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.

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 9 questions on this exam and the maximum grade on this exam is 80 + 3 (bonus) points.

<table>
<thead>
<tr>
<th>Page</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
1. Software Configuration Management (SCM)

(a) Your MPs used trunk and tags folders in SVN, but there is also branches. What are branches in version control? Give one good and one bad reason to branch a project.

Solution:
Branches are deviation from the main line of development (trunk) and create parallel lines of development. Some good reasons to branch a project are fixing a bug in deployed code, or implementing an experimental feature to an existing project or for political reasons. Bad reasons to create a branch are creating customized version for a different hardware or for a different customer.
A common mistake is to describe disadvantage of creating new branches, instead of a bad reason to create a new branch. An example of a disadvantage is that branches may need to be merged into the main line, and this process could be time-consuming and tedious.

(b) Eclipse has automated, regularly-occurring builds. Discuss two benefits of such builds.

Solution:
 Automated, regularly-occurring builds save the time of manually building code after modification as well as the time to manually create and maintain build scripts/make files. It also saves debugging time by providing real-time feedback to the programmer, especially if the most recent change in the code introduced build errors.

(c) Give an example of a tool for tracking bug reports, and describe two parts of a bug report.

Solution:
Bugzilla, Trac, and Jira are some examples of bug tracking tools. Common parts of a bug report are an identifier, reporting person, description, severity, status, assigned developer, attachments etc.
JPF and JUnit are testing tools, but not bug tracking tools.

2. Refactorings and Code Smells

(a) Name one automated refactoring from Eclipse that is not a Rename, Move, or EncapsulateField. Describe briefly how it transforms the code.

Solution:
Extract Superclass. Multiple classes may have common fields or methods. Select such fields or methods, and perform the “Extract Superclass” refactoring to create a new common superclass that includes the selected fields or methods. Other refactorings include “Extract Method”, “Pull Up”, “Push Down”, etc.
Common incorrect answers include formatting, commenting, indentation etc, which are source code modifications.

(b) Name one “code smell”. Describe how we can identify that code smell. Name an appropriate refactoring that removes the code smell.
Solution:
Long Methods. It can be identified as methods that do not fit in one page, for example, or are more than 60 lines long. Use the “Extract Method” refactoring to split long methods into small methods.
Other code smells include “Large class”, “Magic Numbers”, “Duplicate Code”, “Long Parameter List”, etc.

(c) Name one code smell that is not based directly on code but rather on software development activities done with the code (such as changing code).

Solution:
Possible correct answers are non-localized plan, hard to change, hard to understand, too many bugs...
Common incorrect answer was to mention a code smell that can be obtained from the code itself (long methods, magic numbers, too few/many comments...).

3. Reverse Engineering

Consider the “Reconstruct the persistent data” reverse engineering pattern from the document by Demeyer, Ducasse, and Nierstrasz assigned for reading on reverse engineering.

(a) State the problem that this pattern addresses.

Solution:
This pattern addresses the problem of finding critical objects in the system. These objects are so critical that they are persisted across executions in a data store. Since objects are available at run-time only, and the system code/specification is static, and run-time tracing is expensive, the problem is quite challenging.
Common mistake is to think that this pattern helps generate the input data for the application to test it for all behaviors.

(b) Summarize the proposed solution (one or two sentences).

Solution:
Check the entities (tables etc) that are stored in the persistent data store. Collect the fields and relationships between these entities and build a corresponding class model of the system. The class model documents the knowledge for the entire project team.

(c) Describe how this pattern reduces the risk for one of the five risk factors listed by Demeyer, Ducasse, and Nierstrasz.
Solution:
This pattern reduces the following risk factors: Reliable information, Accurate abstraction, Sceptical colleagues. By providing a schema of critical data and a model of corresponding data structure, team communication is improved. This is because many development processes stress the importance of data.

4. Metrics
(a) Describe one way in which “Lines of Code” metric can be used in a software development process.

Solution:
Estimate complexity of code, identify long methods or large classes for refactorings, arguably judge programmer’s productivity...
Common incorrect answer was to use lines of code to estimate size.

5. Testing Concurrency
Listing 1 shows two classes created for counting the occurrence of words (called terms) within a collection of documents. The current design instantiates a single TermCounter object counter, then creates multiple threads to retrieve different Document objects from the collection of documents, calling doc.addTerms(counter) for each Document object doc. The same instance counter is shared among all threads.

Note that the code uses maps from the standard library. The operations put and get are atomic themselves; put(k, v) adds a mapping from the key k to the value v (overriding the previous mapping, if any existed), and get(k) returns the current value for the key k (or null if there is no such value).

```java
public class TermCounter {
    private Map<String, Integer> termCounts;

    public TermCounter() {
        termCounts = new HashMap<String, Integer>();
    }

    public void addTerm(String term) {
        Integer count = termCounts.get(term);
        if (count == null) {
            termCounts.put(term, 1);
        } else {
            termCounts.put(term, count + 1);
        }
    }

    public Map<String, Integer> getTermCounts() {
        return termCounts;
    }
}

public class Document {
```
private String text;

public Document(String text) {
    this.text = text;
}

public String[] getTerms() {
    // ... splits text into separate terms ... 
}

public void addTerms(TermCounter counter) {
    for (String term : getTerms()) {
        counter.addTerm(term);
    }
}

Listing 1: Classes for Term Counting in a Collection of Documents

(a) Write one JUnit test case to demonstrate that the TermCounter class is not thread-safe. Your test case should have two or more threads that modify one TermCounter object, and could fail some assertion(s) on this object for some thread schedule.
Solution:
The problem is that in the time between one thread executes line 9 and either line 11 or 13 for a given term \( t \), another thread may also execute line 9 for the same term \( t \). This will cause the count of \( t \) to be off by one, for each such interleaving. A minimal solution is listed below.

```java
public class TestConcurrency {
    private final TermCounter counter = new TermCounter();

    @Test
    public void testConcurrency() {
        Thread t1 = new Thread() {
            public void run() {
                counter.addTerm("foo");
            }
        };

        Thread t2 = new Thread() {
            public void run() {
                counter.addTerm("foo");
            }
        };

        t1.start();
        t2.start();
        t1.join();
        t2.join();

        assertEquals(2, counter.getTermCounts().get("foo"));
    }
}
```

Common incorrect solutions included using strings or Documents in each thread that share no terms, using different instances of TermCounter in each thread, using the Map accessors incorrectly, or comparing objects of different (non-coercible) types in the assert statement.
6. XP Practices

(a) (True or False) Indicate whether each of the following statements is true or false by writing the word True or False on the line that precedes the statement.

True ______ When done properly, code refactoring improves some aspects of a program, such as readability and maintainability, but does not change the functional behavior of the program.

False ______ User stories are brief descriptions of requirements written by the developer in her/his language.

True ______ Test-driven development involves the use of automated or manual tests that define requirements before writing the code.

True ______ XP entails the practice of pair programming, in which two developers work side-by-side.

False ______ A JUnit test case class cannot contain a common method in which initialization code can be placed, because tests are independent.

(b) In some cases, it is not beneficial to write automated (unit) tests for your code. Name one advantage and one disadvantage of having automated tests.

Solution:
Advantages include instant feedback on code that passes the test or refactorings that fail, reinforcement of programmer’s mental model of the code, increases confidence by catching some bugs, pays back the time spent writing a test if the test is repeated many times, and increases long-term productivity.

Disadvantages include difficulty in testing GUI, tests can be a burden because they have to be maintained over the life of a project, a sense of false confidence because of unseen gaps in coverage and because passing tests do not disprove bugs. That they may not be worth the time if run infrequently was also accepted.

(c) Name and describe two types of software tests.

Solution:
Many answers were accepted if the descriptions were correct, including white-/black-box testing, unit testing, regression testing, random testing, performance/load testing, configuration tests and security tests.

Common mistakes were to limit the definitions of unit test and smoke test to checking for regression errors, and to state that smoke tests check that code builds correctly.

(d) Describe two XP practices that are not among the following four: planning game, pair programming, test-driven development, and refactoring (design improvement).
Solution:
Many answers were accepted if the descriptions were correct, including Whole team (on-site customer), Metaphor, Simple design, Small releases, Customer tests, Collective code ownership, Continuous integration, Sustainable pace (40-hour week), and Coding standards.
Common mistakes were to mention user stories or iterations (part of planning game), software configuration management (e.g., version control, change control), and maintenance. XP as a whole is iterative, but this is arguably not a practice of XP (outside of the planning game), and answers were not given in this broad sense.
7. Java PathFinder

(a) Describe one of the three main JPF operations.

**Solution:**
There are three main JPF operations. A brief description of any one of the following would receive full credit.

**Bytecode execution:** JPF is a JVM. It runs programs by executing the bytecode contained in .class files. Examples include IADD, GETFIELD, etc.

**State storing/restoring/backtracking:** JPF stores the entire JVM state (threads, dynamic area, static area) for the subject under test at nondeterministic choice points (e.g., a thread switch or a call to Verify.getInt()). JPF will then restore this state when backtracking occurs.

**State Comparison:** JPF compares the entire JVM state (threads, dynamic area, static area) for the subject under test so as to avoid reexploring redundant paths.

Common incorrect answers described the purpose of JPF (e.g., exploring all interleavings of a concurrent program) and were typically given partial credit.

(b) Describe one of the three main parts of a state of a Java program.

**Solution:**
The state of a Java program consists of three main parts. A brief description of any one of the following would receive full credit.

**Stack(s)/thread(s):** Each thread has a stack with a number of stack frames that store local variables Also a program counter (PC) and thread info.

**Heap (or Dynamic Area):** Objects in dynamically allocated memory (can be shared among threads).

**Classinfo (or Static Area):** Static data once it is loaded.

(c) JPF provides a mechanism called the Model Java Interface (MJI) analogous to the Java Native Interface (JNI) in regular JVMs. Discuss at least two reasons why a tester would utilize this mechanism.

**Solution:**
A brief description of any two of the following would receive full credit.

- Running native Java methods in JPF (e.g. System.currentTimeMillis()).
- Intergrating library methods with the JPF runtime system (e.g. java.lang.Thread.currentThreadInfo).
- Interfacing to JPF functionality – enabling usage of specialized verification API in applications (e.g., Verify.getInt(3, 8)).
- There are also several scalability reasons, e.g.:
  - native methods can save state space.
  - native methods are executed atomically.
  - native methods execute much faster.
8. JPF Exploration

Consider the following program.

```java
public class JPFExample {
    public static void main(String[] args) {
        if (Verify.getBoolean() == Verify.getBoolean()) {
            int i = Verify.getInt(0, 1);
            System.out.println("i = " + i);
        }
    }
}
```

(a) What output does JPF print to the console (in the default, depth-first exploration mode) for this program?

Solution:

```
i = 0
i = 1
```

The most common incorrect answer was: i = 0

```
i = 1
i = 0
i = 1
```
(b) How many states/paths does JPF explore when it runs this program? Do not just write a number but explain why it is your answer. Hint: you may want to draw the state space graph or list program paths that JPF explores.

Solution:
This was an admittedly difficult question. As such, it was graded leniently, provided an adequate explanation and/or diagram accompanied the answer.

If you were to run this program under JPF, the following statistics (and others) would be displayed:
states: new=6, (re)visited=3, backtracked=8, end=4
search: maxDepth=3, constraints=0
choice generators: thread=1, data=4

But those are just numbers, and don’t really tell the whole story. See the JPFExampleSearch.pdf file that accompanies this exam solution on the wiki.

9. JPF and Refactorings

(a) Some refactorings that preserve behavior of programs when they are executed in regular JVMs can change the behavior of programs when they are explored in JPF. Provide one example program where a refactoring changes the JPF exploration, i.e., the number of states that JPF explores and/or what JPF prints differs before and after the refactoring. Your answer should include the program code both before and after the refactoring.