Spring 2016 Course: CS 498, Section DM

Software Testing

Machine Problem 3
Assigned: March 21, 2016
Due: Monday, March 28, 2016 (by 11:59:59pm)

This MP covers the material from chapter 3 of the textbook and provides some starting point for your project. You will obtain some initial files and submit your added and modified files through SVN. Please do not add unnecessary files or directories (e.g., target); you will get negative points. As before, your individual SVN directory is https://subversion.ews.illinois.edu/svn/sp16-cs498dm/netid. You will also have team/individual project directory (see Problem 4 for more details). You are encouraged to check all your code and tests on the VM because they are graded there.

There are four problems (and our recurring, no problem task), worth a total of 120 points. You need only 100 points to get the maximum score for this MP, the same as for previous MPs.

You should not collaborate on any problem, except that you should do Problem 4 with your team. You should not share your solutions or code on Piazza! Write your solutions individually, without discussing them with others. If you use some online resource, you must credit the original source. Many online resources have bugs; if you credit no source but have the same bugs, you will get negative points!

No Problem [5 points]: If you notice some bugs related to the course (in book, slides, code, tests, text...) during this MP, please add a file mp3/bugs3.txt. You can also report bugs directly to Darko.

Problem 1 [35 points]: (This is a modified version of Exercise 1 after Section 3.3, page 130, Chapter 3.) The goal is to show how logic coverage is sensitive to the code structure/syntax rather than semantics, i.e., two code versions that are syntactically different but semantically equivalent can require quite different test sets to satisfy some coverage criteria. Consider the method checkIt below (also committed in SVN, where you should add your changes as marked by TODO):

```java
public static void checkIt(boolean a, boolean b, boolean c) {
    if (a && (b || c)) {
        System.out.println("P is true.");
    } else {
        System.out.println("P is false.");
    }
}
```

(a) [5 points]: Transform checkIt to checkItExpand, a method where each if statement tests exactly one boolean variable as discussed in Section 3.3.1 (pages 127-129).

(b) [10 points]: Instrument checkItExpand and add code to record which edges are traversed by a test set not only one test. For example, your instrumentation should print that calling checkItExpand with {TTT, FFF} covers some edges, say, branch1-then, branch1-else, and branch2-then. You can assign unique ids to edges instead of using descriptive names. Hint: you cannot simply add print statements to the code but have to use some data structure to track what is covered.

(c) [10 points]: Derive a GACC test set $T_G$ for checkIt. Write your test set in JUnit.

(d) [10 points]: Derive an Edge Coverage test set $T_E$ for checkItExpand such that $T_E$ does not satisfy GACC on the predicate in checkIt. Write your test set in JUnit.
Problem 2 [32 points]: (This is a modified version of exercises after Section 3.2, page 119, Chapter 3.) Write answers in a file `mp3/prob2.txt` in SVN. For each of the predicates \( P_1 = (a \land b) \lor (b \land c) \lor (a \land c) \) and \( P_2 = a \land (b \lor c) \), do the following:

(a) [2*2 points]: Identify the clauses that go with the predicate \( P \).
(b) [2*2 points]: For each clause \( X \), show all values for other clauses that make \( X \) determine the value of \( P \). (You can compute and simplify \( P_X \) as shown in the book, or you can use an ad-hoc approach.)
(c) [2*2 points]: Write the complete truth table for the predicate. (Use the format provided in the given file, with rows going from TTT to FFF, labeled starting from 1, as in the example underneath the definition of combinatorial coverage on page 107 in Section 3.2.)

(d) [2*2 points]: Identify all pairs of rows from your table that satisfy GACC with respect to each clause.
(e) [2*2 points]: Identify all pairs of rows from your table that satisfy CACC with respect to each clause.
(f) [2*2 points]: Identify all pairs of rows from your table that satisfy RACC with respect to each clause.
(g) [2*2 points]: Identify all 4-tuples of rows from your table that satisfy GICC with respect to each clause. Identify any infeasible GICC test requirements.
(h) [2*2 points]: Identify all 4-tuples of rows from your table that satisfy RICC with respect to each clause. Identify any infeasible RICC test requirements.

Problem 3 [28 points]: (This is a modified version of Exercise 5 after Section 3.3, page 131, Chapter 3.) This problem considers the TestPat class from page 56, Chapter 2. (This is the same class from Problem 3 in MP1 and Problem 3 in MP2.) Write answers in a file called `mp3/prob3.txt` in SVN. Identify the predicates in this code and find one test set for each of these coverage criteria:

(a) [7 points]: Predicate Coverage
(b) [7 points]: Clause Coverage
(c) [7 points]: Combinatorial Coverage
(d) [7 points]: Correlated Active Clause Coverage

Your tests should ensure reachability. You do not need to derive the expected outputs, but you can use the test cases from Table 2.5 on pages 59-60, which provide the expected outputs.

Problem 4 [20 points]: The goal of this problem is to help you with your project. You will commit some files in the `sp16-cs498dm/_projects/netid1[_netid2][_netid3]` directory for your project team, where netid1, netid2, and netid3 are the NetIDs of your team members (sorted alphabetically). That SVN directory can be accessed by all team members and the teaching staff but not by any other students.

(a) [5 points]: Commit a file `/_projects/netid1[_netid2][_netid3]/repo.txt` that describes where you will keep and share your team’s work. (If you’re doing the project individually, still commit `repo.txt` in the project directory `sp16-cs498dm/_projects/netid` in the individual directory `sp16-cs498dm/netid`). You can keep your work either in some online repository (e.g., GitHub) or in the provided SVN. If you use SVN, please DO NOT commit in SVN the entire source of the project you’re testing; instead, commit only the files you’re changing and some script to reconstruct the entire source (e.g., something similar as `mp0/openmrs.sh`). If you’re using GitHub (likely easier and better), fork the original project and ensure that all team members can access the forked repo, e.g., have each team member push some test commit to your fork (and accept it as pull request if you set it up that way). If you need some help, please ask Darko!

(b) [5 points]: Find some non-trivial predicate (say, with at least three clauses) in the code you’re testing, or write some non-trivial predicate for some part of the code, e.g., a precondition for some complex method or a specification for some interesting functionality. To search the code, you can use some command like this: `find -name "*java" | xargs grep -rE "(&&|\n)\* \((&|\n)\)" | less`. Describe in `/_projects/netid1[_netid2][_netid3]/predicate.txt` the predicate you found or wrote.

(c) [10 points]: Develop tests that satisfy some logic coverage criterion for the predicate. Describe in `/_projects/netid1[_netid2][_netid3]/logic.txt` where your tests are (committed in your project’s repo), discuss what criterion you used, what other criteria could have been used, and compute how many test inputs you’d have for “Combinatorial Coverage (CoC)”. As always, if you notice some bug, please do report it to Darko, and you can even submit a bug report if you don’t want to wait for Darko’s comments.