January 2015 Software Engineering Qualifying Exam

This is an open book exam. Basic calculators are allowed, but no computers or other devices that have communication capability are allowed; this means that cell phones cannot be used as calculators. There are a total of 100 points on this exam. Be sure to show your work in case your answer may deserve partial credit, but do not add spurious or frivolous content in hopes that something you say might be right. Content in an answer that is irrelevant to the problem may cause point deductions.

[50pts] 1. Process Methodologies, Modeling, and Architecture

[10pts] A. *Scrum* is a software development methodology that fits within the *agile methods* context. Explain the main ideas of Scrum and how a Scrum development process is organized and proceeds. Include, but do not limit your answer to, the ideas of backlogs in a Scrum process and how they are utilized.

[20pts] B. Consider that you are building a product that will manage Scrum backlogs (both product and sprint backlogs). Using proper UML class diagram notation, create a domain model for your problem. Note that you are not being provided a problem description—you must use your knowledge of Scrum to understand what the problem domain is, and thus what to include in your model.

[10pts] C. Give a concrete (real) example of a project that would be appropriately developed using Scrum, and explain why you think so. Then give a concrete (real) example of a project that would *not* be appropriate to develop using Scrum, and explain why. Finally, explain what development methodology you *would* use for the second project, and why.

[10pts] D. The county we live in (Doña Ana) has an election website where one can browse dynamic, interactive maps of the county that can show the various voting districts, and one can even enter their own address and see what districts they are in. The maps are dynamic and interactive because there are many overlapping districts for different election purposes (county commission, state representative, state senator, school board, US representative, state PRC, etc.). Would it be appropriate to build this system around a pipe-and-filter architectural style? Explain why or why not, and if your answer is no, also provide an architectural style that would be appropriate to use, and explain why. Finally, if your answer is no, you *cannot* choose client-server as your appropriate architectural style; you must choose some other style.

[50pts] 2. Program Analysis and Verification

The following questions deal with the program shown below, and with the sample executions shown below the program. The class has three different implementations of a collision detection calculation for checking to see if two **circles** are touching or overlapping. The main() method accepts *integer* command line arguments defining the circles, and then invokes all three collision detection methods. Unnumbered lines are simply formatting continuations of the previous numbered line; treat them as part of the numbered line.

Note that the method "inCollision3()" is on the left hand side, while "inCollision1()" and "inCollision2()" are on the right. Note also that when figuring out what any of the methods might produce from any new test cases you create, *be careful* to follow the code exactly; do not assume you "generally" know what it is doing. Bugs may or may not abound!

```
 1:
 2: public class Collide
 3: {
 4:
 5: public static boolean inCollision3(
              int x1, int y1, int r1,
              int x2, int y2, int r2)
 6: {
 7:    int x, y, r;
 8:    double cx, cy;
 9:    double d;
10:    // find line coeff's for line between centers
11:    double m =  (y2 - y1) / (double) (x2 - x1);
12:    double b = y2 - (m * x2);
13:    // choose leftmost center
14:    if (x1 < x2) {
15:       x = x1; y = y1; r = r1;
16:    } else {
17:       x = x2; y = y2; r = r2;
18:    }
19:    // find circle point on line
20:    cx = x; cy = y;
21:    d = 0.0;
22:    while (d < r) {
23:       cx += 0.001;
24:       cy = m*cx + b;
25:       d = Math.sqrt( (cx-x)*(cx-x) +
                         (cy-y)*(cy-y) );
26:    }
27:    // check if point is in other circle
28:    if (x == x1) {
29:       d = Math.sqrt( (cx-x2)*(cx-x2) +
                         (cy-y2)*(cy-y2) );
30:       if (d <= r2)
31:          return true;
32:    } else {
33:       d = Math.sqrt( (cx-x1)*(cx-x1) +
                         (cy-y1)*(cy-y1) );
34:       if (d <= r2)
35:          return true;
36:    }
37:    return false;
38: }
39:
```

```
40: public static boolean inCollision1(int x1, int y1, int r1,
                                       int x2, int y2, int r2)
41: {
42:    if (Math.abs(x1-x2) <= (r1+r2) &&
           Math.abs(y1-y2) <= (r1+r2))
43:       return true;
44:    else
45:       return false;
46: }
47:
48: public static boolean inCollision2(int x1, int y1, int r1,
                                       int x2, int y2, int r2)
49: {
50:    int xd = x2 - x1;
51:    int yd = y2 - y1;
52:    xd = xd * xd;
53:    yd = yd * yd;
54:    int d = (int) Math.sqrt(xd + yd);
55:    if (d <= r1+r2)
56:       return true;
57:    else
58:       return false;
59: }
60:
61: public static void main(String args[])
62: {
63:    int x1, y1, r1, x2, y2, r2;
64:    x1 = Integer.parseInt(args[0]);
65:    y1 = Integer.parseInt(args[1]);
66:    r1 = Integer.parseInt(args[2]);
67:    x2 = Integer.parseInt(args[3]);
68:    y2 = Integer.parseInt(args[4]);
69:    r2 = Integer.parseInt(args[5]);
70:    System.out.println("One: ("+x1+","+y1+") radius="+r1);
71:    System.out.println("Two: ("+x2+","+y2+") radius="+r2);
72:    if (inCollision1(x1,y1,r1,x2,y2,r2))
73:       System.out.println("1: Objects are in collision");
74:    else
75:       System.out.println("1: Objects are NOT in collision");
76:    if (inCollision2(x1,y1,r1,x2,y2,r2))
77:       System.out.println("2: Objects are in collision");
78:    else
79:       System.out.println("2: Objects are NOT in collision");
80:    if (inCollision3(x1,y1,r1,x2,y2,r2))
81:       System.out.println("3: Objects are in collision");
82:    else
83:       System.out.println("3: Objects are NOT in collision");
84: }
85:
86: }
```

Two sample runs of the program are:

```
shell> java Collide 1 2 3 4 3 1
One: (1,2) radius=3
Two: (4,3) radius=1
1: Objects are in collision
2: Objects are in collision
3: Objects are in collision

shell> java Collide 1 2 3 6 3 1
One: (1,2) radius=3
Two: (6,3) radius=1
1: Objects are NOT in collision
2: Objects are NOT in collision
3: Objects are NOT in collision
```

In the sample two runs above, all three implementations produce the correct answer.

[10pts] A. What statement coverage do the two tests achieve in total (i.e., treat them together as one *test suite*) for each of the three *inCollision* methods? You can leave your answer as a ratio of covered line count versus total line count. List any lines not covered for any of the three methods.

[12pts] B. Create a test suite (a set of tests) of *no more than* six tests by using *black box* testing ideas **only**. You may use the existing two tests if desired. For each individual test you create, explain why it is a good choice based on black box testing ideas. Points will be deducted for non-black-box explanations (e.g., using white box testing ideas, ad hoc reasoning, or other) and for creating too many tests. Full black box testing of this problem may indeed require more than six tests, but you are limited in this problem to just six.

[8pts] C. For each test from (B), determine the outcome of **only** "inCollision1()" and "inCollision2()", and state if the outcome is correct or not.

[8pts] D. Would any of your tests from (B) improve the coverage of "inCollision3()"? Explain why or why not. If not, create a test that will.

[6pts] E. If you have found any bugs, explain them.

[6pts] F. Which implementation of the collision detection computation is best, and why?