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.

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 17 questions on this exam and the maximum grade on this exam is **120 points**.
1. Software Configuration Management (SCM)

(a) The following change requests have been approved by the change control authority. Indicate whether branching in version control is justified and state why.

Some of our customers want to upgrade to the new Quadex-64 hardware platform, so we need to support that platform.

Justified? Yes No Reason: ________________________________

A bug has been identified in the latest release version, and a fix must be produced before the next scheduled release.

Justified? Yes No Reason: ________________________________

We will make some customizations needed by our new, important customer.

Justified? Yes No Reason: ________________________________

Sam and Erin are working on different requested features for the same project, but they can’t seem to agree on one approach.

Justified? Yes No Reason: ________________________________

(b) (Matching) On the lines provided, clearly 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   E. change request   I. private workspace
B. automated build   F. identification      J. release
C. baseline         G. mainline           K. status accounting
D. change control    H. private system build L. version control

Descriptions

_____ a formal process to ensure that changes to the system are introduced in a controlled and coordinated manner

_____ assigns revision numbers for each change to a file and permits branching and merging

_____ involves the compilation and linking of source code file 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

_____ done before committing changes to the Repository to avoid breaking the build

_____ a software configuration item that the developers give to other people
(c) If you encountered a merge conflict in your project, describe where it was and how you resolved it. If you did not encounter any merge conflict, describe advantages and disadvantages of the two approaches (locking vs. merging) used to solve the problem of two people changing the same software at the same time.

(d) Suppose that you already checked out your project from the trunk directory. Your teammate (i) checked out the same revision, (ii) changed some files locally, (iii) committed the changes to the repository, and (iv) created a tag for the changed version in the tags directory. What operations (check in, check out, commit, update, or tag) do you need to perform to obtain the latest version from trunk and the recently created tag from tags?

2. Refactorings and Code Smells

(a) Select one of the following code smells and describe how to identify it:
(b) If you identified in your project (either in the code you wrote or in the provided code) one code smell NOT listed above, name the smell and describe how you identified it. If you didn’t identify any code smell in your project, name any one code smell NOT listed above, describe how to identify it, name an appropriate refactoring to remove the code smell, and describe how this refactoring transforms the code.

(c) If you performed a refactoring in your project different from the one named above, (i) name that refactoring and (ii) describe where and how you performed it. If you didn’t perform any refactoring different from the one named above, (i) name any one refactoring different from the one named above, (ii) describe what code smell it addresses, and (iii) describe how this refactoring transforms the code.

(d) Why are tests important for refactoring? Give one good reason and one bad reason to modify a test after refactoring.
3. Reverse Engineering

(a) *A Pattern Language for Reverse Engineering* by Demeyer, Ducasse, and Nierstrasz discusses reverse engineering patterns. If you used one such reverse engineering pattern in your project this semester, name the pattern and describe how you used it. If you didn’t use any pattern in your project, name any one pattern, state the problem that it addresses, summarize the proposed solution (one or two sentences), and describe how the pattern reduces the risk for one of the five risk factors listed by Demeyer, Ducasse, and Nierstrasz.

(b) Describe how one could use debuggers for reverse engineering. Which specific function provided by a debugger during program execution is particularly useful?

4. Metrics

(a) What is the difference between technical and non-technical software metrics? Discuss why “number of tests” should be a technical or non-technical metric.
(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.

1. Code Coverage  
2. Cyclomatic Complexity  
3. Function Points

(c) Describe one technical object-oriented metric either from “OO Design Quality Metrics” by Martin or from “A Metrics Suite for Object Oriented Design” by Chidamber and Kemerer. How could you use this metric to make some decision about your project?

5. Testing (Concurrency)

The singleton pattern is a design pattern that ensures that only one object of some class is created. Listing 1 shows a class that attempts to implement the singleton pattern.

```
public final class MySingleton {
    private static MySingleton instance = null;

    private MySingleton() {} // cannot be called from outside

    public static MySingleton getInstance() {
        if (instance == null) {
            synchronized (MySingleton.class) {
                instance = new MySingleton();
            }
        }
        return instance;
    }
}
```

Listing 1: An Attempted Singleton Class
(a) Write a JUnit test case \texttt{testIsSingleton} that checks whether two calls of \texttt{getInstance()} return the same object.

(b) Write a JUnit test \texttt{testIsThreadSafe} to demonstrate that \texttt{getInstance()} may not enforce the singleton pattern in a multi-threaded program. Your test case should have two or more threads that fail some assertions on return values of \texttt{getInstance()} for some thread schedule.

```java
public class TestMySingleton extends TestCase {
    public void testNonNull() { // no explicit threads
        MySingleton s = MySingleton.getInstance();
        assert s != null;
    }

    public void testIsSingleton() { // no explicit threads
    }

    public void testIsThreadSafe() { // multi-threaded
    }
}
```

Listing 2: Tests for Singleton Class
6. Testing

(a) (Matching) On the lines provided, clearly write the letter of the test type that best matches each description. Each term may be used at most once; some will not be used.

**Test Types**

A. Black box  E. Performance  I. System  
B. Exploratory  F. Regression  J. Unit  
C. Integration  G. Security  K. Usability  
D. Load  H. Smoke  L. White box

**Descriptions**

_____ "Touches" all areas of the application without getting too deep. The purpose is to determine whether or not the application is so badly broken that testing functionality in a more detailed way is unnecessary.

_____ Seeks to uncover previously working software functionality that no longer works as intended.

_____ Uses an internal perspective of the system to design test cases based on internal structure. The tester chooses test case inputs to exercise paths through the code and determines the appropriate outputs.

_____ Checks if each of the smallest testable parts (e.g., function or procedure) of an application are fit for use.

_____ Determines how fast some aspect of a system performs under a particular workload.

(b) The assigned reading included Brian Marick’s catalog for testing (both short and long versions). If you used some idea(s) from the catalog in your own project, describe where and how you used one idea. If you didn’t use any idea in your project, describe two ideas.
7. XP Practices

(a) Give specific examples of how your Final Project group has followed these two XP practices: planning game and test-driven development.

(b) Outside of the class, you don’t need to follow XP, but you can still benefit from using some XP practices. Name and describe one XP practice that is NOT among the main four we covered (planning game, pair programming, test-driven development, and refactoring) that you would like to use in your other projects even if they follow other processes. Why would you want to use that practice?

(c) Name and describe one XP practice that is NOT among the main four we covered (planning game, pair programming, test-driven development, and refactoring) that you would NOT like to use in your other projects. Why would you NOT want to use that practice?
8. Concurrency
   (a) In the context of concurrency, what is *atomicity* (the quality of being *atomic*)? How can it be enforced in Java?

9. Actors
   (a) Unlike many concurrent programs, Actor programs do not communicate using shared memory. How do actors communicate?
   
   (b) Data races cause errors in shared-memory programs when threads read and write data in the wrong order. Actor programs do not have such races. What is a key source of errors for actor programs?
   
   (c) An actor can be thought of as an autonomous process or object. Actors were described in lecture as having four key components. For example, each actor has independent control (i.e., its own thread of computation). Describe **two** of the other three components.
10. ReLooper

(a) What are the advantages of using parallel libraries and frameworks (like `ParallelArray` and `ForkJoinTask`) to write parallel code? What are the disadvantages?

(b) Sometimes a system can only be parallelized by re-architecting the entire system. Other times, parallelism can be introduced via refactoring. Give one scenario where the parallelization would require re-architecting, and one scenario where it would be better to parallelize a system via incremental refactoring.

11. Debugging

(a) Debugging activities include tracing, backtracking (not as in JPF), and cause elimination. Describe one way to perform tracing.
(b) Define each of the following terms, and for each term state whether it is found by testing or debugging.
Failure:

Fault:

12. Assertions

(a) What is an assertion? Describe how assertions can be used as a programming tool for software development.

(b) As discussed in the lecture on debugging and assertions, preconditions are assertions that must be true before a method is executed in order for it to work. Do we need to unit test methods with inputs that violate preconditions? Discuss why or why not; simple “yes” or “no” does not suffice.
13. Reuse

(a) **Describe** two costs of reusing free or open-source software.

(b) Components are one approach to software reuse. Describe one of the other approaches.

14. Design Patterns


(a) Choose one pattern that we **did cover** in lectures. Describe the “Intent” for this pattern, i.e., the goal behind the pattern and the reason for using it.

(b) Name one pattern that we **did NOT cover** in the lectures but that you did encounter in your own project, either in the provided (JPF) code that you worked with or in the new code you wrote. Where did you encounter this pattern: existing code or your code, what module/package/class? If you did not encounter any pattern in your project, then describe the “Intent” for one pattern that we did NOT cover in the lectures.
(c) What were the “Participants” in this pattern that we did NOT cover in the lectures, i.e., which classes/methods mapped to which parts of the pattern? If you did not encounter the pattern in your project, then give one concrete example of that pattern.

15. Documentation

(a) Name two other artifacts apart from user manual that can serve as software documentation.

(b) Documentation for software systems sometimes provides a description of the entire system for the current version and sometimes provides only the differences from a previous version. Describe the advantages and disadvantages of documenting the entire system vs. documenting only the differences.

(c) What part of the documentation in the provided code (for JPF, ActorFoundry, Eclipse, Java libraries...) did you find useful and why? What part of documentation was lacking in the same code?
16. Potpourri

3 (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.

Design patterns provide reusable pieces of code like libraries and components do.

One benefit of software reuse is shortening the time needed to implement the system.

Object-oriented frameworks should have only concrete classes and have no abstract classes.

Binary search function should always traverse its entire input array to determine whether it is sorted.

MJI interface in JPF requires mangling of method names to handle overloaded methods.

A program transformation is a refactoring only if you can automate the transformation.

17. Coding Style

3 (a) Which of these two code fragments would you prefer to use in your code to call some method with a name of some class? Discuss also why; a simple “A” or “B” does not suffice.

A. someMethod("somePackage.someClass");
B. someMethod(somePackage.someClass.class.getName());