The Craftsman: 20 SMCRemote Part X Backslide.

Robert C. Martin
28, Nov, 2003

...Continued from last month. You can download last month's code from:
www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_19_Tolerance.zip

---

I returned to the lab after lunch, but Jean was not there. There was neither note, nor email from her; and her knitting basket was nowhere to be seen. After waiting a few minutes, I decided to sit down and continue working on the `SCMRemoteServer`.

So far the server doesn't serve anything. It's just a couple of functions that build and execute the SMC command line. It seemed reasonable to me that the server code should open a socket and accept connections from the `SMCRemoteClient` that we wrote earlier.

I was pretty frustrated with our pace today. We'd worked the whole morning and only gotten two small functions, and their attendant unit tests written. I felt like making *progress*. So I grabbed the keyboard and began to type.

"First," I thought, "this server needs to serve something. So lets use the `SocketServer` class that Jerry and I built last week."

Getting the server to serve was pretty simple. I just needed to write a constructor that created a `SocketService` object and passed in a `SocketServer`. Then the `SocketServer.serve` method would automatically be called as soon as `SMCRemoteClient` tried to connect.

```
public SMCRemoteServer() throws Exception {
  service = new SocketService(9000, new SocketServer(){
        public void serve(Socket socket) {
          // SMCRemoteClient has connected.
        }
      });
}
```

I looked in the `SMCRemoteClient` code and saw that the client expects a string to be sent upon connection. That string should begin with "SMCR". So I wrote that string in the `serve()` method.

```
        public void serve(Socket socket) {
          // SMCRemoteClient has connected.
          try {
            ObjectOutputStream os =
                  new ObjectOutputStream(socket.getOutputStream());
            os.writeObject("SMCR");
          } catch (IOException e) {
```

```
    }
  }
```

Next the server needs to read a `CompileFileTransaction` from the client. It needs so write the files contained in that transaction, invoke the compiler, and then return the resulting files in a `CompilerResultsTransaction`. This didn't seem like a hard thing to write so...

```
  public void serve(Socket socket) {
    // SMCRemoteClient has connected.
    try {
      ObjectOutputStream os =
            new ObjectOutputStream(socket.getOutputStream());
      os.writeObject("SMCR");
      ObjectInputStream is =
            new ObjectInputStream(socket.getInputStream());
      CompileFileTransaction cft =
            (CompileFileTransaction)is.readObject();
      String filename = cft.getFilename();
      cft.sourceFile.write();
      String command = buildCommandLine(filename);
      executeCommand(command);
      //OK, what file do I put into the Result?
    } catch (Exception e) {

    }
  }
```

Hmmm. Compiling the file didn't seem very difficult, but what output file should I put into the result transaction? What is its name? I remember from this morning that Jean had coached me in writing a test for invoking a compile. I looked at that test and found that the input file has a `.sm` suffix, and the output file as a `.java` suffix. So all I had to do is replace "`.sm`" with "`.java`" in the filename.

```
  public void serve(Socket socket) {
    // SMCRemoteClient has connected.
    try {
      ObjectOutputStream os =
            new ObjectOutputStream(socket.getOutputStream());
      os.writeObject("SMCR");
      ObjectInputStream is =
            new ObjectInputStream(socket.getInputStream());
      CompileFileTransaction cft =
            (CompileFileTransaction)is.readObject();
      String filename = cft.getFilename();
      cft.sourceFile.write();
      String command = buildCommandLine(filename);
      executeCommand(command);
      //Figure out the file name.
      String compiledFile = filename.replaceAll("\\.sm", ".java");
      CompilerResultsTransaction crt =
            new CompilerResultsTransaction(compiledFile);
      os.writeObject(crt);
      socket.close();
    } catch (Exception e) {

    }
  }
```

This looked like it should work. All I needed to do was to start up the server and run the client. That should be pretty simple. So I created a source file in my directory named F.sm, just like the one that Jean

had me create this morning.

```
Context C
FSMName F
Initial I
{I{E I A}}
```

And then I wrote a simple main function in `SMCRemoteServer`.

```
  public static void main(String[] args) throws Exception {
    SMCRemoteServer server = new SMCRemoteServer();
  }
```

And then I ran it. And it just hung there, waiting for a client to connect. Now *this* was exciting! I was getting something *done*! So next I ran the client with "`F.sm`" as its argument. And it *RAN!* Or rather it exited after several seconds with a normal exit code, and without any error messages either from the client or the server. Cool!

But what had it done? I couldn't immediately tell. So I checked the directory and found an `F.java` file sitting there! It had the right date on it, so it must have been created by my run of the client. Way cool! I opened the file and it looked like generated Java code. It even said it was generated by SMC. My code worked!

It had been a couple of weeks since I had felt this good; since before working with Jerry as a matter of fact. *This* was what programming was all about! I was filled with the rush of getting code working. I was *invincible*! I got up and did a little jig around my seat chanting "Oh yes! Oh yes! I'm a programmer. Oh yes!"

Jerry must have been nearby, because he came into the lab at that point. "Hay, Alphonse, what's are you celebrating?"

"Oh, hi Jerry! Look at this! I just got the `SMCRemoteServer` working!"

"Really? That's great Alphonse." The look on his face was funny -- as if he didn't believe me. So I showed him. I showed him how the server was sitting there running. I ran the client again. I showed him that an `F.java` file with the right date was created. I even showed him that it contained generated java code.

Jerry looked at me almost fearfully, and then nervously glanced at the door. "Alphonse, where are your tests?"

"Jerry, you don't need unit tests for something this simple. Look, it was just a dozen lines of code or so. Jerry, I made *progress* here. I got something *done*. And it didn't have to take all day! I think you guys waste too much time on all those unit tests!"

Jerry just stared at me for a few seconds, as if he couldn't quite process what I was telling him. Then he shut the door to the lab and sat down next to me.

"Alphonse, has Jean seen this?"

"No, she hasn't come back from lunch yet."

"Delete it, Alphonse."

"Delete what?"

"Delete the code you just wrote. It's worthless."

I had half expected this kind of reaction from someone. I drew my mouth into a sneer and said: "Oh, come on Jerry! It works! How could it be worthless?"

"Are you sure it works, Alphonse?"

"You saw it for yourself!"

"Are YOU sure it works?"

"Of course it works. The `F.java` file is proof!"

"Alphonse, why is the date of the `F.sm` file exactly the same as the date on the `F.java` file?

"What?" I looked at the directory, and sure enough they were identical to the second. But that didn't make any sense. I had written the `F.sm` file by hand several minutes ago. Why did it have the same date as the `F.Java` file that was created seconds ago when I demonstrated the system to Jerry? I stared at it for awhile, and then admitted that I didn't know.

"Delete the code, Alphonse. You don't understand it."

"Aw, come one Jerry, I understand it. I just can't figure out why the dates don't look right. It works Jerry...it works." But I wasn't as sure any more. *Why was that date different?*

"Alphonse, by any chance did you happen to run the client and server in the same directory?"

"Uh..." I hadn't specified a different directory for them so I guessed they must have been running in the same place. "...I guess so."

"So the client and the server were both reading and writing the same files in the same directory?"

"...oh..."

Jerry nodded grimly. "Yeah."

I shook my head. "OK, Jerry, you've got a point. I don't understand everything that's going on. But look, there's an `F.java` file there. *Something* had to be working!"

"So what? You don't know what is and what isn't working. You don't understand what you've done. The things that appear to be working may be working by accident. Is it possible, for example, that that `F.java` file was left over from a previous unit test?"

It *was* possible! Yikes, Jean and I had caused the system to write an `F.java` file this morning! "Damn! Yes, it's *possible* -- but it's not very likely!"

"Delete the code, Alphonse. It's crap."

I just stared at him. I had made *progress* damn it! Now he wanted me to delete all that progress. But he was right. I didn't understand this code. And I had no tests that would demonstrate, step by step, that everything was working as it should. If my code was working (and now I was really starting to wonder just how much of it was working) it was working more by accident than by design.

"Delete the code, Alphonse. You don't want Jean to see this. Right now she things the world of you, and this would be very disappointing for her."

My resolve finally failed. My shoulders fell, my head hung, and I reached over and deleted the code.

Jerry walked out of the room, shaking his head.

A few minutes later, Jean walked in. "Hello Alphonse dear. I'm sorry I'm late, but I got into a conversation with an old friend of mine and we started comparing pictures of our grandchildren. I love showing the pictures of my grandchildren. Have you seen them dear? Oh, never mind dear, we've got work to do you and I. What have you been up to while I was detained?"

"Nothing, Jean, I was just waiting for you."

To be continued...

The code that Alphonse and Jean finished can be retrieved from:

www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_20_Backslide.zip