University of Illinois at Urbana-Champaign  
Department of Computer Science

Final Exam  
CS 427 Software Engineering I  
Fall 2011  
December 13, 2011  
TIME LIMIT = 3 hours  
COVER PAGE + 18 PAGES

Upon receiving your exam, print your name and netid neatly in the space provided below; print your netid in the upper right corner of every page.

Name: ________________________________  
Netid: ________________________________

This is a closed book, closed notes examination. You may not use calculators or any other electronic devices. Any sort of cheating on the examination will result in a zero grade.

We cannot give any clarifications about the exam questions during the test. If you are unsure of the meaning of a specific question, write down your assumptions and proceed to answer the question on that basis.

Answering "I don’t know" (and nothing else) to any exam question is automatically worth 25% partial credit for that question.

Do all the problems in this booklet. Do your work inside this booklet, using the back of pages if needed. The problems are of varying degrees of difficulty so please pace yourself carefully, and answer the questions in the order which best suits you. Answers to essay-type questions should be as brief as possible. If the grader cannot understand your handwriting you may get 0 points.

There are 33 questions on this exam and the maximum grade on this exam is 149 points.

<table>
<thead>
<tr>
<th>Page</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>78</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>71</td>
<td></td>
</tr>
</tbody>
</table>
Software metrics

1. (a) A good software design has (circle one)
   
   i. High Cohesion, High Coupling  
   ii. High Cohesion, Low Coupling  
   iii. Low Cohesion, High Coupling  
   iv. Low Cohesion, Low Coupling

   (b) With regards to coupling or cohesion, what does each of the code smells below indicate (e.g., high cohesion)?
   
   i. A single class contains many unrelated functionalities
   
   ii. A method in ClassA seems way too interested in the workings and data fields of ClassB

2. FURPS is an acronym representing a model for classifying software quality attributes, and it stands for Functionality, Usability, Reliability, Performance, and Supportability. For each of the FURPS attributes write one sentence that relates it to your CS427 refactoring project.

3. (a) A system with 100% test coverage does not mean that the system is 100% correct. Explain why.

   (b) Two teams tested the same system independently. One wrote 1,000 tests and the other wrote 200 tests. Did the team who wrote more tests necessarily did a better job than the other team? Explain why.
4. Suppose that the project grade in CS427 will be based on the lines of code (LOC) software metric. Do you think that this grading scheme is a good idea? Describe one advantage and one disadvantage of using lines of codes to measure productivity.

5. Product metrics fall into two classes: dynamic metrics and static metrics. Give two examples of dynamic metrics and two examples of static metrics.

6. Answer the following questions about the software metric called Function Points.
   (a) Briefly describe what Function Points are.
   (b) What factors are used to compute Function Points?
   (c) How can Function Points be used in software project management?
Software Configuration Management

7. Write the letter of the term that best matches each description. Each term may be used at most once, some will not be used.

Terms
A. audit and review  D. change control  G. mainline  J. release
B. automated build  E. change request  H. private system build  K. status accounting
C. baseline  F. identification  I. private workspace  L. version control

Descriptions
(a) A formal process to ensure that changes to the system are introduced in a controlled and coordinated manner.

(b) Assigns revision numbers for each change to a file and permits branching and merging.

(c) Involves the compilation and linking of source code files in the correct order a software configuration item that has been reviewed and agreed upon, and that can be changed only through formal change control procedures.

(d) Done before committing changes to the Repository to avoid breaking the build.

(e) A software configuration item that the developers give to other people.

Debugging

8. You have been using Eclipse extensively during this course. Eclipse provides many great debugging features that help you when you write your code. Give two examples of Eclipse debugging features and explain why they are useful.
9. State the precondition, postcondition, and the loop invariant for the following code.

```java
/**
 * @param a[] an array of integers that may or may not contain val
 * @param val the integer value we want to find in a[]
 * @return -1 if a[] does not contain val;
 * index if a[] contains val.
 */
int binarySearch(int a[], int val) {
    int lower = 0;
    int upper = a.length - 1;

    while (lower <= upper) {
        int i = lower + (upper - lower) / 2;
        if (val == a[i]) {
            return i;
        } else if (val < a[i]) {
            upper = i - 1;
        } else {
            /* val > a[i] */
            lower = i + 1;
        }
    }

    return -1;
}
```
Design Patterns and Refactoring

10. (a) Refactor the following classes to use the **Composite** pattern, and (b) **name** 2 kinds of refactoring you used.

```java
public class Mesh {
    private Set<Triangle> triangles = new HashSet<Triangle>();
    public Mesh() {
        // code to generate a mesh
    }
    public void drawMesh() {
        for (Triangle t : triangles) {
            t.drawTriangle();
        }
    }
}

public class Triangle {
    private Line line1, line2, line3;
    public Triangle(Line line1, Line line2, Line line3) {
        this.line1 = line1;
        this.line2 = line2;
        this.line3 = line3;
    }
    public void drawTriangle() {
        line1.drawLine();
        line2.drawLine();
        line3.drawLine();
    }
}

public class Line {
    public Point p1, p2;
    public Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
    public void drawLine() {
        Graphics g = new Graphics();
        g.drawLine(p1.x, p1.y, p2.x, p2.y);
    }
}
```
continue your answer for question 10...

(a) What design pattern would you use to make it easy to change the kind and number of objects that react to changes in another object?

(b) What design pattern would you use to make it easy to change the algorithm that is being applied to a tree of objects, where the nodes in a tree are in many different classes?

(c) What design pattern should you think of first when you want to implement undo?

(d) Which pattern do you think about when a design requires a tree of objects?

(e) What design pattern should you think of when you want to reuse an object but it has the wrong interface?

(f) Which pattern do you think about when trying to use different algorithms for solving the same problem?

(g) Which pattern do you think about when you want to evaluate a sentence in a particular language?

(h) Which pattern do you use when you want to write a code skeleton of an algorithm?

(i) Which pattern would you use to hide the complexity of a legacy system?

(j) Which pattern would you use to traverse a collection of objects?
12. Briefly describe the **Visitor** pattern and give one advantage and one disadvantage of using it.

13. Pick either **Visitor** pattern or **Template Method** pattern, and write skeleton code that illustrates the essence of the pattern.
14. Write the interfaces that the participants in the **Observer** pattern must implement.

15. Identify a smell in the code below, and suggest a refactoring to remove the code smell.

```java
public class Line {
    private Point start, end;
    public void setStart(int x) {
        this.start = x;
    }
    public void setEnd(int y) {
        this.end = y;
    }
    public Point getStart() {
        return start;
    }
    public Point getEnd() {
        return end;
    }
}

public class Utilities {
    public double getLength(Line line) {
        int x1 = line.getStart().getX();
        int x2 = line.getEnd().getX();
        int y1 = line.getStart().getY();
        int y2 = line.getEnd().getY();
        return Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))
    }
}
```
16. Give two usage examples of the **Command** pattern.

17. Consider the following classes. Each of these classes implements a certain design pattern. Name each pattern by filling in the blank line. Briefly describe why you chose the answer in each case.

```java
public class YelpAPIService {
    public Collection getRestaurants (double lat, double lon, double radius) {
        ...
    }
    public String[] getReviews (int restaurantID) {
        ...
    }
    public void postReview (int restaurantID, String review) {
        ...
    }
}
```

```java
public class AddressToCoordinates {
    public static double[] getCoordinates (String address) {
        ...
    }
}
```

```java
public class GUIController implements Listener {
    public void buttonPressed (Event e) {
        String address = e.getField().value();
        double[] coords = AddressToCoordinates.getCoordinates(address);
        Collection searchResults = YelpAPIService.getInstance().getRestaurants(coords[0], coords[1], 2);
        // show searchResults;
    }
}
```
18. Briefly describe the non-localized plan (a.k.a. shotgun surgery) code smell. What refactoring can be applied to remove this smell?

19. Remove Parameter Refactoring removes unused parameters of a method. Describe two scenarios for the refactoring. One scenario should illustrate when the refactoring preconditions are not met (thus the refactoring cannot proceed) while the other scenario should illustrate when the refactoring is allowed to proceed.
Reverse Engineering and Reengineering

20. In the class project for CS427, you had the opportunity to apply some of the reverse engineering patterns described in the book *Object-Oriented Reengineering Patterns*. Choose one reverse engineering pattern that your team used and describe how your team applied the pattern. If your team didn’t use any pattern, then you should pick any one from the reading.

21. (a) For each scenario below, name and describe a software engineering technique you could use to solve the problem.

i. As a new software developer for company Valve, your first task is to add functionality to an existing piece of software that has limited documentation. The developer who was responsible for maintaining the software has left Valve a few months ago.

ii. After working for Valve for a few months, you are given a more complex piece of software to work on. You have a difficult time understanding the functionality of each method because most methods are long and have non-descriptive names. You are allowed to make changes to the existing code if necessary.

(b) Based on scenarios (i) and (ii) above, describe one difference between refactoring and reverse engineering.
Refactoring Project

22. In the CS427 refactoring project, you implemented a new refactoring in Eclipse. Most of the implementation of your new refactoring is distributed among three steps in the Refactoring class. Briefly describe two of the steps in the Refactoring class.

23. The following are the 3 pillars of Eclipse refactoring engines. Briefly describe the functionality for each pillar.

   (a) Java Model (IJavaElement):

   (b) Search Engine:

   (c) AST:
Frameworks and Libraries

24. The game company Zumba wants to develop a new browser-based game. The options are to either implement the game from scratch or reuse a game engine developed by Zynga. Name two major costs that Zumba must take into account when making this decision?

25. Name two product-line software and briefly explain why they are considered product lines.

26. What is the difference between a framework and a library?

27. Give one example of a framework-specific feature of Eclipse that you used while implementing your class project.
XP and Software Development Process

28. Indicate whether each of the following statements about XP practices is true or false.
   (a) When done properly, code refactoring improves some aspects of a program, such as read-
       ability and maintainability, but does not change the functional behavior of the program.

   (b) User stories are brief descriptions of requirements written by the developers in their lan-
       guage.

   (c) Test-driven development involves writing tests that define requirements before writing the
code.

   (d) In XP any developer can change any part of the code base.

   (e) The development team should always be working on the latest version of the software.

29. Describe the relationship between design and refactoring in XP.

30. Software development can be thought of as a spiral/iterative process. Draw a diagram and
    briefly describe each phase of the process.
31. (a) According to Parnas and Clements, what is a **Rational process**?

(b) Give two reasons why this might be impossible.

(c) In the context of the rational process, what is meant by **you should fake it**?

32. (a) Give two reasons why a project needs documentation.

(b) Give two examples of how you test software documentation.
Testing

33. Consider the course management system below. Write a JUnit test class with at least four tests for the method `getCoursesForStudent()`. The scenarios covered by your tests should be varied.

```java
class Department {
    private Set<Course> courses;
    private Set<Student> students;

    public void addCourse(Course c) { courses.add(c); }
    public void removeCourse(Course c) { courses.remove(c); }
    public void addStudent(Student s) { students.add(s); }
    public void removeStudent(Student s) { students.remove(s); }

    public Set<Course> getCoursesForStudent(Student student) {
        Set<Course> studentCourses = new HashSet<Course>();
        for(Course c : courses) {
            if(c.hasStudent(student))
                studentCourses.add(student);
        }
        return studentCourses;
    }
}

class Student {
    private Department department;
    private String name;
    public Student(Department department, String name) {
        this.department = department;
        this.name = name;
    }
    // well-implemented hash and equals methods
}

class Course {
    private Department department;
    private Set<Student> students;

    public Course(Department department, String name) {
        this.department = department;
        this.name = name;
    }

    public addStudent(Student student) { students.add(student); }
    public removeStudent(Student student) { students.remove(student); }
    // well-implemented hash and equals methods

    public boolean hasStudent(Student student) {
        return students.contains(student);
    }
}
```
JUnit test class: