

Midterm

This exam is worth 25% of the overall grade. You are allowed one sheet of notes, both sides. No devices are permitted, nor any other kind of assistance. Numbers in square brackets are the points for each problem or subproblem; there are 100 points total.

Total time: 80 minutes

1. True/False [18 points (3 each)]

For each question, answer true or false. You can explain your view if needed below.

- a) XP is the extreme form of iterative programming.
- b) Garbage collection can work with C.
There exist conservative collectors for C/C++.
- c) UML is the merger of multiple modeling languages.
- d) Java does not have memory leaks.
You can create connected subgraphs that are not usable.
- e) XP programmers start by writing simple versions of the target program.
They start by writing test cases.
- f) Version control is only useful for team projects.

A	True
B	True
C	True
D	False
E	False
F	False

2. Design Patterns [10 points]

- a) [3] Which pattern is good for a set of subclasses that implement the same interface?

Composite

- b) [3] Explain the motivation for the *Abstract Factory* pattern.

The factory creates a set of related classes that have the right features for the current platform. This avoids having lots of conditionals in the main code (as they are all inside the factory).

- c) [4] What is the difference between a visitor and an iterator?

Visitors allow type-specific actions.

Midterm

3. Testing and Debugging [34 points]

a) [5] Assertions are a useful tool for development. What problem do they lead to though?

Primary answer: run-time errors, which makes them better for development than in released code. They can also be wrong sometimes.

b) [5] What is the most effective technique for finding bugs?

Code review (much faster than testing)

c) [5] Most projects become delayed during what phase of development?

Integration

d) [6] What is regression testing?

The process of regular automated testing (usually nightly) against a set of test cases from previous bugs. Most bugs should lead to new test cases for regression testing (unless that's how they were found).

e) [3] True/False: Programmers should not do unit tests, but leave it to testers instead

False

f) [4] Given two specific examples of static analysis being used to reduce bugs.

Many examples were fine: type checking, lock verification, splint, ...

g) [6] Explain what kind of bad writes Purify can detect.

It can detect writes over unallocated memory. It can not detect writes to memory allocated to other objects. It can also detect reads to uninitialized memory and read to unallocated memory.

Midterm

4. Version Control [20 points]

- a) [4] What is the problem with waiting a few days to check in your changes?

High probability of conflicts for a normal active team project... Besides conflicts during your own checkin, you may cause others to have conflicts as well. Of course, you may also spend time fixing problems that have already been fixed.

- b) [3] True/False: In CVS, conflicts might be detected when you *compile* files.

True

Two concurrent updates may have conflicts that weren't detected at checkin time, as they affected different lines. The example from lecture was that one person changed the number of arguments to a function, and another person added a use of the old function. This will be detected by the compiler (but not by CVS).

- c) [6] Explain when to use "branches".

A few different cases:

- when you need to fix a bug on a released version, you keep a separate branch for it. Every release typically has a named branch.

- when you are starting a major revision that touches much of the code, you typically use a new branch so that others can get work done; later you merge your changes into the main branch when it is reasonably stable.

- d) [7] Sometimes when programmers modify the same file, CVS is able to merge the changes without any conflicts. Roughly, how does this work?

It looks at differences from the original source file (using "diff3") and looks for changes that affect the same lines -- these are conflicts. Those that don't conflict can be merged in using a process like "patch", which uses the context of the lines, rather than the line numbers, to figure out where to apply the changes.

Midterm

5. Exceptions [18 points]

This question refers to exceptions in Java.

a) [6] What is the relationship between exception handling and opening files?

Looking for two things in this answer:

1) opening files has several exceptional conditions (e.g. file not found)

2) need to make sure the file gets closed in all cases, for which try/finally works well

b) [6] What leads to multiple levels of nesting in exception handling?

The basis nesting that we covered in lecture is due to the need to clean up each exception involving files, sockets, etc. For example, if you open three files, the simplest solution has nested exception handling with a depth of three. That is, you only open the second file if the first one opens correctly, hence it is nested within the first one.

c) [6] Many programmers use empty exception handling clauses. Why do they do this and what is wrong with it?

The primary reason is laziness -- they just wanted to make the compiler errors go away.

This is bad for many reasons, but the primary one is that they aren't actually handling the exceptions at all, which is rarely the right response.