Analysis of Software Artifacts

MSIT-SE-M-04

Project 3 – Black & white box testing design and assessment

Due: Date shown on syllabus

Overview

In this project, you will test the `ShapeClassifier` program using both black-box and white-box testing techniques. The test suites will then be evaluated in terms of the number of test cases, the coverage achieved, and the number of defects identified.

The goals of this project are:

- To learn to model the input domain of an application in terms of equivalence classes
- To learn to use combinatorial testing tools to generate test cases
- To learn to use structured (basis paths) testing techniques and tools
- To gain an appreciation of the strengths and weaknesses of black and white box testing
- To use coverage measures to assess the adequacy of a given test suite
- To evaluate the efficiency of the test suites employed

Resources:

1. Testona (http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html)
2. JaCoCo (http://www.eclemma.org/jacoco/)
3. pitest (http://pitest.org/)
4. McCabe IQ (Available through your VM)
5. `ShapeClassifier.java` source code on the Moodle website

Process

The project comprises four phases: black box testing, white box testing, mutation testing and evaluation.

Important note: Measurements should only be taken for the `evaluateGuess` method

1. Black-box Testing:
    a. In this phase the group will derive test cases from the program specification below. You should refrain from looking at the code as a source for test cases
    b. Model the input domain using the classification tree method
    c. Document any exclusions or assumptions made
    d. Generate the following test suites: All single values, All pairs and All triples
    e. Identify the expected result for each test case
    f. Run the test cases and identify whether the test passes (result matches the expected result) or not
    g. Measure the level of statement and branch coverage achieved by each suite
    h. Measure the number of basis paths executed using McCabe IQ

2. White-box testing:
   a. In this phase of the project you will derive test cases from the source code of the `evaluateGuess` method using structured testing
   b. Using McCabe IQ, generate the control flow graph for the `evaluateGuess` method
   c. Using the baseline method described in NIST Special Publication 500-235 generate the set of test cases for the `ShapeClassifier`
   d. Identify the expected result for each test case
   e. Run the test cases and identify whether the test passes (result matches the expected result) or not.
   f. Measure the level of statement and branch coverage achieved by the test suite
3. Mutation testing
   a. Fix the faults identified. All the test cases on the test suite must pass before you can run this tool
   b. Run the mutation tool with each of the test suites recording the mutation score achieved by each of them in Table 1
4. Evaluation
   a. Complete the Test Summary Table below
   b. Respond to the lessons learned questions

Analysis of Software Artifacts
MSIT-SE-M-04
Project 3 – Black & white box testing design and assessment
Due: Date shown on syllabus

Table 1 Test Summary Table

Reference	Metric	Basis paths (A)	All singles (B)	All pairs (C)	All triples (D)
1	# number of lines of code				
2	# lines executed by the test suite				
3	# number of branches				
4	# branches executed by the test suite				
5	# basis paths				
6	# basis paths executed by the test suite				
7	# test cases				
8	# of failures identified				
9	# of faults identified				
10	Fault density	$(9, A)/(1)$	$(9, B)/(1)$	$(9, C)/(1)$	$(9, D)/(1)$
11	Statement coverage	$(2, A)/(1)$	$(2, B)/(1)$	$(2, C)/(1)$	$(2, D)/(1)$
12	Branch coverage	$(4, A)/(3)$	$(4, B)/(3)$	$(4, C)/(3)$	$(4, D)/(3)$
13	Basis path coverage	$(6, A)/(5)$	$(6, B)/(5)$	$(6, C)/(5)$	$(6, D)/(5)$
14	Mutation score	From tool	From tool	From tool	From tool
15	Coverage per test case wrt Statement Coverage	$(11, A)/(7, A)$	$(11, B)/(7, B)$	$(11, C)/(7, C)$	$(11, D)/(7, C)$
16	Coverage per test case wrt Branch Coverage	$(12, A)/(7, A)$	$(12, B)/(7, B)$	$(12, C)/(7, C)$	$(12, D)/(7, C)$
17	Coverage per test case wrt All Basis Paths Coverage	$(13, A)/(7, A)$	$(13, B)/(7, B)$	$(13, C)/(7, B)$	$(13, D)/(7, B)$
18	Faults per test case	$(8, A)/(7, A)$	$(8, B)/(7, B)$	$(8, C)/(7, C)$	$(8, D)/(7, C)$

Analysis of Software Artifacts
MSIT-SE-M-04
Project 3 – Black & white box testing design and assessment
Due: Date shown on syllabus

Deliverables

Prepare a report and presentation slides (15+10 min) including:

1. The list of test cases (As an appendix)
2. The classification tree for the input domain
3. The specifications used to generate the All singles, All pairs and All triples test suites
   a. Parameters
   b. Values
   c. Relations
   d. Constraints
4. The control flow graph and the set of test paths generated with McCabe IQ
5. The faults identified together with the reference to the test case that identified it
6. The coverage reports
7. Mutation scores for each test suite
8. Test Summary Table
9. Analysis and comparison of white-box and black-box techniques.
   a. Did one technique out-perform the others?
   b. Which one is preferable?
   c. Why?
   d. Do you need to take in consideration other factors?
10. The curve describing the average marginal cost of finding a new fault for the All singles, All pairs and All triple test suites

Lessons learned

Discuss the following as well as other points you might find relevant.

1. Are measurements 15 to 18 in Table 1 good proxies for the efficiency of a test suite? Explain your answers
2. How could you justify the extra effort of finding an additional fault?
3. What did you find difficult?
4. What did you find effective?
5. If you had to start again, what would you do differently?

Grading & Expectations

1. Slides and Report – 85 points. The report is simply an organized collection of the slides and all the material developed for the project. No need for an extensive write-up. Just a table of contents, section heads and a brief description or explanation of the material included so that we can understand what you did.

   a. Classification tree for the problem (10 points)
   b. CFG & basis paths (10 points)
   c. The faults identified together with the test case which identified it (10 points)
   d. Coverage & mutation reports (10 points)
   e. A table summarizing the results (10 points)
   f. Marginal cost curve (5 points)
   g. Analysis. What is preferable, why, assumptions (15 points)
   h. Lessons learned. What did you find difficult? What did you find effective? (15 points)

2. Class questions and presentation – 15 points

Appendix A. `ShapeClassifier` Specification

The function of the `ShapeClassifier` is to verify that a user provided shape description matches the parameters defining it. For example the identification "SQUARE, LARGE, Yes" matches the parameters 50, 50, 50, 50 which defines a square, e.g. it has four sides which are all equal, the product of the sides is even and their sum is greater than 100.

In the case above the program will respond "Yes" and in the case of miss identification, e.g. a "Square" with different sides, the program will respond "No". If the program computes three bad answers in a row, it will display a message and stop.

Limit your testing to valid inputs, do not use invalid test cases such as non-defined shapes values, sizes and evenness and non-numeric parameters. Beware, that a "Equilateral" guess, with two parameters is not an invalid test case but a bad guess.

<ShapeClassifier Input>::= <Shape Description>, <Shape Parameters>

<Shape Description>::= <Shape>, <Size>, <Even>

<Shape>::= ["Line" | "Circle" | "Ellipse" | "Equilateral" | "Isosceles" | "Scalene" | "Square" | "Rectangle"]

<Size>::= <Large> | <Small>

<Large>::= Any shape whose perimeter , or length in case of a line, is greater than 100

<Small>::= Any shape whose perimeter, or length in case of a line, is less or equal than

<Even>::= <Yes> | <No>

<Yes>::= Means that the product of the parameters is even

<No>::= Means that the product of the parameters is odd

<Shape Parameters>::= <Line> | <Ellipse> | <Triangle> | <Quadrilateral>

<Line>::=, <Length>

<Ellipse>::= <Minor Axis>, <Major Axis>

<Triangle>::= <Side 1>, <Side 2>, <Side 3>

<Quadrilateral>::= <Side 1>, <Side 2>, <Side 3>, <Side 4>

<Length>::= An integer number ϵ [1, 4095]

<Minor Axis>::= An integer number ϵ [1, 4095]

<Mayor Axis>::= An integer number ϵ [1, 4095]

<Side x>::= An integer number ϵ [1, 4095]

The usual geometric definitions must be used in deciding the test results for a given <Shape Description>. Table 2 provides some example of inputs and their corresponding answers.

Table 2 Examples

Input	Result	Explanation
"Equilateral,Large,Yes,100,100,100"	"Yes"	The shape has three equal sides → Equilateral triangle, the sum of the sides is greater than 100 and the product of the sides is even
"Square,Small,No,5,5,5"	"No"	The parameter part has only three sides, a square is described by four equal sides
"Rectangle, Large, Yes, 24, 30, 24, 30	"Yes"	The figure has four sides and opposite sides are equal, the sum of the sides is greater than 100 and the product of the sides is even
"Line,Small,No,2"	"No"	The length of the line is even

Appendix B. Marginal cost of detecting a new fault

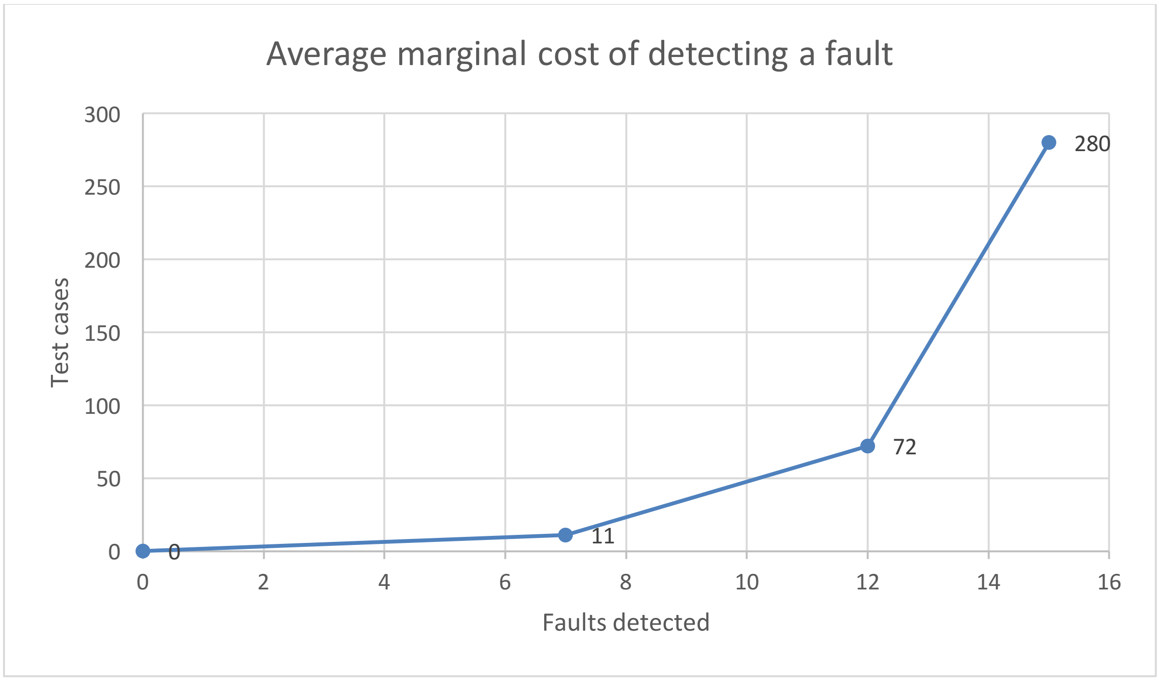
The marginal cost of detecting a failure is the number of additional test cases required to identify it.

The marginal cost is usually represented by a curve that has the cumulative number of test cases in the ordinate (y) axis and the number of cumulative faults in the abscissa (x) axis.

In the example below we see the results of executing three different test suites generated from the same parameters and values. In this case, the total number of faults detected by all pairs will include the total number of faults detected by all singles as well as its test cases and those of the all triples will include the values of the all pairs.

Table 3 Example

Suite	Faults detected	Test cases employed	Average
All single	7	11	$1.57 \frac{test\ cases}{fault}$
All pairs	12	72	$12.2 \frac{test\ cases}{fault}$
All triples	15	280	$69.3 \frac{test\ cases}{fault}$


Average marginal cost of detecting a fault
Test cases
Faults detected