Prof. Dr. A. Podelski,
Dr. B. Westphal

Sommersemester 2017

Softwaretechnik/Software Engineering

http://swt.informatik.uni-freiburg.de/teaching/SS2017/swtvl

Exercise Sheet 1

Early submission: Wednesday, 2017-05-03, 12:00 Regular submission: Thursday, 2017-05-04, 12:00

Exercise 1 – Metrics (10)

1.1. Lines of Code Metrics

Consider the following lines of code metrics:

- LOC_{tot} = Total number of lines of code
- LOC_{ne} = Number of non-empty lines of code
- LOC_{pars} = Number of lines of code that that do not consist entirely of comments or non-printable characters.

(i) Calculate the value of the LOC metrics for the Java program in the file `MyQuickSort.java` accompanying this exercise sheet. (2)

(ii) The LOC metrics are often used as derived measure for the complexity or effort required to develop the code being measured.

In particular the family of LOC metrics is notorious for being subvertible. If a metric is subvertible, its value can be manipulated to increase it arbitrarily while preserving the same program semantics. I.e., for every program, there always exists a semantically equivalent program (that performs the same computation, and thus should have needed roughly similar effort to develop) but has substantially different metric values. Convince yourself of this claim for the case of LOC_{pars}: give two semantically equivalent programs with substantially (at least an order of magnitude) different metric values. Can you recognize a pattern that allows you to increase the value of the LOC_{pars} metric to any number you wish? (2)

(iii) What is the value of the LOC metrics for the largest method you have programmed? Describe briefly what the method does, the program to which it belongs and in what programming language. (1)

1.2. Cyclomatic Complexity

Consider the *cyclomatic complexity* or *McCabe* metric.

(i) Construct the control flow graph (CFG) for the method `quickSort` of the class `MyQuickSort` in the file accompanying this sheet and calculate the value of its cyclomatic complexity as shown in the example of Fig. 1. (4)

(ii) In the example, we introduced additional, auxiliary nodes to the control flow graph that serve as junction points for control paths, see e.g., the round node directly after node number 5. Another possibility of constructing the CFG would be to directly connect the nodes representing program locations. In our example, there would be a direct edge from node 5 to node 6 and from node 8 to node 6.

Does this choice of CFG construction alter the value of the cyclomatic complexity metric? Justify your answer. (1)

For program:

```
1  void insertionSort(int[] array) {
2    for (int i = 2; i < array.length; i++) {
3      tmp = array[i];
4      array[0] = tmp;
5      int j = i;
6      while (j > 0 && tmp < array[j-1]) {
7        array[j] = array[j-1];
8        j--;
9      }
10     array[j] = tmp;
11   }
12 }
```

Cyclomatic complexity is defined for graph G corresponding to the program as

$$v(G) = e - n + p$$

Number of edges: $e = 11$
Number of nodes: $n = 6 + 2 + 2 = 10$ (nodes are marked with the corresponding line numbers)
External connections: $p = 2$

$v(G) = 11 - 10 + 2 = 3$

Corresponding graph G

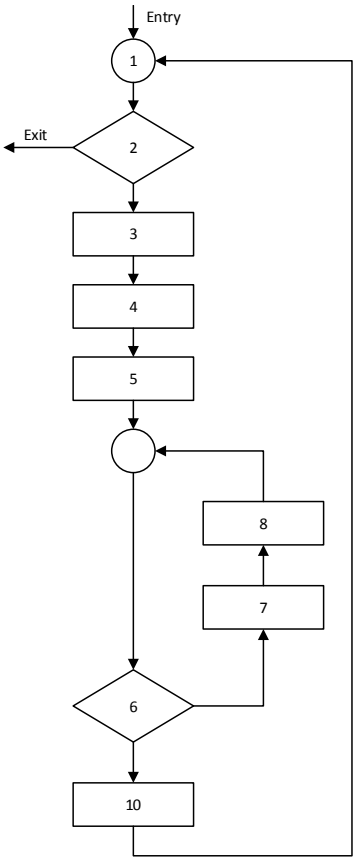

Figure 1: Example of the calculation of cyclomatic complexity