University of Illinois at Urbana-Champaign
Department of Computer Science

Final Exam

CS 427 Software Engineering I
Fall 2010
December 15, 2010
TIME LIMIT = 3hrs
COVER PAGE + 16 PAGES

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 form 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.

Do all the problems in this booklet. Do your work inside this booklet, using the backs 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 will get 0 points.

There are 8 questions on this exam and the maximum grade is **100 points**

Page	Points	Score	Page	Points	Score
2	8		**10**	6	
3	5		**11**	6	
4	8		**12**	6	
5	3		**13**	8	
6	10		**14**	7	
7	10		**15**	8	
8	2		**16**	4	
9	9				
Total	**56**		**Total**	**44**	
Final Total				**100**	

1. **Software Quality**
    a. List the three major parts of a quality assurance plan and choose any **one (1)** and explain why it is important? **[4 points]**

    b. Number of tests can also be used as software metric—do a large number of tests directly translate to better quality software? Why or why not? Give **two** reasons to support your answer. **[2 points]**

    ```
    Yes

    Large number of tests means high rate of code coverage

    1. Identify undested part of codebase
    2. Improve the quality by improved test coverage
    3. Identify testing gaps or missing tests
    4. Identify the redundant/dead code
    ```

    c. With FURPS, the "URPS" categories describe non-functional requirements that are generally architecturally significant. Briefly describe what each of the following is concerned with: **[2 points]**
        i. Usability

        ii. Reliability

d. Supportability is a key component of FURPS; describe two metrics that can be used for supportability. **[2 points]**

2. **Software Reuse**

a. During system development it is sometimes beneficial to reuse Commercial Off-The-Shelf (COTS) systems. Describe **TWO** benefits that can be derived from reusing COTS. **[2 points]**

1. Increased reliability
2. Reduce process risk
3. Effective use of specialists
4. Standard compliance
5. Accelerated Deployment

b. Your boss has been introduced to a new concept called frameworks; explain to him briefly what a framework is? **[1 point]**

Framework is sub-system design containing a collection of abstract and concrete class along with interface between each class

Framework is reusable entities

c. Describe **TWO** problems that may occur with software reuse? **[2 points]**

1. Lack of tool support
2. Pervasiveness of the "not invented here" syndrome
3. Need to create and maintain a component library
4. Finding and adapting reusable components

3. **Metrics**

   a. Explain why software metrics are more useful as indicators of what's wrong rather than what's right about a system. **[2 points]**

   b. Select two of the following metrics and describe (i) what the metric measures and (ii) how the metric can be used to improve software. a) Code Coverage, b) Cyclomatic Complexity, c) Weighted methods per class. **[4 points]**

```
Code Coverage
- Measure: How many parts of code are coverage by the tests
- Improve software:
  - Identify the untested part which will lead high risk
  - Identify testing gaps or missing test

Cyclomatic Complexity
- Measure: Branch rate of one method(if, while, for),
  measure the complexity of one specific method
- Improve software:
  - Point out those complex method, the more complex the
    method is, the higher risk the code will perform bugs
  - Indicates that the method should be refactor
```

c. In OO Design Quality Metrics, Robert Martin describes a set of metrics that can be used to measure the quality of an object-oriented design in terms of the interdependence between the subsystems of that design

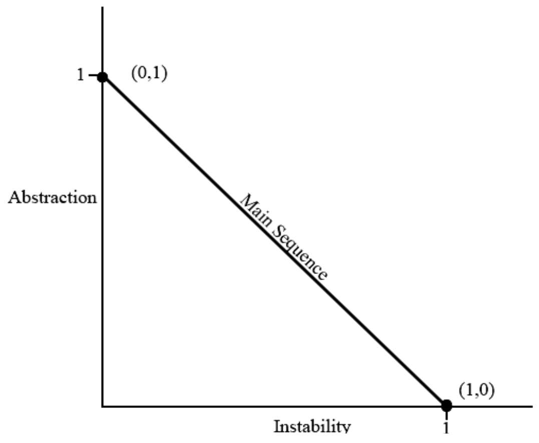
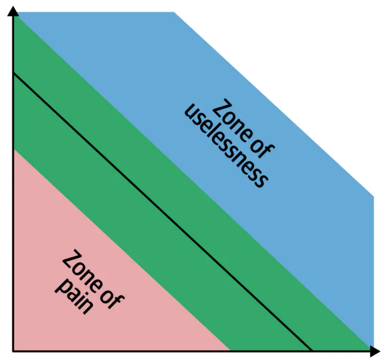


Figure 1

i. Describe the category at position (0, 0) in Figure 1 in terms of stability and abstraction. Why is such a category not desired? **[2 points]**



Zone of pain

Too many implement and less abstractness makes project hard to maintained

Zone of uselessness

Too abstract ness make code hard to use

ii. Why is it desirable for a category to be as close as possible to the main sequence shown in Figure 1? **[1 point]**

Because, if you are in the main sequence, that means the higher abstractness you get, the lower instability you get, so as the reverse. This means you need to write the code focus on either abstraction or the instability, or balance each one. It will make the code more reliable and usable

4. **Design Patterns**

   a. For each of the following 10 statements, choose the pattern from the list below that most closely matches it. Write the appropriate letter for each response. **[10 points]**
   A. Composite, B. interpreter, C. visitor, D. template method, E. observer, F. strategy, G. command, H. adapter, I. facade, and J. iterator

Description	**Answer**
i. Consider this pattern when you want to treat a group of objects the same as a single object in the group.	
ii. Consider this pattern when you want to remember an operation and possibly undo it.	G
iii. Consider this pattern when you want to make a little language for your tool.	
iv. Consider this pattern when you want to make it easy to define a new algorithm that works on a collection of objects of varying classes, and the algorithm treats different kinds of objects differently.	
v. Consider this pattern when you'd like to have a set of possible algorithms that an object could use for a particular purpose, and you'd like it to be fairly easy to add a new one.	F
vi. Consider this pattern when you have a number of different classes that all have the same operation, and this operation tends to be mostly the same but usually varies just a little from one class to the next.	
vii. Consider this pattern when you want to reuse an object, but its interface is not quite right.	
viii. Consider this pattern when you want to look at each element in collection of objects but you don't want to depend on how that collection is implemented.	
ix. Consider this pattern when you want to hide the complexity of a large subsystem	I
x. Consider this pattern when you want to allow some objects to change whenever another object changes, but you want to make it easy to change the number of objects that depend on it.	

b. Suppose that classes **AssignmentNode** and **LoopNode** are both subclasses of **StatementNode,** and they have the following two methods, each called **emitReturn**. Change this code to use the **Template Method** pattern. **[10 points]**
(If answer cannot hold on this page, continue on next page)

```
public class AssignmentNode extends StatementNode {
  private Node variable;
  private Node value;

  public void emitReturn(CodeStream output, Node returnNode) {
        variable.emitStore(output, value, this);
        output.noteSource(returnNode);
        output.generateReturn();
  }
}
```

```
public class LoopNode extends StatementNode {
  public void emitReturn(CodeStream output, Node returnNode) {
        emitEffect(output);
        output.pushConstant(nil);
        output.noteSource(returnNode);
        output.generateReturn();
  }
}
```

c. Name two refactorings that you (could have) used when you refactored it to use Template Method? **[2 points]**

5. **Refactoring and Code Smells**
    a. Select one of the following code smells and describe how to identify it: **[2 points]**
        i. Non-localized plan   ii. Refused Bequest   iii. Data class.

    b. Describe the refactoring steps for eliminating the selected code smell from part (a) **[2 points]**

    c. Even if your code is ugly and smelly, give a situation when it should **not** be refactored? **[1 point]**

    d. Give two (2) specific reasons when you should refactor. **[2 points]**

e. Suppose you perform a refactoring and then a test fails, when should you modify the test and when should you **not** modify the test? **[2 points]**

6. **Testing and Documentation**
   a. Explain briefly **two (2)** ways for testing documentation such as a user manual? **[2 points]**

   b. Automated testing is an important aspect of software development:
      i. You were exposed to the automated testing framework JUnit during the class; give two features that such a framework offers to developers to make testing more effective. **[2 points]**

1. Sharing common test data among tests
2. Test suites for easily organizing and running tests
3. Display statistic results of tests

ii. Your supervisor read a section of Brian Marrick’s article which stated that “attempting to automate all tests is a bad idea”, explain to your supervisor why this is so. **[2 points]**

c. The assigned reading included Brian Marrick's catalog for testing. Describe where and how you used one idea from the catalog in your own project. **[2 points]**

d. In MP5, you have used EclEmma to measure coverage on several Photran tests. Is it always possible to achieve 100% code coverage using EclEmma? If not, describe a situation where 100% code coverage cannot be achieved. **[2 marks]**

e. What is an equivalence partition? Give an example. **[2 points]**

f. Write **2** *JUnit* test cases for the ***factorial****(int n)* method below. **The first test case should show the bug in the method**. You can use the testing conventions for JUnit3.x or JUnit4.x **[6 points].** *(Use the extra page at the back if you need more space)*

```
public int factorial(int n)
{
       if (n <= 1) {
        return 0;
    }
    else {
        return n * factorial(n-1);
    }
}
```

```
@Test
public void testFactorial(){
    int number = 3;
    int factorialNum = factorial(number);
    assertEquals(6, factorialNum);
}


@Test
public void testFactorial2(){
    int number = -1;
    int factorialNum = factorial(number);
    assertEquals(0, factorialNum);
}
```

7. **Software Development Project**
    a. Explain why it is important to have a basic design before you can make a schedule for a software project. **[2 points]**

    b. List two things that should be included in a vision statement for a software development project? **[2 points]**

    c. According to Parnas and Clements, what is a “Rational process”? **[2 points]**

    d. Give two reasons why this might be impossible. **[2 points]**

e. In the context of the rational process, what is meant by “but you should fake it”? **[2 points]**

f. Give two (2) reasons why change control is important in a software development project? **[2 points]**

Help team manage code conflict

Help identify which file should track the change and which is not important

8. **Miscellaneous**

a. Give specific examples of how your Final Project group followed these two XP practices: planning game and test-driven development. **[2 points]**

b. XP says that you do not necessarily have to write documentation, give one example when you do need to write documentation. **[1 point]**

You are developing software reuse component or SDK that will be used by other person

c. Suppose that you already checked out your project from the *trunk* directory. Your team-mates (i) checked out the same revision, (ii) changed some files locally, and (iii) committed the changes to the repository. What operations do you need to perform to obtain the latest version from trunk? **[2 points]**

fetch & merge?

d. In your class project you worked on different refactorings for Photran, what are the four (4) steps in which all refactorings must proceed? **[4 points]**

e. Several reverse engineering patterns were outlined in the paper, *“A pattern language for reverse engineering”*; describe **one** that was used during your final project, to better understand the Photran system. If you did not use any of the patterns presented, describe any **two** that can help during reverse engineering. **[2 points]**

f. Based on your answer from **part d**, state the problem that this pattern addresses *(If you described* ***two****, simply choose one)* **[1 point]**

g. Summarize the proposed solution (one or two sentences). **[1 point]**

h. Describe how this pattern reduces the risk for one of the five risk factors listed by Demeyer, Ducasse, and Nierstrasz. **[2 points]**

Extra Page