Spring 2015 Course: CS 498, Section DM

Software Testing

Machine Problem 3
Assigned: March 10, 2015 (by 11:59:59pm AoE time)
Due: Wednesday, March 18, 2015 (by 11:59:59pm local time)

This MP covers more material from chapters 2 and 3 of the textbook and more advanced usage of Jenkins. You will obtain some initial files and submit your added and modified files through SVN. If you add some unnecessary files (e.g., target), you will get negative points. As before, your SVN directory is https://subversion.ews.illinois.edu/svn/sp15-cs498dm/netid, where netid is your NetID. You are required to use the department-provided virtual machines (VMs) for some parts of this MP, but you are NOT required to use these VMs for all other parts. However, you are strongly encouraged to set up Jenkins to run your MP3 code on your VM even if you develop the code on another machine (e.g., on your laptop).

There are four problems (and an easy, no problem task), worth a total of 120 points. You need only 100 points to get the maximum score for this MP, i.e., if you have 100 points on each MP, you get A+ for the MP portion of the course. If you have more than 100 points, you will get credit for the future homework assignments and the project.

Discussing these problems on Piazza is allowed and encouraged, but do NOT share your solutions or code on Piazza! If you share your solutions, we will have to give negative points. Write your solutions individually, without discussing specific code with others. However, you can use any online resource, including the book web page (http://cs.gmu.edu/~offutt/softwaretest/) for help, but if you copy some solution/idea, please do credit the original source. Many online resources have bugs; if you credit no source but have exactly the same bugs (which would be rather unlikely), you will get negative points!

No Problem [5 points]: If you notice some bugs related to the course (in book, slides, code, tests, text…) any time during the course, please report them to Darko, and you can get extra credit!

Problem 1 [25 points]: This problem is based on an FSM problem created by Michael Robson from Craig Zilles’ cs233 class. The requirements are somewhat ambiguous but represent well how (im)precise real requirements can be:
My sleep cycle can receive three inputs—b (bell), m (music), and e (exhaustion)—and has three states—A (awake), L (light sleep), and D (deep sleep). I behave according to the following rules:
1. I am initially awake (A). I will also move back to this state if the bell b on my alarm clock goes off, unless I am exhausted. If my alarm bell b continues to go off, I will stay awake.
2. Whenever I am in the initial state (awake) and m is heard, I move into a light sleep (L). I will stay here until I receive another input.
3. When I am in the initial state and an e is felt then I will immediately go into deep sleep. As long as e is received, I will remain in this state (regardless of other inputs).
4. If both e and m are received at once, e takes precedence.
5. If I hear music m while lightly sleeping L, I will move into a deep sleep D. If I am already deeply sleeping I will continue to do so.
(a) [0 points]: Draw an FSM to model these sleep habits. Use the variables and states as named above. Indicate what your start state is. Write expressions for your transitions. This will help with the rest, but you need not turn anything in.
(b) [5 points]: We have provided some starting code in SVN in _shared/mp3/trunk but not in your netid/mp3 because you may want to set up your 1(b) solution with branches, to be used for problem 2(c).
Read 2(c) before proceeding. While you can use some other code, and not this 1(b), for your work in 2(c), it may be easier for you to use this code from 1(b) in 2(c). For 1(b), you should set up a directory in your own netid/mp3 in which one can run your code and tests for 1(c) and 1(d). Once you “cd” into netid/mp3, see copy.sh for more info, and you run it with “./copy.sh”. Remember to “svn add” all the files needed to run your code; the best is to check with Jenkins!

(c) [10 points]: Write at least 10 JUnit tests for the FSM in SleepCycleTest.java. We’ll run your tests on other students’ code as usual.

(d) [5 points]: Write code for the FSM in SleepCycle.java. We’ll run other student’s tests on your code as usual.

(e) [5 points]: Write your answer in the provided mp3/cost.txt. Consider testing, say, 1000 lines of code. How much would you charge for achieving 80% statement coverage? (You can give an absolute amount in dollars or relative to something else.) How much (more) for 90%? How much (more) for 100% (all feasible statements, showing that others are infeasible)? How much (more) for 100% prime path coverage? How much (more) for 100% all-du-paths?

Problem 2 [30 points]: You are again hired as an expert Jenkins consultant.

(a) [10 points]: Install the Git plugin (https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin) in your Jenkins at cs498dm-XYa and create a job for some non-cs498dm project that uses Git (e.g., your project from another course, your own GitHub/ACM/research code, or some small-ish open-source project such as JGraphT or OpenMRS). Run at least three builds for this job (you need not run all tests, especially for OpenMRS). Submit in SVN the Jenkins configuration file for this job as config_git.xml and the latest build console as console_git.txt.

(b*) [5 points]: So far you have been manually triggering Jenkins builds. However, a more common setup is to automatically trigger builds when the code goes into the repository. Your task is to set up some automatic trigger for one of your projects. While we try to arrange with TSG to set up SVN hooks, (1) read “Post-commit hook” at https://wiki.jenkins-ci.org/display/JENKINS/Subversion+Plugin and “Push notification from repository” at https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin, and (2) set up some shell script/alias to trigger a job when you commit to your SVN repo or push to your Git repo. Submit in SVN a file called trigger.txt that describes how you set up auto trigger.

(e*) [5 points]: So far you have had only one directory/branch where you have been committing your code and tests, even if the tests fail. However, a more common setup is to have multiple branches, commit in one branch, and only if the tests pass, merge into the main branch. Your task is to set this up for one of your projects. Read https://wiki.jenkins-ci.org/display/JENKINS/Subversion+Merge+Plugin or “Using Git, Jenkins and pre-build branch merging” at https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin, and set up one of your jobs to handle this scenario. Submit a file merge.txt that describes your setup.

(d***) [5 points]: We mentioned flaky tests in one lecture. Qinzhou Luo and John Micco developed the Flaky Test plugin (https://wiki.jenkins-ci.org/display/JENKINS/Flaky+Test+Handler+Plugin). Connect this plugin to one of your jobs, run some “Deflake Build” builds, and submit the Jenkins configuration file as config_flaky.xml and the latest build console from a “Deflake Build” as console_flaky.txt.

(e***) [5 points]: Jenkins has not only unit tests, some of which we saw in MP2, but also acceptance tests (https://github.com/jenkinsci/acceptance-test-harness) that automate end-to-end testing via a browser. The script mp3/jenkins-acceptance.sh can run three tests (two JUnit tests for the Cobertura plugin and one Cucumber test for a basic freestyle job) in a “headless” mode on your VM. Capture a screenshot from a run of one of these tests (either running in a real browser on your laptop or capturing screenshot in PhantomJS), and put the screenshot on your VM in a file called acceptance.jpg.

Problem 3 [32 points]: (This is a modified version of exercises after Section 3.2, page 119, Chapter 3.) Write answers in a file called prob3.txt in your mp3 directory. For each of the two predicates $P_1 = (a \land b) \lor (b \land c) \lor (a \land c)$ and $P_2 = a \land (\neg b \lor c)$, do the following:

(a) [2*2 points]: Identify the clauses that go with the predicate $P$. 

(b) [2*2 points]: Identify the clauses that go with the predicate $P_1$.
(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 prob3.txt, with rows going from TT…T to FF…F (changing first clause ‘a’ then ‘b’ then ‘c’), 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 4 [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 page 56, Chapter 2. (This is the same class from Problem 4 in MP2 and Problem 4 in MP3.) Write answers in a file called prob4.txt in your mp3 directory. Identify the predicates and find test sets for 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.