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 10 questions on this exam and the maximum grade on this exam is 80 points.

<table>
<thead>
<tr>
<th>Page</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</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. eXtreme Programming (XP)

(a) One of the 12 pillars of XP is *Simple Design*. Practitioners of XP prefer simple design over *big design up front* (BDUF). One challenge, though, is how to maintain a simple design as the project evolves. Pick **two** other pillars of XP that you are familiar with and describe how those two pillars help drive a simple design.

**Solution:**

- **Refactoring** - Refactoring aggressively helps keep the design simple while still maintaining all the desired qualities.
- **On site customer** - By having the customer on site, it is easier to ask for clarifications about features before implementing a change that might not be necessary.

(b) In the context of XP, how are acceptance tests similar to unit tests? How are they different?

**Solution:**

Both acceptance tests and unit tests are part of the testing pillar of XP. They ensure that the software is behaving as the customer wants. They are usually automated to ensure that they can be run continuously.

Acceptance tests are black box tests written from the perspective of the customer. Passing an acceptance test is an indication that the user story has been completed successfully. Unit tests, on the other hand, are usually white box tests design for developers to test the internals of the system.
2. Reverse Engineering

(a) Describe what reverse engineering is in one sentence.

**Solution:**

The process of learning about a system that is not well-documented, e.g., rediscovering the design of a software that was written long ago.

(b) In lecture, Prof. Johnson mentioned several tools that developers could use to help them reverse engineer a system. Give two examples of such tools. For each tool, describe (i) what information they can provide and (ii) how you would use them to reverse engineer a system.

**Solution:**

- **Debugger** - set breakpoints and then step through the code to. The debugger can help reverse engineer the logic and data that are being exchanged between different modules.
- **Code browsers** - use the navigation features such as class hierarchy, call hierarchy, etc. This can us statically understand the components of the code and how they are dependent on one another.

(c) Sometimes using software metrics can be useful in guiding the reverse engineering process. Pick one software metric that you are familiar with and describe what it measures. Then, describe how you would use that metric to help guide you through the reverse engineering process.

**Solution:**

- **Number of changes to the file in version control** - measures how frequently this class is modified for each commit.

A high number is an indication that this file (and the classes in it) is particularly complex and participates in too many interactions between modules. A low number indicate that this file is not changed much and can be an example of a stable part of the system. Both complex and stable files are worth examining in greater detail during the reverse engineering process since they depict different spectrum of the project.
3. Software Architecture

3 (a) Eclipse is an example of a “plug-in” architecture. What is a plug-in, and what does it mean to have a plug-in architecture?

Solution:

3 (b) Why is a plug-in architecture a good choice for building an integrated development environment (IDE)?

Solution:

4. Software Quality Attributes

2 (a) What does the acronym FURPS stand for?

Solution:

4 (b) Describe a situation in which a design decision that improves one quality attribute will make another one worse.

Solution:
3. (c) The article *Capturing Architectural Requirements* from IBM says that providing a printing capability (in an application) is a functional requirement of particular significance to architecture. Explain.

Solution:

5. Software Configuration Management

3. (a) Name three components/documents that are part of a software product that you might want to keep under Software Configuration Management.

Solution:
Code, test suites, manuals, requirements, design documentation, change control, build configuration etc.

2. (b) Name two goals of proper Software Configuration Management?

Solution:
To keep track of how software changes over time and to be able to reproduce any version of the software.
6. Software Metrics

(a) Bill Gates has been quoted as saying “Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs”. And, yet, lines of code is the most popular software metric that we have today. In fact, look up any published paper or website and lines of code is the dominant (if not the only) metric reported. Besides being simple to collect, what are two other advantages of measuring lines of code? What is one problem of measuring lines of code?

Solution:

Universal - can be measured on any of the many different textual programming languages out there.

Build complexity - while not necessarily a good measure of productivity, lines of code has very strong correlation with the build time. Longer files take longer to parse and thus affect the build time.

The main problem of using lines of code is that it is highly dependent on the formatting of the code. Additionally, it tends to be conflated by the inclusion of comments.

(b) The object-oriented metrics we discussed in class have strong correlations with code smells. For instance, when a particular component is not within the acceptable range of a metric, it is usually a good indication that it is suffering from some code smell as well. For each metric below, list and describe one code smell that could contribute to it being out of the acceptable range. To prevent guessing, list only one code smell for each metric. If you list more than one, you get zero for that subpart.

i. Coupling Between Objects

Solution:

Feature Envy, Inappropriate Intimacy

ii. Number of Children

Solution:

Speculative Generality

iii. Response for a Class

Solution:

Divergent Change, Shotgun Surgery
7. Code smells and refactorings

Name three code smells that are present in the code below. It is OK if you circle or underline the offending code. For each code smell that you identify, suggest how you would refactor it to remove the offensive code smell. You do not need to implement the actual refactoring. Use the space on the next page for your suggestions.

```java
public class Account {
    private int accNo;
    private double amount;
    private double interestRate;

    // Constructor omitted

    public int getAccountNumber() { return accNo; }
    public double getAmount() { return amount; }
    public double getInterestRate() { return interestRate; }
}

public class Customer {
    private String name;
    private List<Account> accounts;

    public String getStatement() {
        // Print statement header
        String statement = "Statement for " + name + "\n";

        // Print info for each account
        for (int m=0; m < accounts.size(); m++) {
            Account account = accounts.get(m);

            // Get info about the account
            int accNo = account.getAccountNumber();
            double amount = account.getAmount();
            double interestRate = account.getInterestRate();

            statement += "For account " + accNo + "\n";

            // Print account summary
            statement += "Amount available: " + amount;
            statement += "interest: " + amount * interestRate;
            statement += "\nTotal: " + amount + (amount * interestRate);
            statement += "\n\n";
        }

        return statement;
    }

    // Other methods omitted

Listing 1: First Iteration of a Banking System
```
Solution:

Duplicate code: the interest rate is computed twice. Refactor the code in a new method.

Feature envy: The Customer class uses data from the Account class. Move everything that has to do with the account information to the Account class.

Uncommunicative Variable Name: The loop variable does not convey its purpose. Rename the variable to something more meaningful or use a foreach construct and get rid of it altogether.

Comments: The comments do not add any new information. Remove them.

Data class: The Account class does not have any behaviour. It’s data is used by the getStatement method in Customer. Move the code into the Account class.
8. Testing Techniques

(a) Describe two reasons why manual testing is better than automated testing. Similarly, describe two other reasons why automated testing is better than manual testing.

**Solution:**

Manual testing is appropriate when:

- tests need to be run once
- tester doesn’t know how to program
- the test is expensive to automate

Automated testing is appropriate when:

- the tests need to be run over and over as the software evolves or is changed
- when we want to use them as documentation.

(b) For each of the following cases, justify why automated testing or manual testing would work better.

i. Exploratory test cases

**Solution:**

ii. GUI tests

**Solution:**

iii. Regression testing

**Solution:**
9. In the following Java code fragment, `getPeopleAsString` is a method that accepts a set of person objects and returns a string containing the first name and last name of each person. In the returned string, the information of each person is separated from the next person by “; “. Having this code snippet, answer the following questions as described in Code Complete 2:

```java
public String getPeopleAsString(Set<Person> people) {
    String firstName;
    String lastName;
    String peopleString = "";
    for (Person person : people) {
        if (person.firstName != null) {
            firstName = person.firstName;
        } else {
            firstName = "";
        }
        if (person.lastName != null) {
            lastName = person.lastName;
        } else {
            lastName = "";
        }
        peopleString += firstName + ", " + lastName + "; ";
    }
    return peopleString;
}
```

(a) What is the minimum number of test cases needed for structured basis testing? Explain your reasoning.

**Solution:**
4 test cases

(b) What is the minimum number of test cases needed for data flow testing? Explain your reasoning.

**Solution:**
6 test cases
10. White-Box and Black-box Testing

(a) What is the main difference between white-box and black-box testing?

**Solution:**
White-box tests are based on code – better for finding crashes, out of bounds errors, file not closed errors
Black-box are tests are based on specifications – better at telling whether program meets specification, better at finding errors of omission

(b) Is equivalence partitioning considered as a white-box or back-box testing techniques?

**Solution:**
Black-box

(c) Explain how you can use equivalence partitioning for testing the code example in Question 9.

**Solution:**
Test the collection with zero item, one item, or many items (more than one item).