Jeff Fennell
CMSI 402
3/15/2016

7.2
These comments are highly redundant, and don't provide any significant improvement to the programmer's understanding of the code. This could have been caused by the programmer utilizing a top-down development strategy, or writing the comments after writing the code without considering fully whether they were absolutely necessary.

7.4
An Illegal Argument Exception should be thrown when a or b is 0.

7.5
Yes, you should add error handling to this code. This method would likely be used inside of another method, and you wouldn't want that method to proceed if a valid gcd is not returned.

7.7
1. Start the car.
2. Back out of the driveway.
3. Drive forward
4. Turn left on Manchester Blvd.
5. Drive forward
6. Turn right on Sepulveda Blvd.
7. Turn right into Supermarket parking lot.
8. Turn left into a parking space.
9. Turn the car off.

8.1
```
Public void testRelativelyPrime() {
   assert(!isRelativelyPrime(21, 35);
   assert(isRelativelyPrime(1,24);
   assert(isRelativelyPrime(7,4);
   assert(!isRelativelyPrime(99, 0);
   assert(isRelativelyPrime(0,1);
   assert(isRelativelyPrime(0,-1);
   assert(!isRelativelyPrime(0,0);
}
```

8.3
I used black-box testing. All I knew writing the tests was some of the edge cases that I could test to see if they worked. I had no implementation details of the **isRelativelyPrime** method, however. This obviously can and should be used if you don't have access to the source code, or any idea of how the algorithm is implemented, because the problem provides information on what correct outputs should be. If you had access to the source code, or knew how **isRelativelyPrime** was implemented, you could perform white box/ gray box testing. Exhaustive testing, while possible, would be incredibly inefficient, because it would need to be performed 2 million times, and calculating prime numbers is an expensive computation. It makes more sense to account for possible edge cases, as well as a few different inputs in between, and assume it works for all inputs.

8.5

Jeff Fennell
CMSI 402
3/15/2016

8.9
Exhaustive testing is testing all possible inputs, and knowing what potential inputs are is not possible unless you have access to the source code, so it is a form of white-box testing.

8.11
A good way to figure out the number of bugs would be to calculate the Lincoln Index for each pair of testers.

Alice & Bob: **5 x 4 / 2 = 10**
Bob & Carmen: **4 x 5 / 1 = 20**
Alice & Carmen: **5 x 5 / 2 = 12.5**

It's good to plan for the worst case, so it would be fair to assume there are 20 bugs in the software.

8.12
If there are no bugs in common, it means you are dividing by 0. Obviously this is impossible, so there is no way to estimate the number of bugs using the Lincoln Index in this case.