The Craftsman: One.

Robert C. Martin
13 February 2002
This chapter is derived from a chapter of Principles, Patterns, and Practices of Agile Software Development*, Robert C. Martin, Prentice Hall, 2002.*

Dear Diary,

13 February 2002,

Today was a disaster – I really messed it up. I wanted so much to impress the journeymen here, but all I did was make a mess.

It was my first day on the job as an apprentice to Mr. C. I was lucky to get this apprenticeship. Mr. C is a well recognized master of software development. The competition for this position was fierce. Mr. C's apprentices often become journeymen in high demand. It *means* something to have worked with Mr. C.

I thought I was going to meet him today, but instead a journeyman named Jerry took me aside. He told me that Mr. C always puts his apprentices through an orientation during their first few days. He said this orientation was to introduce apprentices to the practices that Mr. C insists we use, and to the level of quality he expects from our code.

This excited me greatly. It was an opportunity to show them how good a programmer I am. So I told Jerry I couldn't wait to start. He responded by asking me to write a simple program for him. He wanted me to use the Sieve of Eratosthenes to calculate prime numbers. He told me to have the program, including all unit tests, ready for review just after lunch.

This was great! I had almost four hours to whip together a simple program like the Sieve. I determined to do a really impressive job. Listing 1 shows what I wrote. I made sure it was well commented, and neatly formatted.

Listing 1
`GeneratePrimes.java version 1.`

```
/**
 * This class Generates prime numbers up to a user specified
 * maximum.  The algorithm used is the Sieve of Eratosthenes.
 * <p>
```

```
 * Eratosthenes of Cyrene, b. c. 276 BC, Cyrene, Libya --
 * d. c. 194, Alexandria.  The first man to calculate the
 * circumference of the Earth.  Also known for working on
 * calendars with leap years and ran the library at Alexandria.
 * <p>
 * The algorithm is quite simple.  Given an array of integers
 * starting at 2.  Cross out all multiples of 2.  Find the next
 * uncrossed integer, and cross out all of its multiples.
 * Repeat untilyou have passed the square root of the maximum
 * value.
 *
 * @author Alphonse
 * @version 13 Feb 2002 atp
 */
import java.util.*;

public class GeneratePrimes
{
  /**
   * @param maxValue is the generation limit.
   */
  public static int[] generatePrimes(int maxValue)
  {
    if (maxValue >= 2) // the only valid case
    {
      // declarations
      int s = maxValue + 1; // size of array
      boolean[] f = new boolean[s];
      int i;

      // initialize array to true.
      for (i = 0; i < s; i++)
        f[i] = true;

      // get rid of known non-primes
      f[0] = f[1] = false;

      // sieve
      int j;
      for (i = 2; i < Math.sqrt(s) + 1; i++)
      {
        if (f[i]) // if i is uncrossed, cross its multiples.
        {
          for (j = 2 * i; j < s; j += i)
            f[j] = false; // multiple is not prime
        }
      }

      // how many primes are there?
      int count = 0;
      for (i = 0; i < s; i++)
```

```
      {
        if (f[i])
          count++; // bump count.
      }

      int[] primes = new int[count];

      // move the primes into the result
      for (i = 0, j = 0; i < s; i++)
      {
        if (f[i])             // if prime
          primes[j++] = i;
      }

      return primes;  // return the primes
    }
    else // maxValue < 2
      return new int[0]; // return null array if bad input.
  }
}
```

Then I wrote a unit test for `GeneratePrimes`. It is shown in Listing 2. It uses the uses the JUnit framework as Jerry had instructed. It takes a statistical approach; checking to see if the generator can generate primes up to 0, 2, 3, and 100. In the first case there should be no primes. In the second there should be one prime, and it should be 2. In the third there should be two primes and they should be 2 and 3. In the last case there should be 25 primes the last of which is 97. If all these tests pass, then I assumed that the generator was working. I doubt this is foolproof, but I couldn't think of a reasonable scenario where these tests would pass and yet the function would fail.

Listing 2

`TestGeneratePrimes.java`

```
import junit.framework.*;
import java.util.*;

public class TestGeneratePrimes extends TestCase
{
  public static void main(String args[])
  {
      junit.swingui.TestRunner.main(
        new String[] {"TestGeneratePrimes"});
  }
  public TestGeneratePrimes(String name)
  {
    super(name);
  }

  public void testPrimes()
  {
    int[] nullArray = GeneratePrimes.generatePrimes(0);
```

```
        assertEquals(nullArray.length, 0);

        int[] minArray = GeneratePrimes.generatePrimes(2);
        assertEquals(minArray.length, 1);
        assertEquals(minArray[0], 2);

        int[] threeArray = GeneratePrimes.generatePrimes(3);
        assertEquals(threeArray.length, 2);
        assertEquals(threeArray[0], 2);
        assertEquals(threeArray[1], 3);

        int[] centArray = GeneratePrimes.generatePrimes(100);
        assertEquals(centArray.length, 25);
        assertEquals(centArray[24], 97);
    }
}
```

I got all this to work in about an hour. Jerry didnt' want to see me until after lunch, so I spent the rest of my time reading the *Design Patterns* book that Jerry gave me.

After lunch I stopped by Jerry's office, and told him I was done with the program. He looked up at me with a funny grin on his face and said: "Good, lets go check it out."

He took me out into the lab and sat me down in front of a workstation. He sat next to me. He asked me to bring up my program on this machine. So I navigated to my laptop on the network and brought up the source files.

Jerry looked at the two source files for about five minutes. Then he shook his head and said: "You can't show something like this to Mr. C! If I let him see this, he'd probably fire both of us. He's not a very patient man."

I was startled, but managed keep my cool enough to ask: "What's wrong with it?"

Jerry sighed. "Lets walk through this together." he said. "I'll show you, point by point, how Mr. C wants things done."

"It seems pretty clear" he continued, "that the main function wants to be three separate functions. The first initializes all the variables and sets up the sieve. The second actually executes the sieve, the third loads the sieved results into an integer array."

I could see what he meant. There *were* three concepts buried in that function. Still, I didn't see what he wanted me to do about it.

He looked at me for awhile, clearly expecting me to do something. But finally he heaved a sigh, shook his head, and...

To Be Continued Next Month