The Craftsman: 12
SMCRemote Part II
Three Ugly Lines

Robert C. Martin
18 March, 2003

...Continued from last month. See <<link>> for last month's article, and the code we were working on. You can download that code from:

www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_11_SMCRemote_1_WhatsMain.zip

---

I spent my break up in the observation deck. We had a little bit of excitement as the ice shield passed through a thick patch of particles that made the ice flicker with blue flashes and transient patterns all over it's surface.

As always, Jerry was waiting for me after break. I wondered if he ever actually took any of this breaks. He beamed at me and said:

"OK, let's ship a file over the socket."

"Did you watch the Cherenkov display?" I asked.

"A beauty!" he said. I guess he *does* take his breaks. But where does he go?

"OK, lets ship a file." I said. "I'll write the test case." I grabbed the keyboard and started to type. The first thing I wrote was the code that created the file to be sent through the socket.

```
public void testSendFile() throws Exception {
    File f = new File("testSendFile");
    FileOutputStream stream = new FileOutputStream(f);
    stream.write("I am sending this file.".getBytes());
    stream.close();
}
```

"I know you like to create your data files in the test code rather than depending on them to just be there."

"Right." said Jerry. "But do you notice that you have quite a bit of duplication?"

I looked around in the test class and immediately saw that we had written almost this exact same code in the `testCountBytesInFile()` method that we wrote before our break.

"It's just a four lines of code." I said. "Hardly major duplication."

"True." Jerry replied. "But duplication should always be removed as soon as possible. If you allow duplicate code to accumulate in your project you'll find yourself with a huge source of confusion and bugs."

"OK, it's easy enough to fix." So I extracted a new function called `createTestFile()` and changed both `testCountBytesInFile()`, and `testSendFile()` to call it.

```
private File createTestFile(String name, String content)
```

```
    throws IOException  {
        File f = new File(name);
        FileOutputStream stream = new FileOutputStream(f);
        stream.write(content.getBytes());
        stream.close();
        return f;
    }
```

I ran the tests just to make sure I hadn't broken anything, and then continued writing the test. I knew that the test had to simulate `main()`, so I called all the functions that `main()` would have to call. Then I added one final call that would send the file.

```
    public void testSendFile() throws Exception {
        File f = createTestFile("testSendFile", "I am sending this file.");
        c.setFilename("testSendFile");
        assertTrue(c.connect());
        assertTrue(c.prepareFile());
        assertTrue(c.sendFile());
    }
```

"Good." Jerry said. You are anticipating that we'll need a method in the client named `sendFile()`."

"Right." I said. "This method will send the prepared file."

I turned back to the test and then got stuck. I didn't know how to proceed. How was I going to test that this file that I had created and "sent" actually got sent to the server. We didn't even *have* a server? Did I have to write the whole server before I could test this? What was I testing anyway?

I sat there for about a minute while Jerry watched expectantly. Then I turned to him and explained my dilemma.

"No, you don't need to write the server." Jerry said. "At this point all we are testing is the client's ability to *send* the file, not the servers ability to receive it. "

"How can I send the file without having a server to receive it?"

"You can create a stub server that does as little work as possible. It doesn't have to receive the file at first, it just has to acknowledge that you sent the file correctly."

"So...hmmm...something like this?"

```
        assertTrue(server.fileReceived);
```

"Yah, that ought to do." Jerry nodded. "Now why don't you make that test pass?"

I thought about this for a minute and then realized that this shouldn't be very hard. So I wrote a simple mock server that did nothing.

```
class TestSMCRServer implements SocketServer {
    public boolean fileReceived = false;
    public void serve(Socket socket) {
    }
}
```

Jerry said: "Ah, more duplication!"

I looked around and saw that before the break we had implemented a very similar mock server in the `testConnectToSMCRemoteServer()` method, so I eliminated it.

```
    public void testConnectToSMCRemoteServer() throws Exception {
        boolean connection = c.connect();
        assertTrue(connection);
    }
```

Then I started the server in the `SetUp()` method of the test, and closed it in `TearDown()`. Thus, before every test method was called, the server would be started; and then it would be closed when the test method returned.

```
protected void setUp() throws Exception {
    c = new SMCRemoteClient();
    server = new TestSMCRServer();
    smc = new SocketService(SMCPORT, server);
}

protected void tearDown() throws Exception {
    smc.close();
}
```

Finally, I wrote a dummy `sendFile()` method in `SMCRemoteClient`.

```
public boolean sendFile() {
    return false;
}
```

The tests failed as expected. I sighed and looked over at Jerry. "Now what?" I said.

"Send the file." he said.

"You mean just open the file and shove it over the socket?"

Jerry thought for a minute and said: "No, we probably need to tell the server to expect a file. So lets send a simple message and follow it with the contents of the file."

Jerry grabbed the keyboard and made the following changes to `SMCRemoteClient`. "First we have to get the streams out of the socket." He said.

```
public boolean connect() {
    boolean connectionStatus = false;
    try {
        Socket s = new Socket("localhost", 9000);
        is = new BufferedReader(new InputStreamReader(s.getInputStream()));
        os = new PrintWriter(new OutputStreamWriter(s.getOutputStream()));
        connectionStatus = true;
    } catch (IOException e) {
        e.printStackTrace();
        connectionStatus = false;
    }
    return connectionStatus;
}
```

"Then we have to make the file available for reading."

```
public boolean prepareFile() {
    boolean filePrepared = false;
    File f = new File(itsFilename);
    if (f.exists()) {
        try {
            itsFileLength = f.length();
            fileReader = new BufferedReader(
                            new InputStreamReader(new FileInputStream(f)));
            filePrepared = true;
        } catch (FileNotFoundException e) {
            filePrepared = false;
            e.printStackTrace();
        }
```

```
        }
        return filePrepared;
    }
```

"Finally, we can send the file."

```
public boolean sendFile() {
    boolean fileSent = false;
    try {
      writeSendFileCommand();
      fileSent = true;
    } catch (Exception e) {
      fileSent = false;
    }
    return fileSent;
  }

  private void writeSendFileCommand() throws IOException {
    os.println("Sending");
    os.println(itsFilename);
    os.println(itsFileLength);
    char buffer[] = new char[(int) itsFileLength];
    fileReader.read(buffer);
    os.write(buffer);
    os.flush();
  }
```

"Wow!" I said. "That was a lot of typing without testing."

Jerry looked at me sheepishly. "Yes, I'm a little nervous about that."

He hit the test button, and the test failed because `server.fileReceived` returned `false`.

"Whew!" said Jerry. "We dodged a muon pulse."

"So." I said. "You are going to precede the file with three lines. The first contains the string "Sending". The second contains the file name, and the third contains the length of the file. After that you send the file in character array."

"Right. I told you before the break that we'd need that file length."

"Hmmm. I guess so." I said.

"So now, all we have to do is receive the file in the mock server. Would you like to take a try?"

I was pretty sure I knew what to do. So I made the following changes. First I changed the test to make sure we got the filename, length, and the file contents.

```
  public void testSendFile() throws Exception {
    File f = createTestFile("testSendFile", "I am sending this file.");
    c.setFilename("testSendFile");
    assertTrue(c.connect());
    assertTrue(c.prepareFile());
    assertTrue(c.sendFile());
    Thread.sleep(50);
    assertTrue(server.fileReceived);
    assertEquals("testSendFile", server.filename);
    assertEquals(23, server.fileLength);
    assertEquals("I am sending this file.", new String(server.content));
    f.delete();
  }
```

Next I changed the mock server to parse the incoming data and make sure it was correct.

```
class TestSMCRServer implements SocketServer {
  public String filename = "noFileName";
  public long fileLength = -1;
  public boolean fileReceived = false;
  private PrintStream os;
  private BufferedReader is;
  public char[] content;
  public String command;

  public void serve(Socket socket) {
    try {
      os = new PrintStream(socket.getOutputStream());
      is = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      os.println("SMCR Test Server");
      os.flush();
      parse(is.readLine());
    } catch (Exception e) {
    }
  }

  private void parse(String cmd) throws Exception {
    if (cmd != null) {
      if (cmd.equals("Sending")) {
        filename = is.readLine();
        fileLength = Long.parseLong(is.readLine());
        content = new char[(int)fileLength];
        is.read(content,0,(int)fileLength);
        fileReceived = true;
      }
    }
  }
}
```

testConnectionToSMCRemoteServer is hanging because of is.readLine()

- the solution I came up with is to close all the sockets when the SocketService closes

Finally, I modified `SMCRemoteClient.connect()` to wait for the SMCR message sent by the mock server.

```
public boolean connect() {
   ...
   String headerLine = is.readLine();
   connectionStatus = headerLine != null &&
                      headerLine.startsWith("SMCR");
   ...
}
```

I didn't type this all at once. I didn't want to have to dodge another muon pulse. So I made the changes in much smaller steps, running the tests in between each step. I could tell that Jerry was impressed. He was still embarrassed about making such a big change. Eventually, when all the tests passed, I felt just a little superior, so I risked an observation.

"Jerry." I said. "This code is pretty ugly."

"What do you mean?"

"Well, sending those three lines, the filename, the length, and the string "Sending". That's ugly."

Jerry looked at me skeptically. He sat up as straight as he could and gave me a condescending look. "I suppose you know a better way."

"I think I do." I said, and I started to type...

To be continued.

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

www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_12_SMCRemote_I1_ThreeUgly Lines.zip