Midterm Exam

CS169 – Fall 2009

November 17, 2009

Please read all instructions (including these) carefully.
Write your name, login, and SID.
There are 12 pages in this exam and 6 questions, each with multiple parts. Some questions span multiple pages. There are some easy parts and some hard parts. **If you get stuck on a question move on and come back to it later.**
You have 1 hour and 20 minutes to work on the exam.
The exam is closed book, but you may refer to your four pages of handwritten notes.
Please write your answers in the space provided on the exam, and clearly mark your solutions.
You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. Partial solutions will be graded for partial credit.

LOGIN:

NAME:

SID:

Problem	Max Points	Points
1	10	
2	12	
3	30	
4	18	
5	18	
6	12	
TOTAL	100	

1. Software Processes (10 points)

a) (4 points) What are some differences between the waterfall and agile software processes?

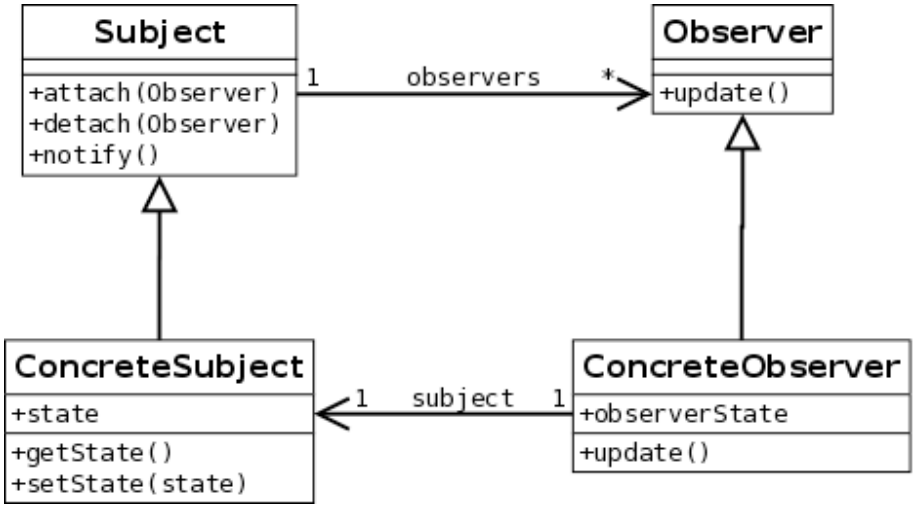
b) (3 points) In what kind of software process do you plan for code refactoring? Why?

c) (3 points) Why is it a good idea to write tests before you write code?

2. Design (12 points)

a) (3 points) What are the differences between requirements, specifications, and design?

b) (4 points) What kind of UML diagram is shown below? Describe the important elements of the design it specifies. There is no need to list the field names or methods in your description.



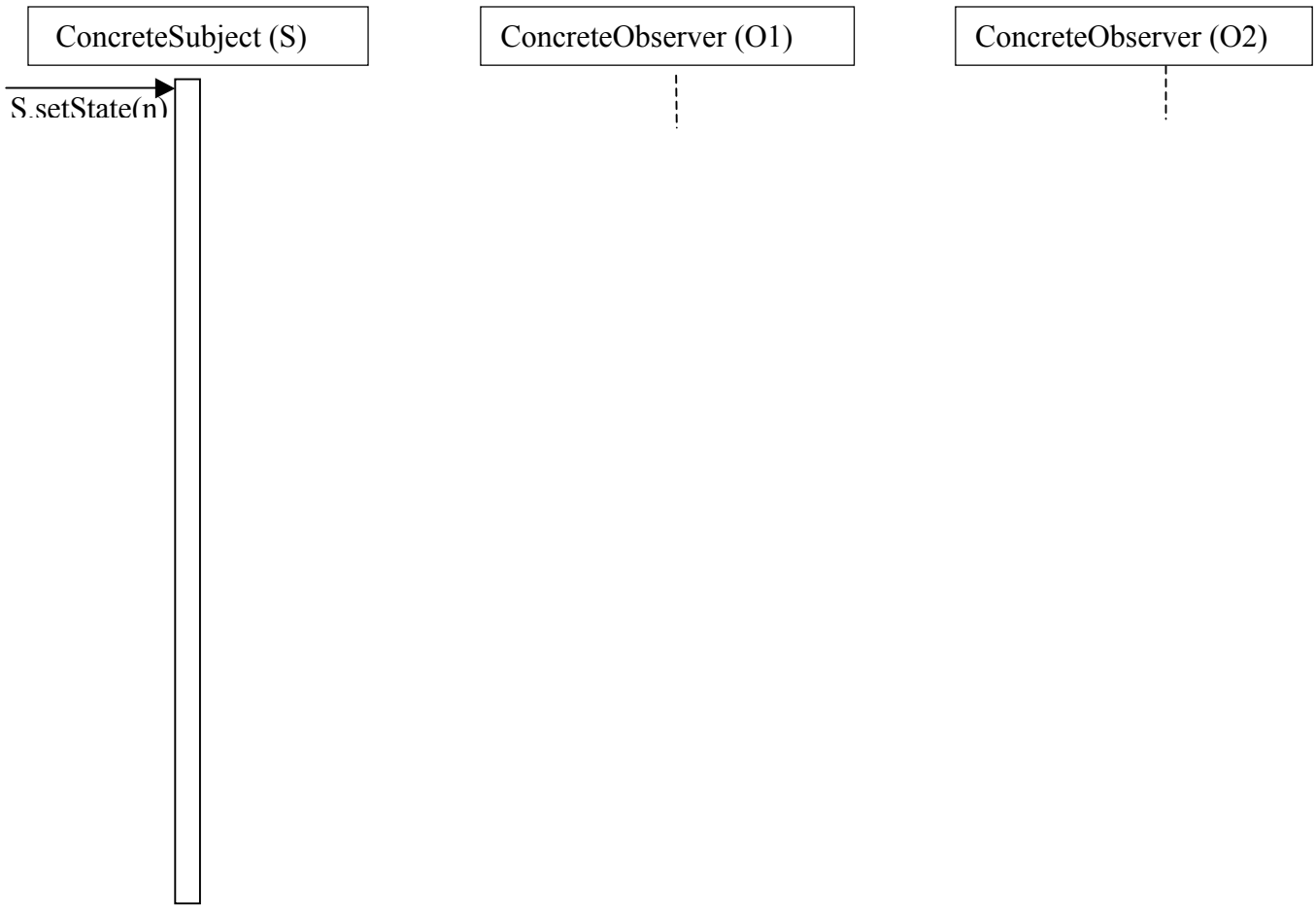
c) (5 points) Here are the concrete implementations of some of the key methods.

```
class ConcreteSubject:
public void setState(int s) {
     state = s;
     notify();
}

Class Subject:
public void notify() {
     for(Observer o: observers) o.update();
}

Class ConcreteObserver:
public void update() {
     observerState = subject.getState();
}
```

Assume we have a scenario with one ConcreteSubject object called S, and 2 ConcreteObserver objects, called O1 and O2. Draw the remainder of the sequence diagram below, assuming we have already attached the two observers O1 and O2 to S and we call S.setState(n):



3. Testing and Code Coverage (30 points)

Consider the following function that given a valid date (year y0, month m0, and day d0) and a positive number of days in the future (delta), computes a valid date in the future that is delta days after the date specified on input. Days and months are numbered starting at 1. This code uses an auxiliary function DM(m) that returns the number of days in the month passed as argument.

```
      incrDate(int y0, int m0, int d0, int delta) {
1       int y = y0;
2       int m = m0;
3       int d = d0 + delta;
4       while ( d > DM(m) ) {
5          d = d - DM(m);
6          m ++;
7          if( m == 13 ) {
8             m = 1;
9             y ++;
           }
        }
10      printf "Result: %d/%d/%d", m,d,y;
```

a) (6 points) Write a test that achieves 100% statement coverage, while using the **smallest** value of "delta". Show the input values for such a test (valid date and positive delta), and show the sequence of statement lines covered.

b) (6 points) Write a test that achieves 100% **branch** coverage, while using the **smallest** value of "delta". Show the input values for such a test (valid date and positive delta), and show the sequence of statement lines covered.

c) (8 points) Now consider data-flow coverage (def-use pairs). Enumerate the def-use pairs separated into 3 groups, for the variables y, m and d. (Use the notation D:U for a def in line D and use in line U). Leave some space around your pairs because you will have to circle some of them later on.

d) (4 points) Circle those def-use pairs in the answer for previous question that are **covered** by the test in your answer to point **b).** What is the value of the data-flow coverage for the test you gave in **b),** expressed as a fraction?

e) (3 points) Is it possible to achieve 100% data flow coverage for our program with a single test? Describe such a test if your answer is "yes" and explain why not otherwise.

f) (3 points) Explain why we can still have bugs in code that is tested with 100% coverage (any structural coverage measure). What is such a bug in our example code (considering only valid input dates and positive deltas) ?

4. Delta Debugging (18 points)

a) (3 points) Assume that you are debugging a command-line tool that takes as input an XML file. A tester gives you a very large XML file that makes your most recent version of the tool (revision number 100) fail an assertion. You then try to use a previous release of the tool (revision number 60) on the same input file, and you do not get the assertion failure. Explain how you can use delta debugging to help debug this problem.

b) (4 points) The delta debugging algorithm relies on several assumptions about the behavior of bugs (monotonicity, consistency, unambiguity). Which one of these assumptions is the most important, i.e., if the assumption fails then delta debugging would not really be effective? Explain your answer.

c) (7 points) Now consider the more general version of the algorithm. We assume monotonicity, consistency and unambiguity. Let the set of changes be {A,B,C,D,E,F,G,H}, and assume that the program fails when all changes are applied, and succeeds when none of the changes are applied. We run the DD algorithm to isolate the smallest possible set of changes that cause the failure. The calls to the DD routine have been shown in the order in which they would be executed. However, some of the calls or results have been intentionally left blank for you to fill in. For any blank with more than one possible answer, list any of the possibilities. Your solutions should be such that the algorithm stops after run 9.

Run 1:	A	B	C	D					√
Run 2:					E	F	G	H	√
Run 3:	A	B	C	D	E	F			√
Run 4:	_____								X
Run 5:	A	B	C	D			G		√
Run 6:	_____								__
Run 7:	A	B			E	F	G	H	X
Run 8:	_____								__
Run 9:	_____								__

d) (4 points) Considering the possible solutions to the previous problem, what is the smallest and the largest possible result of the delta debugging algorithm?

5. Version Control (18 points)

You have a project with 4 files: foo.cc, bar.cc, test.cc, and install, all checked in a Subversion repository. After a Subversion "update" command you see the following output:

```
U   test.cc
C   bar.cc
G   foo.cc
```

Recall the subversion abbreviations: U – update, C – conflict, G – merge

a) (4 points) Explain, for each of the 4 files, in what state they were before the merge in the local copy and the repository and what has happened to them during the merge.

b) (2 points) What should you do before you try to commit the result of the "update" above?

c) (2 points) Explain what measures you can take to minimize the occurrence of merge conflicts.

d) (3 points) Is the following statement true or false? Why? “If we merge two working versions of the program, and there are no conflicts during a merge, then the result of the merge will be working too.”

e) (3 points) Explain how you can use a merge command to revert in your local copy the changes committed in revision 12?

f) (4 points) Compare the advantages and disadvantages of two branching strategies: release branches (where development is done on the trunk and branches are created for testing and release) and development branches (where development is done on branches which are then merged for testing and release)

6. Concurrency (12 points)

a) (4 points) Why are data races and deadlocks generally hard to find by testing? Which of those two kinds of bugs are easier to debug once found?

b) (4 points) Here is a snippet from the Java String implementation. Strings are immutable, and we further assume the hash code computation is deterministic, i.e. it always returns the same result for the same String. Imagine multiple threads calling hashCode on a shared String object. Are there any data races or deadlocks in this code? If so, why do you think they haven't been fixed?

```
public final class String {
     private int hash; // initialized to 0 in constructor
     ...
     public int hashCode() {
          int h = hash;
          if (h == 0) {
               // … compute hash code in h …
               hash = h;
          }
          return h;
     }
     ...
}
```

c) (4 points) Consider a hypothetical SafeStringBuffer implementation, in which we extend the original StringBuffer class to provide a safe, properly synchronized append method:

```
public void safeAppend(SafeStringBuffer target) {
     synchronized(this) {
          synchronized(target) {
               this.append(target);
          }
     }
}
```

Consider two threads sharing two SafeStringBuffers s1 and s2:
T1: s1.safeAppend(s2);
T2: s2.safeAppend(s1);

Is this code safe? If not, how can you fix it?