# The Craftsman: 18
# SMCRemote Part VIII
# Slow and Steady

Robert C. Martin
24 Sept, 2003

*...Continued from last month. You can download last month's code from:*
*www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_17_Call_the_Guards.zip*

---

"All right then dear, I think it's time we started to work on the server part of this application, don't you? After all the client portion seems to be working as well as we can tell, and a client without a server just seems so lonely. Don't you agree?"

Jean had just settled her ample behind into her seat. As she spoke she reached into her bag and pulled out her knitting. The shawl (or whatever it was) had grown considerably during our break (during which I had learned all about Jean's children, grandchildren, nephews, and nieces)

"Er, server?" Two words were all I could ever seem to muster in response to one of Jean's vocal deluges.

"Yes dear, the server. The server is going to have to receive that file you've been so diligently sending, save it to disk, and then compile it with SMC. Remember dear, we are building a remote compiler for SMC, the State Machine Compiler. Now, dear, what's the first test case we should write?

What test case indeed? I thought furiously about this. The server side of the pair of programs had to listen at a socket and receive `CompileFileTransaction` objects. These objects contain encapsulated files in `FileCarrier` sub-objects. The server needs to write these encapsulated files to a safe place on the disk, compile them, and then send the results back to the client in a `CompilerResultsTransaction` object.

My first worry was how to compile the files. Somehow we needed to invoke the compiler and get it to compile the saved file.

"Uh, could we write a test case for invoking the compiler?"

Jean smiled the way a grandmother smiles at an infant taking its first steps. "Why of course, dear, that would be a lovely place to start. Do you know how to invoke the compiler? Let's see, I think we just have to build the right command line and then invoke it. Now what was that command line syntax? It's been so long since I used SMC, I don't quite remember. Hear, dear, type this command: `java smc.Smc`. That's right dear, now what does that error message say? Class def not found? Oh yes, we forgot the classpath. Don't get old dear; you start forgetting things. Type this: `java -cp C:/SMC/smc.jar smc.Smc`.[1] That's better, now what does that message say?"

The message on the screen was usage error that described all the command line arguments for SMC. "Er, it's a usage message."

---

[1] You can download `smc.jar` from:
`http://www.objectmentor.com/resources/downloads/smc_java`

"So it is, dear, so it is. And it looks to me as though we want to invoke SMC with the following command: `java -cp C:/SMC/smc.jar smc.Smc -f myFile.sm`. Don't you agree dear? Why don't you write a test case that will generate that command line?"

So I created a new unit test class named `SMCRemoteServerTest`.

```
public class SMCRemoteServerTest extends TestCase {
  public void testBuildCommandLine() throws Exception {
    assertEquals("java -cp C:/SMC/smc.jar -f myFile.sm",
      SMCRemoteServer.buildCommandLine("myFile.sm"))  ;
  }
}
```

And I wrote the `SMCRemoteServer` class too.

```
public class SMCRemoteServer {
  public static String buildCommandLine(String file) {
    return null;
  }
}
```

"Very good, dear. The test fails, as it should. You've been learning your lessons well. Now it should be a simple matter to make that test pass, don't you think?

"Er -- sure."

```
  public static String buildCommandLine(String file) {
    return "java -cp C:/SMC/smc.jar smc.Smc -f " + file;
  }
```

"That's fine dear, now run it for me, won't you? Good. Oh! Why, the test failed dear, what could be the matter?"

The message on the screen was: `expected:<.....> but was:<...smc.Smc ...>`. I looked carefully at the code and realized that I had forgotten the `smc.Smc` in the test case. "Er, I guess I wrote the test wrong."

"So you did, so you did. Yes, sometimes we do write the tests wrong. Bugs can occurr anywhere. That's why it's always a good idea to write both tests and code. That way you are writing everything twice. I have a nephew who does accounting in ship stores, and he always enters every transaction into two distinct ledgers. What did he call that? Oh, yes, *dual entry bookkeeping*. He says it helps him prevent and track down errors. And that's just what we're doing, isn't it dear. We write every bit of functionality twice. Once in a test, and once in code. And it *does* help us to prevent and track down errors doens't it dear?"

While she was droning, I fixed the test case. Jean is a nice old lady, and pretty smart too, but my ears were ringing from the continual chatter.

```
  public void testBuildCommandLine() throws Exception {
    assertEquals("java -cp C:/SMC/smc.jar smc.Smc -f myFile.sm",
      SMCRemoteServer.buildCommandLine("myFile.sm"))  ;
  }
```

This made the test pass.

"Wonderful, dear, wonderful; but the code is a bit messy, don't you think? Let's clean it up before we go any further. Clean your messes before they start, I always say."

I looked at the code and didn't see anything particularly messy. So I looked back and Jean and said: "Er, where would you like to start?"

"My goodness dear; why it's as plain as the nose on your face! That `buildCommandLine` function needs a bit of straightening. That string in the return statement is just full of unexplained assumptions that are sure to confuse the next person to read this code. We don't want to confuse anyone, do we? I should

say we don't. Here, dear, give me the keyboard."

She set down her knitting (is that the beginnings of a sleeve) and with considerable effort took control of the keyboard. Her typing was slow but deliberate, as if each keystroke caused her pain. Eventually she changed the return statement as follows:

```
return "java -cp " + "C:/SMC/smc.jar" + " " + "smc.Smc" + " -f " + file;
```

Then she ran the test, and it passed.

"There, dear, do you see now? We need to replace some of those strings with constants that explain how the command line is built. Do you want to do it dear, or shall I?"

Her pained expression was answer enough. I took the keyboard and added the constants I thought she was after.

```
public class SMCRemoteServer {
  private static final String SMC_CLASSPATH = "C:/SMC/smc.jar";
  private static final String SMC_CLASSNAME = "smc.Smc";

  public static String buildCommandLine(String file) {
    return "java -cp " +
      SMC_CLASSPATH + " " +
      SMC_CLASSNAME +
      " -f " + file;
  }
}
```

"Yes, dear, that's just what I was after. Don't you think that others will make a bit more sense out of it now? I know I would if I were them. Now dear, why don't you and I go to the break room and settle our brains?"

Something inside me snapped.

"Jean, we've barely gotten anything done!"

"Why dear, whatever do you mean, we've gotten quite a bit done."

I wasn't thinking about whom I was talking to. I kept right on blathering. "We've only gotten one stupid function done. If we keep moving at this snail's pace, Mr. C is going to transfer us to the husbandry deck to clean out the Dribin cages!"

"Really dear, why ever would you think such a thing? I've been working in this department for well over thirty years, dear, and nobody has ever suggested that I should be cleaning up after Dribins."

She stopped for a moment as if composing her thoughts. Then she looked at me with a kindly, but stern, expression on her face.

"Alphonse dear, the only way to get done with a program quickly, is to do the very best job you can with it. If you rush, or if you take shortcuts, you will pay for it later in debugging time. Mr. C. is paying you for your time, dear, and he wants the very best work you can do. The code is your product, dear, and Mr. C. doesn't pay for poor products. He doesn't want to hear that you saved a day by being sloppy because he knows that he'll have to pay dearly for that sloppiness later. What he wants is the very best code you can write. He wants that code to be as clean, expressive, and well tested as you can make it. And, dear, Mr. C. knows that if you do that, you'll be done with it quicker than all those young fools who think they can go faster by cutting corners. No, dear, we aren't going too slow. We're moving along at just the right pace to get done as quickly as possible. Now, let's go get a cup of tea, shall we?"

*To be continued...*

*The code that Alphonse and Jean finished can be retrieved from:*
*www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_17_Slow_And_Steady.zip*