of 1942 FDR's advisors were pressing him to declare war upon the Axis. The successful detonation of an Atomic bomb in August, and the increasing supply of those terrible weapons and the rockets to deliver them, was emboldening the hawks. And, horrible though it was, it made sense. It seemed unlikely that the U.S. would be unable to hold the technological advantage for long. Eliminating the virulence of fascism might be possible now, but the chance could quickly disappear leaving the U.S. to face a huge and fearsome opponent.

The contingent had waited to tell the President about Turing's results. They had checked, and double checked the results. They had improved the computers, the telescopes, the programs, and the observations. But the results would not be denied. A 22km rock named Clyde was almost certainly going to slam into the Pacific at 53km per second on April 29th, at 0943GMT.

On November 12th, 1942 as FDR read the report from the contingent, the certainty of destruction seemed to leave the pages and clamp down upon his skull like an iron fist. He called out to his secretary: "Grace, I have a terrific headache!" Grace Tully found him collapsed over his desk, never to regain consciousness again.

---

22 Feb 2002, 0800

I walked into the lab, just as people were gathering for the morning stand-up meeting. Avery, I, Jasper, Jerry, Carole, and Jean stood together in a circle. Each of us answered three questions: What did you do yesterday? What do you plan to do today? What's in your way?

When my turn came I said: "Avery and I got the `RegisterNormalSuit` acceptance test to pass. I don't know what I'm doing today, and nothing but that ignorance is in my way." I saw Carole role her eyes at that remark, and Jerry gave a little smirk.

Then it was Avery's turn, and he simply said: "My report is the same as Alphonse's."

After the standup Jean came over and said: "Boys, I think I can solve your problems. Avery, dear, why don't you work with Jerry? He already has his next acceptance test. Alphonse, I'd like you to work with Jasper today. He'll be putting together the acceptance test for manufacturing rejection. I'm sure you and Jasper will get along wonderfully; you are both such fine young men. " She gave us a warm smile and then said "Off with you! Shoo!"

I caught Avery's eye, and gave him a regretful wave, and then headed over to where Jasper was working. He was staring intently at his screen and didn't seem to know I was there. So I said: "Hello Jasper, I guess we're working together today."

Jasper jerked around with a big grin on his face and said "Alphonse! Yeah, great. Hay, should I call

you Al, or Fonse? I kind of like Fonse. Whaddya say?"

His toothy grin, and the sparkle in his eyes, put me off guard. I was about to say that "Fonse" is the kind of nickname that sticks, and that I wasn't sure if I really wanted it to stick; but he cut me off and said: "Great! Now let's get busy on this new test."

I sighed and sat down. "Jean said something about a rejection from Manufacturing?"

"Yeah, that's right. Here look at the story card." He handed me the index card from Carole's planning meeting yesterday morning[1].

Register new suit.

- Bar Code Patch X(6)
- Register new suit, screen function.
- Send confirmation to mfg.
    - Reject registration on denial.
    - 10s time out & reject
- If already reg'd
    - Reject reg & don't send conf to prod.
- Sched for inspection.

"Oh, yeah." I said. "I remember now. Carole told us that if someone tries to register a suit that didn't come from manufacturing, we should reject the registration."

"Right." Said Jasper. "We don't want someone trying to register some old suit they found in a locker somewhere. Eh?"

"Yeah. OK, so we need a new acceptance test for this case, don't we?"

"Right you are, Fonse, I was just working on that when you came up here." He pointed to his screen. "I just created a new page named `SuitRegistrationRejectedByManufacturing`. I took the `RegisterNormalSuit` page that you guys got working yesterday, and pasted it into this new page."

I examined the page, and sure enough it was a perfect copy of the `RegisterNormalSuit` page. "So I guess you want to alter it to reflect that manufacturing rejects the confirmation request?"

"Right again, Fonse. Want to take a crack at it?"

"Uh, sure." I wasn't sure if I liked his over-friendly demeanor. He didn't seem to mean anything by it, but I thought it could get annoying after awhile. I edited the page, changing a few comments and titles. The meat of the difference was to change the message sent my manufacturing to a rejection, and to then assert that the suit did not get placed into inventory. The result, complete with test results, looked like this:

[1] http://www.sdmagazine.com/documents/s=7764/sdm0407h/

Manufacturing rejects a 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 attempt to 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 rejects the confirmation.

Message received from manufacturing

message id	message argument	message sender	message recipient
Suit Registration Rejected	314159	Manufacturing	Outside Maintenance

The rejected suit should not be in the registered inventory.

Suits in inventory

bar code?	next inspection date?
314159 surplus	Thu Feb 21 00:00:00 CST 2002

"Nice work!" said Jasper. "That's exactly right. Now, can you make this pass?"

"I think I can." I said. I opened up the fixture that handled the "Message received from manufacturing" table. It looked like this:

```
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);
```

```
    Registrar.acceptMessageFromManufacturing(message);
  }
}
```

It just passed the table data along to the `acceptMessageFromManufacturing` method of the `Registrar` class. That method looked like this:

```
public class Registrar {
  public static void acceptMessageFromManufacturing(Object message) {
    SuitRegistrationAccepted suitAck = (SuitRegistrationAccepted) message;
    Suit acceptedSuit = new Suit(suitAck.argument, Utilities.getDate());
    SuitGateway.add(acceptedSuit);
  }
}
```

I made the following simple change:

```
public class Registrar {
  public static void acceptMessageFromManufacturing(Object message) {
    SuitRegistrationAccepted suitAck = (SuitRegistrationAccepted) message;
    if (suitAck.id.equals("Suit Registration Accepted")) {
      Suit acceptedSuit = new Suit(suitAck.argument, Utilities.getDate());
      SuitGateway.add(acceptedSuit);
    }
  }
}
```

When I ran the test, it passed.

"Nicely done, Fonce; but you didn't write a unit test!"

"Do you really think one is necessary for just this `if` statement?

"How hard would it be to write?"

I shook my head and just typed. It was a simple test to write.

```
public class RegistrarTest extends TestCase {
  public void testAcceptRegistration() throws Exception {
    final String acceptId = "Suit Registration Accepted";
    SuitRegistrationAccepted suitAck =
      new SuitRegistrationAccepted(acceptId, 9999, "me", "you");
    Registrar.acceptMessageFromManufacturing(suitAck);
    Suit[] suits = (Suit[])SuitGateway.getArrayOfSuits();
    assertEquals(1, suits.length);
    assertEquals(9999, suits[0].barCode());
  }
}
```

The test passed on its first try. I felt a little smug. So as I looked over to Jasper and raised an eyebrow. His supercilious grin framed by his sugarbowl haircut made him look clueless.

"You didn't write the negative case." He said, still grinning widely.

I thought to myself: "This must be how Avery feels." I suppressed the desire to roll my eyes, and simply kept typing.

```
  public void testRejectRegistration() throws Exception {
    final String rejectId = "Suit Registration Rejected";
    SuitRegistrationAccepted suitNak =
      new SuitRegistrationAccepted(rejectId, 9999, "me", "you");
    Registrar.acceptMessageFromManufacturing(suitNak);
    Suit[] suits = (Suit[])SuitGateway.getArrayOfSuits();
```

```
    assertEquals(0, suits.length);
  }
```

This test also worked the first time. So I raised my eyebrow again.

"Hey Fonse, the name of that class isn't quite right, is it?"

I did a silent little curse. I had been hoping he wouldn't notice what I had notice while typing this test. The name `SuitRegistrationAccepted` really wasn't the right name for this class anymore. I hated to admit it, but this is something that the unit test forced me to see, that the acceptance test completely hid from me. So I changed the name to `SuitRegistrationAcceptanceMessage`.

Jasper tilted his head as though thinking about something, and then cheerily said: "Fonse, do you think those message id's should really be long strings? Do you think those strings ought to be scattered around the code? I think that's kind of ugly don't you?"

Drat, he did it again. That was just what I was thinking. Strings like "Suit Registration Accepted" and "Suit Registration Rejected" probably weren't good values to use as message ids. Moreover, they probably shouldn't be scattered around the code. "Yeah, I agree. What do you think we should do about it?"

"I think we should get all the tests to pass, and then refactor those id strings out of the code altogether. Perhaps we can make them independent classes."

"Uh, wait. The tests *do* all pass."

"Have you run them all since that last change you made Fonse?"

"Yeah, I..." Actually I had only run the `testRejectRegistration` test. I hadn't run any of our other unit tests. So I turned back to the screen and pushed the button the ran *all* the existing unit tests in the DTrack system. To my horror, `testRejectRegistration` failed! "Wait! That test just passed!"

"Yeah, I think we've got a database cleanup problem." Smiled Jasper. "Suits are being placed into inventory, and not removed at the start of each test. Try running all the acceptance tests. I'll bet they fail too."

Sure enough, as I pushed the *suite* button on the top level of our acceptance tests, the `RegisterNormalSuit` page passed, but the `SuitRegistrationRejectedByManufacturing` page – the page I had just gotten working ten minutes ago, failed.

Jasper smacked his hands together and started rubbing them. "OK Fonsie, my young apprentice, let's clean this all up."

The code for this article can be located at:

http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_37_DosageTrackingSystem.zip

To be continued...

The Craftsman: 38
Dosage Tracking XV
Test Independence

Robert C. Martin
11 May 2005

...Continued from last month.

President Henry Wallace was sworn into office on the 13th of November, 1942. Wallace was a scientist and a visionary. Before he took office he had dreamed of conquering fascism, and creating a just and lasting peace throughout the world. He felt the Earth was on the verge of "The Century of the Common Man".

A weaker man might have given up hope upon learning that "the common man" had less than twenty years to live. But Wallace was not cowed. He had read the Contingent's reports on the use of Atomic Energy to deflect, or at least escape, Clyde. He determined that the resources of the entire world must be brought to bear on this problem, and he was determined to lead the U.S. in making that happen.

Only one thing was in his way... The Eastern Hemisphere.

---

22 Feb 2002, 0830

"The first thing to do, my dear Fonse, is to get all these tests to pass. Do you know why they are failing?" Jasper's toothy grin was both congenial and grating.

"I think so." I said, trying not to appear befuddled. "Apparently the first test to run changes the database in a way that the next test doesn't expect."

"You got it, buddy! I'm pretty sure that's the case. So we need to clear the database between each test case."

This sounded strange to me, like extra work. "Why don't we just change the second test to expect what the first test leaves behind? That way we don't have to clean up. Even better, it means that the tests can build on each other. Each test leaves the database all set up for the next test." I was proud of myself for thinking this through.

Jasper waggled his finger at me. His thick blond hair, cut of at his ears, jiggled as he shook his head. "Nosirree, Fonse ol' pal. We don't let tests depend on each other. We want to be able to run any test at any time. We don't want to have to run them in sequence."

In the back of my mind I could see Avery rolling his eyes at this thought. I suppressed that gesture, and simply asked: "Why is that? Wouldn't it be much easier to run the tests in sequence?"

"It might be a little bit easier *right now*", Jasper said with sugary emphasis on the last two words, "but it would make it very hard to diagnose problems later on." Jasper put his arm around my shoulder as he continued his syrupy lecture: "Think about what would happen, Fonse, if one of the tests in your sequence failed? *All* the downstream tests would fail too, wouldn't they?"

I politely disengaged from his arm and said: "Well, sure, but so what? You'd know what test failed."

A glimmer of regret passed over Jasper's face as I backed away from him. "True, but you wouldn't know if any of the downstream tests would have passed. To find out you'll have to get that first failing test to pass. If other, later, tests fail, you'll have to get them all working one by one."

Jasper paused for a second and then said: “It’d be like compiling a C++ program. Have you ever done that?”

I shook my head.

“Never mind. You can see that it would be inconvenient. It’s better for each test to fail for it’s own reasons, than because a previous test failed. Also, it’s very convenient to be able to run any test at any time. If we put them in a sequence, then we wouldn’t be able run individual tests.”

I sighed. This actually made some sense. I wondered why I didn’t like hearing sense from Jasper. Perhaps it was because the more he talked, the wider his grin got, making it look like his head was split in half.

“OK.” I said, “So we need to clear the database in front of each test case.”

Jasper winked and said: “Attaboy, Fonse!”

This was torture. I didn’t know how much longer I could keep cool. So I faced the keyboard and ran all the unit tests again. Two unit test files were failing: `RegistrarTest`, and `UtilitiesTest`. I opened each, and looked them over. Sure enough, each of the failing test cases made use of the `SuitGateway`, our front end to the database. I opened `SuitGateway` and saw this:

```
public class SuitGateway {
  // todo move initialization somewhere else.
  public static final ISuitGateway instance = new InMemorySuitGateway();

  public static int getNumberOfSuits() {
    return instance.getNumberOfSuits();
  }

  public static Object[] getArrayOfSuits() {
    return instance.getArrayOfSuits();
  }

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

I thought to myself that perhaps it was time to follow the advice in that `todo` comment. So I changed the initialization of the `instance` variable to `null`, and ran all the unit tests. Of course I got lots of null pointer exceptions, but only in `RegistrarTest` and `UtilitiesTest`. So I opened `RegistrarTest` and modified it as follows:

```
public class RegistrarTest extends TestCase {
  protected void setUp() throws Exception {
    SuitGateway.instance = new InMemorySuitGateway();
  }

  ...
}
```

The `setUp` function is called in front of every test function, so now each test function will start with an empty database. I pushed the test button and *all* the unit tests now passed, even the ones in `UtilitiesTest`!

“That’s strange.” I said.

Jasper grinned at me like he’d just heard a funny joke. “C’mon, Fonse, use your head there buddy. The last test case in `RegistrarTest` just left the database empty.”

I could feel the heat rise in my face. I hoped the flush wasn’t visible. I took a deep breath and said: ”I see what you mean. But I still need to clear the database in `UtilitiesTest`, just to make sure.”

“Oh, absolutely Fonse! I won’t argue with that!”

His Cheshire cat smile hung in space behind my closed eyes. When it faded I made the same change to `UtilitiesTest`, and the unit tests all continued to pass.

Jasper gave me a gentle elbow in the ribs and said: "Nice sailing captain, now let's fix those acceptance tests."

Something cold began to rise in me, replacing my flush with ice, or perhaps steel. I took another deep breath, and realized it was the last one I was willing to take. I ran the acceptance tests. Sure enough they all failed for null pointer violations.

The first fixture in both acceptance tests was `DTrackContext`. This fixture was made for initialization. So I made the following changes.

```
public class DTrackContext extends ColumnFixture {
  public Date todaysDate;

  public void execute() throws Exception {
    Utilities.testDate = todaysDate;
    SuitGateway.instance = new InMemorySuitGateway();
  }
}
```

This made sure that each of the two test pages now started with a cleared database. I pushed the suite button, and saw green; all the acceptance tests pass.

I knew it was coming without even looking. I could feel his enthusiastic congratulations before they left his mouth. I waited for it with ice water in my veins, knowing what I had to do.

He patted me on the head. "Nice work Fonse!"

He patted me on the head! I closed my eyes. I was done. There would be *no more* of this. I tightened my stomach and visceral muscles and turned towards him.

"Jasper, my name is *Alphonse*. Please don't call me Fonse anymore. And *please* stop being so patronizing. I'm not going to be able to work with you if you keep treating me like a pet. I know you are more experienced than I am, but that doesn't make me a low grade moron who needs constant cuddling. Now, if you'll excuse me, I'm going to the washroom."

Had I really just said all that? Had I really been so calm and articulate? As I approached the washroom I could feel the ice-water drain away to be replaced with fear and embarrassment. I stood in the washroom and waited for the shakes to stop. Jean was going to be furious with me. She was going to take me to the woodshed. But I couldn't go on like that. I had to stop him.

Didn't I?

The code for this article can be located at:

http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_38_DosageTrackingSystem.zip

To be continued...

The Craftsman: 39
Dosage Tracking XVI
Test Refactoring

Robert C. Martin
7 June 2005

...Continued from last month.

By hook, and by crook, President Wallace poured *money into project Nimbus. He directed the Contingent to develop technologies pursuant to: 1. Saving the human race from Clyde. 2. Defending this effort against the Axis powers.*

The latter of these two imperatives enjoyed more short term success than the former. Technologies for weapons systems were created and deployed in absurdly short time frames. A now famous quote by Edward Teller summarized the period: "When you fight for a desperate cause and have good reasons to fight, you usually win."

By August of 1943, Von Braun's rockets were launching cameras into orbit to spy on Europe and Asia. Canisters of film were ejected from these satellites and recovered by aircraft that snagged them from the sky as they fell on their parachutes.

That's how the U.S. learned that the Axis was planning to invade Mexico.

---

22 Feb 2002, 0830

When I returned from the washroom, Jasper was still sitting at the workstation waiting for me. His smile had been replaced by a hang-dog grimace.

"Alphonse, look." He said with an "aw shucks" tone. "I'm sorry; I guess I was trying too hard. The truth is that I just recently made Journeyman status, and I've never had an apprentice before. Can we start over?"

I had been on the verge of apologizing myself, so I was glad he beat me to it. "Sure, Jasper, we can start over. Just ease off the attitude a bit, OK?"

"I'll try, Alphonse. And if I forget, don't hesitate to remind me. Deal?"

"Deal." I wasn't sure if this would last, but we had a lot of work to get done, and we didn't need personality problems getting in the way. "So where are we?"

Jasper turned to face the screen, and said: "We just got the `SuitRegistrationRejectedBy-Manufacturing` test running. So the next test we should work on is the rejection of a duplicate suit."

"You mean if someone tries to register a suite that's already registered, we should reject it? That makes sense."

"Right. So let me show you this neat new fixture that Rick Mugridge wrote. It's called `DoFixture`, and it really helps writing acceptance tests like this one."

"Rick who?"

"Rick Mugridge. He's an astrogator by training, but has a passion for programming. So lets start by creating the new test page."

Jasper opened the `RegisterSuit` page and made the following change:

```
|^RegisterNormalSuit|''The Happy Path.''|
|^SuitRegistrationRejectedByManufacturing|''Manufacturing rejects the
registration.''|
|^RejectDuplicateRegistration|''You can't register a suit that's already
registered.''|
```

Then he clicked on the `RejectDuplicateRegistration` link and copied the following onto the new page:

```
!path C:\DosageTrackingSystem\classes

!3 DTrack rejects a duplicate suit registration.

!|Import|
|dtrack.fixtures|

''We assume that today is 2/21/2002.''
!|DTrack Context|
|Today's date|
|2/21/2002|
```

I pointed to the first line on the screen and said: "All three of our test pages has that `!path` line. Is there a way to put it in one place?"

Jasper winked and said: "Sure beans, Fon...Alphonse. We can just move it to the parent page." So he made the following change to the `RegisterSuit` page, and removed the `!path` line from the three subpages.

```
!path C:\DosageTrackingSystem\classes

|^RegisterNormalSuit|''The Happy Path.''|
|^SuitRegistrationRejectedByManufacturing|''Manufacturing rejects the
registration.''|
|^RejectDuplicateRegistration|''You can't register a suit that's already
registered.''|
```

Then Jasper typed the following table into the `RejectDuplicateRegistration` page:

```
|Do Fixture|
|start|dtrack.fixtures.DTrackFixture|
|set suit|314159|as registered|
|check|register suit|314159|false|
|check|was a message sent to manufacturing|false|
|check|count of registered suits is|1|
|check|error message|Suit 314159 already registered.|true|
```

"OK, that's different." I said. "Let's see if I understand it. First you force suit 314159 into the registration database. Next you try to register 314159, and expect it to fail because it's a duplicate. You make sure that no message was sent to manufacturing, that the number of suits in the database is still one, and that an appropriate error message was created."

Jasper's smile was back. "Right you are Alphonse old sport!"

Maybe Jasper just couldn't turn his attitude off. I ignored it and asked: "This is pretty simple to understand, but what's that `start` line at the top?"

Jasper raised his eyebrows twice and said: "Good catch! That names the class, `DTrackFixture`, that has all the methods that this fixture will call."

I didn't understand this and must have looked puzzled because Jasper said: "I'll show you. Go ahead and click the test button." I obeyed and saw:

```
Could not find fixture: DoFixture.
```

"Right," said Jasper. "You've got to import the package `fitLibrary`."

So I added `fitLibrary` to the `Import` table as follows:

```
!|Import|
|dtrack.fixtures|
|fitlibrary|
```

Now when I hit the test button I got a whole bunch of error messages. The first was:

```
Unknown class: DTrackFixture
```

Before Jasper could comment I said: "OK, so I need to create a class named `dtrack.fixtures.DTrackFixture`, right?"

"Correctomundo, Al! Uh, Alphonse."

He was clearly making an effort. I was surprised that it was so hard for him, but I made no comment. I just typed:

```
package dtrack.fixtures;

public class DTrackFixture {
}
```

That made the first error message go away. The next message was:

```
Unknown: "setSuitAsRegistered" with 1 argument
```

Again, before Jasper could comment I said: "OK, this must be a method of `DTrackFixture`, right?" I could see him starting to wind up to deliver another syrupy answer so I held up my hands and gave him a meaningful look. He paused, puzzled; then gave me a sardonic smile, and simply said: "Right." "Progress!" I thought to myself. I typed the method:

```
public class DTrackFixture {
  public void setSuitAsRegistered(int barcode) {

  }
}
```

That made the error message go away. Likewise I was able to make all the other error messages go away one by one. Eventually the fixture looked like this:

```
public class DTrackFixture {
  public void setSuitAsRegistered(int barcode) {

  }

  public boolean registerSuit(int barcode) {
    return true;
  }

  public boolean wasAMessageSentToManufacturing() {
    return true;
  }
```

```
  public int countOfRegisteredSuitsIs() {
    return 0;
  }

  public boolean errorMessage(String message) {
    return false;
  }
}
```

Now when I hit the test button, there were no more error messages. Instead, I had red cells in the table that looked like this:

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 expected 0 actual	
check	error message	Suit 314159 already registered.	true expected false actual

Jasper shook his head as he looked at the screen. "You figured that out pretty fast, Alphonse. Do you think you can wire the fixture up to the application?"

There wasn't even a *hint* of condescension in his demeanor. I think he was actually impressed instead of pretending to be. "Sure, I think so."

The first method of the fixture was easy. All I had to do was jam the specified suit into the database:

```
public void setSuitAsRegistered(int barcode) {
  SuitGateway.add(new Suit(barcode, Utilities.getDate()));
}
```

Then I changed `DTrackFixture.registerSuit` to call `Manufacturing.registerSuit`.

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

The `wasAMessageSentOtManufacturing` method simply checked to see if a message was sent.

```
public boolean wasAMessageSentToManufacturing() {
  return Utilities.manufacturing.getLastMessage() != null;
}
```

The `countOfRegisteredSuitsIs` method was also trivial.

```
public int countOfRegisteredSuitsIs() {
  return SuitGateway.getNumberOfSuits();
}
```

But I didn't know what to make of the `errorMessage` method. So I left it unchanged. When I hit the

test button, I saw:

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

I looked expectantly at Jasper. He said: "Hot ziggedy dog, Alphonse! -- er – Sorry, I mean, uh, good. But what about the `errorMessage` method?"

"I don't know exactly what to do for this method. It looks like you are trying to see if an error message was printed. But I don't know how to check that."

"Oh SURE you do, Alphonse! -- er – I mean, I think we want to make a mock object that represents the screen, and remembers the messages that were sent to it. The `errorMessage` method then simply asks if a particular error message has been sent to the screen."

Of the two Jaspers, I liked the second one better. His idea made sense. So I said: "OK, so we could create an `IConsole` interface that has a method like `display(String message)`. And then we can create a dummy `Console` that implements that interface and saves any message sent there. That sounds easy." So, bit by bit I wrote the following test, and the code that made it pass:

```
public class MockConsoleTest extends TestCase {
  private MockConsole c;

  protected void setUp() throws Exception {
    c = new MockConsole();
  }

  public void testEmptyConsole() throws Exception {
    assertEquals(0, c.numberOfMessages());
  }

  public void testOneMessage() throws Exception {
    c.display("message");
    assertEquals(1, c.numberOfMessages());
    assertTrue(c.hasMessage("message"));
    assertFalse(c.hasMessage("noMessage"));
  }

  public void testTwoMessages() throws Exception {
    c.display("messageOne");
    c.display("messageTwo");
    assertEquals(2, c.numberOfMessages());
    assertTrue(c.hasMessage("messageOne"));
    assertTrue(c.hasMessage("messageTwo"));
    assertFalse(c.hasMessage("no message"));
  }
}
```

```
public interface IConsole {
  public void display(String message);
}
```

```
public class MockConsole implements IConsole  {
  HashSet messages = new HashSet();
  public void display(String message) {
    messages.add(message);
  }

  public int numberOfMessages() {
    return messages.size();
  }

  public boolean hasMessage(String s) {
    return messages.contains(s);
  }
}
```

"OK, that's good." said Jasper. "Now let's see if we can get this acceptance test to pass."

The code for this article can be located at:
http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_39_DosageTrackingSystem.zip

To be continued...

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...

The Craftsman: 41
Dosage Tracking XVIII
Yeah, Sorta.

Robert C. Martin
23 August 2005

...Continued from last month.

In April of 1944, the three leaders of the Axis, Hitler, Stalin, and Tojo (Mussolini's status had declined of late) traveled to the Canary Islands to witness the launch of "Operation Underbelly", the Mexican Invasion. They were expecting to make very short work of Mexico and then to plow through the breadbasket of the United States.

At exactly *8pm on the eve of the launch, the sky directly above the Canary Islands suddenly blazed forth with actinic fury. For a fraction of a second the landscape was bathed in noon-day light. Anyone who happened to be looking straight up at the time was dazzled as though they had looked at the sun. There was no sound, no thunder, indeed, the silence was eerie. A second or so after the flash the sky took on a strange green glow that faded into a blood-red aura spreading across the horizon for several minutes.*

In the hotel where the three Axis leaders were enjoying an evening meal, the electrical power failed, and telephone and radio communications were disrupted for a few minutes. When communications were restored the leaders learned that similar events had occurred directly above Berlin, Tokyo, and Moscow.

---

22 Feb 2002, 1000

"Damn, when did we break that unit test?" I was pretty embarrassed that I had allowed the tests fail. I made a mental note to run the tests more frequently.

Jasper's big toothy grin and his condescending demeanor continued to grate on me. "Why don't you check the error, Alphonse?"

The error said:

```
Expected:Suit Registration
Actual  :dtrack.messages.SuitRegistrationApprovalRequest
```

I clicked on the error and it took me to the failing test. (I've marked the failing line with an arrow.)

```
public class ManufacturingTest extends TestCase {
  public void testRegisterSuitSendsMessageToMfg() throws Exception {
    MockManufacturing mfg = new MockManufacturing();
    mfg.requestApprovalForRegistration(7734);
```

```
    SuitRegistrationApprovalRequest message =
              (SuitRegistrationApprovalRequest) mfg.getLastMessage();
 -> assertEquals("Suit Registration", message.id);
    assertEquals("Outside Maintenance", message.sender);
    assertEquals(7734, message.argument);
  }
}
```

"Oh, OK." I said. "When I changed the ID of the message, I forgot to change the test. That's easy to fix." So I changed the test as follows.

```
    assertEquals("dtrack.messages.SuitRegistrationApprovalRequest",
                 message.id);
```

I ran the unit tests and they all passed.

But Jasper didn't like it, and he said so: "I don't like it Alphonse."

"Why not, Jasper? It works."

"Sure it works, Alphonse, but if the message id changes again, the test will break again."

"What do you suggest we do about it, Jasper?"

"How about this?" And with that, Jasper took the keyboard and changed the test to this:

```
    assertEquals(SuitRegistrationApprovalRequest.ID, message.id);
```

This didn't compile because the `ID` field wasn't public. Jasper made than simple change than then ran all the tests. They all passed.

Then Jasper passed the keyboard back to me, locked my gaze with those sparking eyes of his, and gave me a quick double eyebrow raise. He clearly thought that his change was very clever. I was beginning to wonder just what it took to become a Journeyman around here. If Jasper was any indication, Mr. C's standards must be pretty low.

I looked back at the screen and tried to focus on my job. "OK, I see your point. It's better not to embed constants in the code if you can avoid it. Now, let's see why so many of the acceptance tests are failing."

"I'll bet it's for the same reason." Jasper quipped.

He was probably right. The acceptance tests probably also mentioned the message ID by name.

"Yeah." I said, and brought up the first failing acceptance test.

Message sent to manufacturing message id?	message argument?	message sender?
Suit Registration expected dtrack.messages.SuitRegistrationApprovalRequest actual	314159	Outside Maintenance

"You see!" Cried Jasper. "It's the same issue."

"Yes, I see Jasper. You are right. It's the same issue." I gave a deep sigh and shook my head.

Jasper noticed my mood and hung his head. "I'm sorry Alphonse, I'm doing it again aren't I."

"Yeah, sorta." I said, without looking at him.

"OK, look, I'll try to back off. I guess I'm just easily excited."

I gave a snort, looked over at him with a grin and said: "Yeah, sorta!".

He smiled back, still embarrassed, and then we both turned back to the code.

"OK, Jasper, I think we need to apply your solution again. Instead of just changing the string in the test table, we need to see if the ID in the message equals the contents of the static ID variable in the class."

"Yeah, I agree. Here's something Jean showed me a few months back." And he took the keyboard and began to modify the test tables.

```
!|Message sent to manufacturing|
|Suit registration approval request?|message argument?|message sender?    |
|true                               |314159           |Outside Maintenance|
```

"OK, I see." I said. "We're simply asking whether or not the message is the right kind. We aren't even mentioning the notion of an ID."

"Right. There's no point in exposing a detail like the message ID, especially if it's likely to change."

Jasper saved the page and ran the test. Of course it complained.

"OK, now we have to add the `SuitRegistrationApprovalRequest` method to the fixture." Said Jasper.

"I'll do it." And I grabbed the keyboard and made the appropriate changes to the fixture.

```
public class MessageSentToManufacturing extends ColumnFixture {
  ...

  public boolean suitRegistrationApprovalRequest() {
    return message.id.equals(SuitRegistrationApprovalRequest.ID);
  }

  ...
}
```

I saved this and ran the test, and it passed.

For a second Jasper's eyes showed the same old excitement, but he reigned it back in and simply said: "OK, good. Now how about the rest of the acceptance tests?"

We found another that was failing for the same reason, and quickly fixed it. That left one more failing acceptance test: `RejectDuplicateRegistration`.

"Oh yeah!" I said happily. "That's what we started working on this morning. This one should be failing because we haven't finished getting it working yet."

fitlibrary.DoFixture			
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." I said. "This is failing because the `Registrar` is not detecting that suit# 314159 has already been registered. We should be able to fix that pretty easily."

"Be my guest." Said Jasper, still somewhat subdued.

So I brought up the code for the `Registrar` and found the appropriate method.

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

"Yeah, see, it's just passing the request to manufacturing without checking."

So I modified the code as follows:

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

I talked while I typed. "OK, now we need to write the `IsSuitRegistered` method."

Jasper stopped me. "No, Alphonse. Now we need to write the unit test for `IsSuitRegistered`."

"Right! I almost forgot." So I looked for the appropriate unit test to modify, but couldn't fine one.

"Wow, it looks like we wrote those gateways without any unit tests!" I brought up the `InMemorySuitGateway` that we'd been using.

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

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

  public Suit[] getArrayOfSuits() {
    return (Suit[]) suits.values().toArray(new Suit[0]);
  }
}
```

Jasper looked at the code and said: "OK, this is pretty simple code. I can see why the unit tests didn't get written, but it's time to write one now." And he grabbed the keyboard and started to write the test.

```
public class InMemorySuiteGatewayTest extends TestCase {
  public void testIsSuitRegistered() throws Exception {
    InMemorySuitGateway g = new InMemorySuitGateway();
    assertFalse(g.isSuitRegistered(314159));
    g.add(new Suit(314159, new Date()));
    assertTrue(g.isSuitRegistered(314159));
  }
}
```

Then, step by step, he wrote the implementation.

```
public boolean IsSuitRegistered(int barcode) {
  return suits.containsKey(barcode);
}
```

The unit tests all passed.

"OK, now we have to connect that to the `SuitGateway` class."

```
public interface ISuitGateway {
  ...
  boolean isSuitRegistered(int barcode);
}

public class SuitGateway {
  public static ISuitGateway instance = null;

  ...

  public static boolean isSuitRegistered(int barcode) {
    return instance.isSuitRegistered(barcode);
  }
}
```

We ran the unit tests, and they all passed. Then we ran the `RejectDuplicateRegistration` acceptance test.

fitlibrary.DoFixture			
start	dtrack.fixtures.DTrackFixture		
set suit	314159	as registered	
check	register suit	314159	false
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

"Great!" I said. "That first failing cell of the test is now passing. Only two more to go."

Jasper looked up at me and said: "I need a break. Let's go watch the starbow for awhile."

"Sounds like a plan." Working with Jasper was tiring. Perhaps he found working with me to be tiring too.

The code for this article can be located at:

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

To be continued...

The Craftsman: 42
Dosage Tracking XIX
Devious Thoughts

Robert C. Martin
2 October 2005

...Continued from last month.

The High Altitude Nuclear Explosions (HANES) went off with clockwork precision. General MacArthur nodded with quiet satisfaction as he received the reports of the detonations. Better still, reports from covert agents confirmed that preparations for the invasion appeared to have ground to a halt. MacArthur eagerly anticipated the satellite photos in the coming days. He expected them to show that the Axis powers were conducting a mass withdrawal and skulking back to their bases.

But as the days and weeks wore on, no photos arrived. To his growing dismay, MacArthur learned that virtually every spy satellite in orbit during the HANES and those that had been launched days, and even weeks later, quickly stopped operating. Eventually he was told that it had something to do with high energy radiation belts created by the explosions.

General MacArthur did not like being blind.

---

22 Feb 2002, 1030

"Jasper is driving me nuts!" I said to Jerry in the break room, once I had found a way to escape from Jasper.

Jerry smiled and nodded knowingly. "Yeah, I know what you mean. He's a bit intense isn't he?"

"A bit? I've confronted him two or three times, and yet he keeps on being condescending. It's like he wants to be my big brother or something."

"Jasper is a good programmer, Alphonse. There's a lot you can learn from him. So try to get passed his personality quirks. OK?"

"If you say so. I haven't been impressed so far. He makes me crazy!"

"Well, you've been working with him for a couple of hours today. Perhaps it's time to switch partners. If you like, I'll ask him to help Avery, while I work with you."

That sounded like a really great idea. Getting away from Jasper, even for just a few hours, had become an imperative. "I'd be in your debt!"

"OK, I'll take care of it. Besides, it should be fun to see how *those* two get along." Jerry gave me a wink; and I realized that putting Avery and Jasper together could indeed light off some fireworks.

I wandered over to a window to watch the starbow make it's lazy circular trek around our ship. The starbow was a hoop of colored stars that encircled our bow, positioned as though the ship were trying to fly through it. It's leading edge was a pale blue that grew in intensity to white at the centerline and then faded away into red. At slower speeds, the colors would fade and the starbow would widen to become the stars of

the nighttime sky. I knew the effect was caused by a combination of Doppler shift, and optical aberration, but that didn't keep me from being mesmerized by it's stark beauty.

Jerry came by a few minutes later and said: "OK, it's all set. This is actually a good breaking point for Avery and I, since we just finished the story we were working on. I've asked Jasper to work with Avery on the story that the two of you were doing. You can help me write some more acceptance tests."

This was good news. I could use a break from the code; and I was tired of that Suit Registration story too. But I had a question.

"Isn't Carole supposed to be writing the acceptance tests, Jerry? After all, she's our customer, isn't she?"

"Oh, she *is* writing them. You'll see her working with Jean and I on them. But she can't write them all, so we help her."

We started walking back to the lab as we talked. Jerry was thoughtful for a second and then said:

"The normal process is for the customer, or analysts that report to the customer, to write the primary acceptance tests, and for QA to write the alternate acceptance tests."

"What's a primary acceptance test?"

"Oh, yeah, you haven't met Ivar yet. It's a term that Ivar uses to describe a test that describes the 'happy path' of a feature."

"Happy path?"

"Yeah, when everything goes right. You know, no invalid entries, no failures, no exceptions thrown, no cockpit error."

"Oh, ok, so that first Suit Registration acceptance test we did was 'Primary' because it tested that a suite would be registered if everything went OK. But the second one we were working on showed that registration failed if the suit was already registered, so it would be – er – Alternate?"

"Yeah, you got it. Anyway, QA are the folks who are trained to think of all the things that can go wrong. They explore the boundary conditions and the failure scenarios. So they usually write the 'Alternate' acceptance tests."

"Usually?"

"Yeah. Have you seen any QA folks in the lab?"

"No, it's just been you, me, Avery, Jasper, Carole, and Jean. We see Jasmine and Adelade from time to time, but they are working on SMC."

Jerry nodded. "Right. We don't have QA on our team yet. Jean's trying to get one or two QA folks assigned. Until then, it's up to us."

"OK, I see. So you and I are going to write some alternate acceptance tests?"

"Yeah, that's the plan. We'll look at the primary tests that Carole has written, and then try to imagine everything that could go wrong with them and write tests for those cases."

We walked in silence until we reached the lab, and then sat down where Jerry and Avery had been working. Jerry pulled up the FitNesse site and navigated to the `StoryDescriptions` page. It looked like this:

```
!path C:\DosageTrackingSystem\classes

^RegisterSuit
^UnRegisterSuit
^AlertManagerIfOutsideUserPastDosageLimit
^SuspendUser
^SuspendedUserAttemptsToCheckOutSuit
^AddUser
^DeleteUser
^UserDosageReport
^UserHistoryReport
^SuitHistoryReport
^SuitInventoryReport
```

```
^CheckOutSuit
^CheckInSuit
^AttemptToCheckOutSuitThatRequiresInspection
^AttemptToCheckOutSuitWhenUserOverMonthlyLimit
^SuitInspection
```

"Wow!" I said. "Somebody's been busy."

"Yeah, Carole and Jean have been entering primary acceptance tests since yesterday. They've got a lot of them done. Let's look at `UnRegisterSuit`."

Jerry clicked on the link, and the screen showed the following:

```
^UnRegisterRegisteredSuit
^UnRegisterCheckedOutSuit
```

"OK, I'll bet that first page simply unregisters a registered suit."

Was this a joke? I flashed Jerry a big Jasper grin and said: "Given the name, I'd say that's a good bet."

Jerry smirked and clicked the link.

Unregister a suit.

Users can unregister a suit that has been properly registered. This will remove the suit from inventory.

set suit	314159	as registered		
check	unregister suit	314159	true	
check	count of registered suits is	0		
check	message	Suit 314159 unregistered.	was printed	true

We both examined the page for a few seconds. Then Jerry said: "OK, that's what I thought. She's loading a registered suit into the database and then unregistering it. Then she makes sure that the count of suits has been decremented, and that the appropriate message was displayed."

"Wait!" I said. "Jasper and I just wrote some of those fixture function in the last hour or so. How did Carole and Jean know to use them?"

"Didn't you tell them?"

"No, should we have?"

"Well, it would have been polite. They do have a lot of acceptance tests to write, and I'm sure they'd appreciate any help they can get."

"Uh..."

"Anyway, these tests *are* on a wiki. They must have seen what you and Jasper did, and made use of your ideas."

"Really?"

"What? Don't you think your ideas are worth using?"

"I, uh..."

"We re a team, Alphonse. We're not a group of disconnected developers. We look at each other's work and learn from it."

"OK, sure, I just didn't expect it so soon, that's all."

"So let's look at that `UnRegisterCheckedOutSuit` acceptance test. I'll bet it tries to unregister a suit that's been checked out, and makes sure the unregistration fails."

“Checked ou?”

“Yeah...being worn by someone outside the ship.”

“OH! Yeah, it wouldn’t be a good idea to unregister a suit that somebody was using.”

Jerry grimaced in agreement and clicked on the link.

You cannot unregister a suit that is checked out.

If a user has checked out a suit, then we cannot allow the suit to be unregistered.

set suit	314159	as registered		
set suit	314159	as checked out to user	Bob	
check	unregister suit	314159	false	
check	count of registered suits is	1		
check	message	Could not unregister suit 314159. Checked out to Bob.	was printed	true

I looked it over and beat Jerry to the punch. “Yeah, she’s just forcing the suit to be checked out, and then showing that unregistration fails.”

“Right. OK, so what has she missed?”

“You mean on this page?”

“No, I mean, what other tests has she missed. Think deviously!”

“I, uh...” What did it mean to think deviously?

“For example, what would happen, Alphonse, if you tried to unregister a suit that was already unregistered? What *should* happen?”

“Oh! I see what you mean. Well, clearly we’d want the unregistration request to fail, and display some appropriate message.”

“Good. So write the test for that!”

I thought about it for a second and then created a new page named UnRegisterUnRegisteredSuit. I smilled when I wrote that name. It thought it was clever. Jerry rolled his eyes, but remained silent. So I wrote the following.

You can't unregister a suit that's not already registered.

Any attempt to do so should be ignored with an appropriate error message.

check	unregister suit	314159	true	
check	count of registered suits is	0		
check	message	Suit 314159 was not registered.	was printed	true

Jerry looked it over and said: “Yeah, that looks about right.” Then he looked up and called over to Carole: “Hay Carole, look at the `UnregisterUnregisteredSuit` page. Does that look right?”

Carole nodded and spend a few seconds apparently navigating to the page. Then she gave us a smile and a thumbs-up and got right back to work.

"Great." Said Jerry. "Now, Alphonse, any other devious thoughts?"

I thought for a bit and said: "No, I can't think of any other alternate tests."

"OK, then. let's look at the next story."

The code for this article can be located at:

`http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_42_DosageTrackingSystem.zip`

To be continued...

The Craftsman: 43 Dosage Tracking XX Language Lawyers

Robert C. Martin
27 October 2005

...Continued from last month.

May, 1944. Henry Wallace had a brass paperweight in the form of a globe of the Earth. He kept it on his desk in the Oval Office. Embossed on the bottom was: "29 Apr, 1959, 0943 GMT". "Fifteen years." He mumbled to himself. "Fifteen years." He wondered how he could save a world that was so hell bent on destroying itself. He reckoned that he had two bad options: collaborate or conquer. The first seemed infeasible given the belligerence of the axis leaders; they yearned to conquer the Americas, not collaborate with them. The second option was feasible, but distasteful. He could, in fact, conquer the world, and he could do it quickly. But could he keep it dominated for the necessary fifteen years? Not likely.

One thing was certain. Option two would not last long. Having seen that nuclear explosions were possible; Hitler, Stalin, and Tojo had to be pushing their remaining scientists to the breaking point.

---

22 Feb 2002, 1100

Just as we were about to start on the next test case, Jasper walked over with that big depressing grin plastered all over his face.

"Hey there Jerry! Can I borrow you for a second or three?"

Jerry looked up at Jasper without a flinch and said: "Sure, Jasper. Alphonse, I'll be back in a few minutes." Jerry got up and walked away with Jasper as if nothing were wrong; but then he quickly turned around, rolled his eyes, and winked at me.

I smiled back at both of them and said: "OK see you soon. I'll go see what Avery is up to."

I sauntered over to where Avery was sitting and said: "Hi."

Avery looked up and gave me a desperate grimace. "Why did you leave me with that moron?"

"He's not a moron. Actually he's pretty smart. But he can be trying."

"Trying? If he smiles at me one more time I'm going to shove this keyboard into his mouth. And it'll fit too."

We both laughed, making sure no one could overhear us. Then Avery said: "Hey, have you looked at the new features in Java 5"

"Yeah, you mean autoboxing, scanners, generics, and stuff like that?"

"Yeah, have you used any of it yet?"

"Not so far, although we could have used generics yesterday when we wrote a mock database. I just didn't think of it then."

Avery's eye's sparkled. "Oh yeah? Show me!"

This sounded like fun. So I sat down next to him and brought up the `InMemorySuitDatabase`.

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

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

  public Suit[] getArrayOfSuits() {
    return (Suit[]) suits.values().toArray(new Suit[0]);
  }

  public boolean isSuitRegistered(int barcode) {
    return suits.containsKey(barcode);
  }
}
```

We both looked at this for a second and then Avery said: "Yeah, we can change that `HashMap` real easily." He grabbed the keyboard and made the following change:

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

The tests all still ran.

"Cool!" I said. Now we should be able to use autoboxing in the `add` method! So I grabbed the keyboard and did this:

```
public class InMemorySuitGateway implements ISuitGateway {
  private Map<Integer, Suit> suits = new HashMap<Integer,Suit>();

  public void add(Suit suit) {
    suits.put(suit.barCode(), suit);
  }
  ...
}
```

The tests all still ran!

"Way cool! That's much cleaner than before!" I said.

"Yeah! And now we can get rid of that ugly cast in `getArrayOfSuits`." Avery grabbed the keyboard and deleted the cast.

```
  public Suit[] getArrayOfSuits() {
    return suits.values().toArray(new Suit[0]);
  }
```

The tests ran just fine.

"Sweet!" I said. Too bad we don't have anything else to clean up with generics. This is fun!

"Well..." said Avery. "Have you considered whether or not there will be more than one kind of `Suit`?"

"What? You mean like different derivatives of `Suit`?"

"Yeah. Imagine that you have two derivaties like `MensSuit` and `WomensSuit`.

"You mean like this?" And I grabbed the keyboard and typed the classes in a junk package.

```
public class WomensSuit extends Suit {
  public WomensSuit(int barCode, Date nextInspectionDate) {
    super(barCode, nextInspectionDate);
  }
}

public class MensSuit extends Suit {
  public MensSuit(int barCode, Date nextInspectionDate) {
    super(barCode, nextInspectionDate);
  }
}
```

Avery watched as I typed, and then said: "Yeah. Now create a class that has two lists. One a list of `MensSuit` and the other a list of `WomensSuit`."

"OK." I said, and I typed the following:

```
public class SuitInventory {
  private List<MensSuit> mensSuits = new ArrayList<MensSuit>();
  private List<WomensSuit> womensSuits = new ArrayList<WomensSuit>();
}
```

Avery nodded and said: 'Great! Now let's assume we want to inspect suits. We can use a method named `inspect` that takes a list of suits."

"You mean like this?" And I typed the following:

```
  void inspect(List<Suit> suits) {
      for (Suit s : suits) {
        // inspect s.
      }
  }
```

"Impressive, Alphonse! You made fine use of the new iteration syntax."

I had missed the old formal banter. Sometimes working with Avery was a lot of fun. Especially after working with Jasper. "Indeed I did, Avery, you are quite observant!"

"No more observant than you are creative; but shall we continue?"

"Indeed, let's."

"Well then, suppose that we now decide to create a method for inspecting only `WomensSuit` objects?"

"I imagine I can write such a method. Shall I?"

"Please do."

And so I typed the following:

```
  void inspectWomensSuits() {
    inspect(womensSuits);
  }
```

"Hay!" I said, forgetting teh formal banter for the moment. "That doesn't compile!"

"As, indeed, it should not!" Quipped Avery. He maintained his formal composure.

"But I don't understand, it should compile!"

"No, Alphonse, if you look carefully you'll notice that we are attempting to pass a `List<WomensSuit>` to an argument defined to take a `List<Suit>`."

"Sure, but a `List<WomensSuit>` *is* a `List<Suit>`!"

"Would you like to reconsider that statement Alphonse?"

"Uh..." I thought for a moment and realized that that wasn't quite right. "OK, technically you're right, a `List<WomensSuit>` does not derive from `List<Suit>`, but a list of womens suits is still a list of suits."

"Not a all! You can add an instance of `MensSuit` to a `List<Suit>`, but you can't add a `MensSuit` to a `List<WomensSuit>`."

"OK, that's true, but so what?"

"So, my dear Alphonse, a `List<WomensSuit>` is not substitutable for a `List<Suit>`, and is therefore not a subtype of `List<Suit>`. You are of course aware of the Liskov Substitution Principle[1], are you not?"

That's when it clicked. "Of course. Of course! Substitutability is the basis of subtying. Therefore a list of derivatives is not a derivative of the list of bases."

"Precisely Alphonse!"

"Yes. Precisely! But that leaves us with a problem, doesn't it my dear Avery?"

"Indeed it does! If a list of derivatives is not subtitutable for a list of superclasses, what is?"

"And is there an answer to this connundrum?" I asked?

"Indeed there is. Shall I demonstrate?"

"Please do!"

So Avery took the keyboard and made the following change:

```
void inspect(List<? extends Suit> suits) {
    for (Suit s : suits) {
      // inspect s.
    }
}
```

I looked at the strange syntax that Avery had written. "Now that is one strange syntax; but I think I understand the intent. The question mark indicates that the type is unknown, and the remainder of the clause tells the compiler that the unknown type is a derivative of `Suit`."

"Indeed, indeed. That is the correct interpretation Alphonse. Notice also that the whole program now compiles."

"Yes, I see that. This is all fascinating! There is great power and expressiveness in these new features."

"Yeah, and they're fun too!" Avery said, switching modes.

I gave him a poke in the ribs, and he bopped me on the head with a one of the books on the table.

"Now, now boys, don't go breaking anything." It was Jean. She must have come by to see what we were up to. "Goodness! You boys certainly have a lot of energy." She looked at the screen and exclaimed: "Oh! I see you boys are having fun with the rather odd new syntax for generics. For my money it's all a bit involved, isn't it. So much syntax for so little effect. I fear the language lawyers may be leading us into a deep and unecessary complexities. Such a shame too. But then I suppose youngters like you revel in such intricacies, don't you. Well then, carry on, carry on. You are such fine young men."

Jean waddled off, while Avery and I suppressed giggles.

The code for this article can be located at:

---

[1] See http://www.objectmentor.com/resources/articles/lsp.pdf

http://www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_43_DosageTrackingSystem.zip

To be continued...