Write your name and netid neatly in the space provided below; write 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.

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 12 questions on this exam and the maximum grade on this exam is 55 points.

<table>
<thead>
<tr>
<th>Page</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>
1. **eXtreme Programming (XP)**

(a) If you could choose one core XP practice for a “waterfall” team to adopt, what practice would you choose and why?

**Solution:**

Example practices: test-driven development, planning game, user stories, pair programming, refactoring, simplicity, onsite customer

Example explanations: All the XP practices that can help provide faster feedback in waterfall are good answers.

Test-driven development make requirements and design clear. Developers know quickly if their code meets requirements or not. Pair-programming helps spotting errors quicker, plus developers learn from each other. Iterations allow each step in waterfall to be changed. Customers on site allow developers to prioritize their work instead of spending too much time on requirements that are not high priority. ...

(b) Describe the relationship between automated unit tests and refactorings in XP.

**Solution:**

The key idea is, you can easily run automated tests to check if your refactoring has broken some functionality. This fast feedback compared to manual tests allows you to make changes more confidently and quickly.

2. **Software Configuration Management (SCM):** Name three artifacts besides source code that are part of a software project that you might want to keep under SCM. For each artifact that you name, describe why it is good to keep it under SCM.

**Solution:**

The main thing this question is asking for is “artifacts”.

Example artifacts: Test suites, so we can test our code. Manuals, so they can be given to end users. Requirements, so we have a record in case they change over time. Design documentation, so future developers may use it to help understand the system. Build configuration, so we can reproduce the builds in a consistent way.

The common mistake students make is discussing the aspects of SCM (such as change control, version control, building, releasing) or discuss what information those artifacts keep track of (such as version of code).
3. Version Control: MPs ask you to tag your code. In general software development, not just MPs, what do tags help with?

Solution:
Tags are markers to highlight notable revisions in the history of the repository. They can be used to provide stable and functional points during the software development, where we can easily revert back if the code becomes unstable.

Common Mistakes: 1. Tags allows parallel development/experimentation without affecting the mainline. This is the property of svn branches. 2. Tags names/Commit messages are better to recall what had been done in past. 3. Tags force the developer to come up with meaningful commit messages. 4. To figure out which feature is implemented by whom.

4. Reverse Engineering

(a) List two reasons for software system reengineering according to Chapter 1 of the book Object Oriented Reengineering Patterns (OORP).

Solution:
1. unbundle a monolithic system so as to sell/deploy its parts separately 2. improve performance 3. port to a new platform 4. extract the design 5. exploit new technology 6. reduce human dependencies by documenting knowledge about the system

Common Mistakes: 1. To fix bugs. 2. Even after fixing the bugs, they keep on appearing. 3. When there are not much testcases to test the code.

(b) During MP3, you were provided with some HTML files, which we generated using javadoc, to help you get a quick overview of an unfamiliar codebase. This follows a reverse engineering pattern that we discussed in class. This pattern is part of a group of patterns called First Contact in Chapter 1 of the book Object Oriented Reengineering Patterns (OORP). Name this pattern.

Solution:
Skim the Documentation

Common Mistakes: 1. Read/Skim through the code. 2. Use Browsers to understand the class hierarchy.
5. **Code Smells**: One of the code smells you learned in this class is called “non-localized plan”. Describe this code smell in **one** sentence.

**Solution:**
If adding a feature requires changing many parts of a program, it is a “non-localized plan”. Common Mistakes: 1. To move the methods to classes which they are more related to. 2. Excessive use of global variables. 3. When there is no centralized plan to follow and this leads to misinterpretations of plans. 4. Having highly complex code. 5. Referring methods/variables of other classes due to poorly planned location. 6. Lack of code coverage. 7. To have a distributed plan.

6. **More XP (Job Interview Style Questions)**
You are interviewing for a job, and they learn that you took CS 427.

(a) **Describe two benefits and one disadvantage** of pair programming (in a professional software development setting, not in the classroom setting where a disadvantage can be “my teammate doesn’t want to meet”). Provide specific examples from your MPs (e.g., from MP1 on Java/JUnit or from MP2 on CodingTracker/Jenkins). Please also write your group number and/or teammate’s name/netid; if your pair changed, pick one MP where you did the most pairing.

**Solution:**
Benefits: - developers learn from each other, e.g., new command line tricks - bugs can be spotted earlier - develops rapport between team members Disadvantage: - for simple tasks, pair programming might waste an extra developer’s time, since there’s nothing new to learn and there’s no need for an extra person to debug code

(b) Because you followed XP, they ask you to **describe two XP practices (besides pair programming) that you believe can be good** (name the practice, describe what it is, and state why you find it good) and **one XP practice (besides pair programming) that you believe can be bad** (name the practice, describe what it is, and state why you find it bad).

**Solution:**
The answers should be the names of the xp practices:
* pair programming
* planning game
* test driven development
* whole team
* continuous integration
* design improvement
* small releases
* coding standard
* collective code ownership
* simple design
* system metaphor
* sustainable pace
As long as students can:
1. name it
2. define it (even with own words)
3. give a good reason, they should get full points.
3. Because you followed XP and learned about Waterfall, they ask you to describe some third software development process — name one process model different from XP and Waterfall, and briefly describe how two of its practices differ or match XP and/or Waterfall. (Hint: http://en.wikipedia.org/wiki/Software_development_process lists several types of software development processes.)

**Solution:**

spiral model

1. iteration: Waterfall model doesn’t have iteration for development, and XP’s iteration is smaller, shorter.
2. steps in each iteration: determine objectives, identify and resolve risks, development and test, plan next iteration.
3. Waterfall’s steps: requirement, design, implement, verification, maintenance
4. XP’s steps: release plan, iteration plan, acceptance test, stand up meeting, pair negotiation, unit test, pair programming.
   
   Other possible answer: iterative and incremental development, agile development, rapid application development, code and fix.

7. Metrics

2. Many metrics, such as code size or complexity, can be hard to compare across different software projects, but they can be used to compare (1) different modules within one version of the same project or (2) different versions of the same project. Describe two ways that managers/developers/testers can use comparisons of metrics on the same project to decide how to prioritize their work.

**Solution:**

1. Decide where to put more resources into testing.
2. Decide where to refactor.
3. Decide where to do more code reviews.
4. Decide if the size/complexity increase over time is warranted by the features that were added.

2. The lecture on OO metrics discussed coupling. Describe two reasons why high coupling among software components is bad.

**Solution:**

1. High coupling makes designs hard to change.
2. High coupling makes classes hard to reuse.
3. High coupling makes classes hard to test.
8. **Eclipse, Jenkins, and CodingTracker**

(a) A software development company develops an email client. However, when a user tries to send an email with a .PDF attachment, the program crashes. The user complains that the program is “buggy”, but “buggy” can mean many things. The words “fault” and “failure” are much more precise. In the given scenario, where is the fault and what is the failure?

**Solution:**
The key to this question is answering “where” the fault is, and “what” the failure is. The fault is in the source code or implementation, specifically the part that caused the program to crash. This might be the part that handles PDF attachments or the part that sends attachments over the network, etc. The failure is something that the fault caused and is observable to the user of the program. In this case, the user sees the program crash, so the program crashing is the failure.

(b) Some developers prefer to use text editors (e.g., notepad, gedit) to perform basic operations, such as adding, deleting, copying, and pasting code. On the contrary, some developers prefer to use Integrated Development Environments (IDEs) like Eclipse that, in addition to the basic operations mentioned, comes with extra features that can potentially help them become more productive when writing code. Describe two benefits and one disadvantage of using IDEs rather than text editors when developing software.

**Solution:**
Benefits - IDEs can make refactoring less error-prone. - IDEs can make it convenient to interact with version control systems. - IDEs can make it easier to navigate code. - IDEs can make it easier to see the whole project overview. - IDEs can make it easier to format code. - IDEs can make it easier to lookup APIs (e.g. auto-complete) - IDEs can come with built-in tools that are useful, such as debuggers. Disadvantage - You may not have IDE on all platforms. - Sometimes, you have less control of what happens to your code, e.g., when using Eclipse refactorings to move methods, sometimes an unnecessary method argument is added. - Sometimes, IDEs perform unnecessary computations, e.g., automatically build large projects when you’re still modifying code. - IDEs take up more resources. - IDEs can be expensive (e.g. Visual Studio). - IDEs may clutter your screen with irrelevant windows/panes/tabs. - IDEs tend to be tied to a particular language. - IDEs may require more complicated installation, configuration and take time to learn.

(c) Why is it good to explicitly specify JAVA_HOME for Jenkins instead of using the system default?

**Solution:**
The key point is that JAVA_HOME allows us to specify the exact JDK we need. This not only helps locating the JDK in case it is in a non-standard location, but more importantly, if there are multiple JDKs on the system, specifying JAVA_HOME will let Jenkins know which version to use.
2. (d) How does CodingTracker replay code changes differently when clicking on the **Step** button vs. the **Fast** button?

**Solution:**
The speed and continuity is different. When I click the Step button, CodingTracker replays my code changes step by step. When I click the Fast button, CodingTracker replays through all my code changes as fast as the machine allows.

9. **Testing Techniques**

3. (a) In the lecture on Testing and XP, we discussed the benefits of writing the tests before the code. Describe any **three benefits**.

**Solution:**
- Developers know when they are "done" with regards to software requirements, i.e., passing all unit tests. - In case of libraries, it gives examples how to use the API, or how classes are created, etc. - It avoids unnecessary complexity, i.e. creating the simplest code that passes the test. - Writing test cases before writing the code doesn’t take any more effort than writing test cases after the code; it simply resequenced the test-case -writing activity. - When you write test cases first, you detect defects earlier and you can correct them more easily. - Writing test cases first forces you to think at least a little bit about the requirements and design before - writing code, which tends to produce better code. - Writing test cases first exposes requirements problems sooner, before the code is written, because it’s hard to write a test case for a poor requirement. - If you save your test cases (which you should), you can still test last, in addition to testing first.

Common Mistakes: Most of them got it right. But some of them are too vague/ illegible that I end up deducting marks. Some examples are: Forces small iteration (of what !!!!) Can implement only those features that customer wants Programmers are lazy to write test cases after implementation.
(b) What is Equivalence Partitioning in testing?

Solution:
The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same w.r.t the way system handles them (for example, by flushing out exactly the same errors for a particular set of test cases). In equivalence-partitioning technique we need to test only one representative condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition. In other words, this technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases required.

Common Mistakes: 1. Partitioning the test cases so that all the testcases are covered or the code coverage becomes maximum. But how this is achieved in Equivalence Partitioning is not mentioned. 2. Dividing the classes in subclasses. 3. Running tests on partitions of input data. 4. Partitioning testcases into useful sets. 5. To test boundary conditions. 6. Testing all possible paths of the program.

(c) What is Boundary Value Analysis in software testing? Suppose a program validates a numeric field as follows: values less than 10 are rejected, values between 10 and 20 are accepted, and values above 20 are rejected; give two examples of inputs (numeric values) which are good candidates to test boundary values for this case.

Solution:
Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values.

Any 2 numbers in the set \{9, 10, 11, 19, 20, 21, INT_MAX, INT_MIN\}
10. **Refactoring:** In a lecture, we discussed that refactoring is sometimes done “secretly”. What does it mean: **from whom (apart from the users)** do the developers hide that they do refactoring, and **why** do they hide it?

**Solution:**

They can hide from managers, who may think that it is a waste of time. They can hide it from other developers to avoid conflicts. They can hide it from themselves if they are doing refactorings without knowing it. (Any one would get full points.)

**Common mistakes:** hiding from customers/users, hiding from other classes

11. **Tests and Smells**

(a) The test code below has two “smells”: (1) there’s duplicated code between the two methods, and (2) the test result depends on the order in which the tests are executed (`saveLibrary` must be executed before `loadLibrary`), while JUnit does not necessarily execute the tests in the order in which they are written. Rewrite the code such that you still have two tests with the same logic, but (1) there’s no duplication (hint: consider a helper method or `@Before`) and (2) the test result does not depend on the order in which the tests are executed. (Your solution should not duplicate the common code in both tests but should have only one copy of the common code.)

**NOTE:** Write your solution on the next page. When you write code for your solution, you need not copy comments.

```java
public class LibraryTest {
    @Test
    public void saveLibrary () throws Exception {
        Book book = new Book("author", "title");
        // create library with one book
        Library library = new Library(book);
        library.saveLibraryToFile("library.txt");
        assert...
        // content of the file library.txt
    }

    @Test
    public void loadLibrary () throws Exception {
        // create a new library from a given file
        Library loadedLibrary = new Library("library.txt");
        // create library with one book
        Library library = new Library(new Book("author", "title");
        assertEquals(library, loadedLibrary);
    }
}
```

Listing 1: Duplicated Code and Order-Dependent Tests
public class LibraryTest {
    Library library;
    Book book;

    @Before
    void createLibrary() throws Exception {
        book = new Book("author", "title");
        library = new Library(book); // create library with one book
        library.saveLibraryToFile("library.txt");
    }

    @Test
    public void saveLibrary() throws Exception {
        assert...
        // content of the file library.txt
    }

    @Test
    public void loadLibrary() throws Exception {
        // create a new library from a given file
        Library loadedLibrary = new Library("library.txt");
        assertEquals(library, loadedLibrary);
    }
}
12. **Test Selection**

The lecture on testing at Google discussed how one can do test selection in regression testing based on what parts of the changed code the tests executed in the past. Consider (I) the test code originally given in Listing 1 of Question 11 and (II) the test code you wrote in your answer to Question 11. Suppose that you run both tests from each code and find that both tests pass. You then make a change to the code under test and want to rerun these tests for regression testing. Consider these scenarios:

(a) if you change something in the body of the constructor `Library(String fileName)`:

i. What test(s) should be selected to check the changes in the constructor’s method body if the tests are as in the original code from Listing 1 in Question 11? (If you think that no tests should be selected, please write “no tests to be run”)

**Solution:**

Any answer that contains `loadLibrary`.

ii. What test(s) should be selected to check the changes in the constructor’s method body if the tests are as in the code that you wrote in your answer to Question 11? (If you think that no tests should be selected, please write “no tests to be run”)

**Solution:**

Any answer that contains `loadLibrary`.

(b) if you change something in body of the constructor `Library(Book book)`:

i. What test(s) should be selected to check the changes in the constructor’s method body if the tests are as in original code from Listing 1 in Question 11? (If you think that no tests should be selected, please write “no tests to be run”)

**Solution:**

Either or both of `loadLibrary` and `saveLibrary`.

ii. What test(s) should be selected to check the changes in the constructor’s method body if the tests are as in the code that you wrote in your answer to Question 11? (If you think that no tests should be selected, please write “no tests to be run”)

**Solution:**

Either or both of `loadLibrary` and `saveLibrary`. 

---

Page 10 of 10