静态代码缺陷预测分析测试用例库

测试用例位置

目前有 5 个测试用例

- TestCase1 TestCase2 TestCase3 in `src/main/java/org/labsse/Supplement.java`
- TestCase4 in `src/main/java/org/labsse/QuickSort.java`
- TestCase5 in `src/main/java/org/labsse/HeapSort.java`

代码维度说明

综述

在静态代码分析当中，代码维度(Metrics)的精准定义和计算有较为重要的意义，可以说代码维度定义的好坏和相应计算方式的准确性在很大程度上决定了我们的项目最终能否取得有价值的成果。设想如果将大量的低质量，甚至堆错误的数据丢给机器学习模型去训练、预测，即使是使用再强大完备的训练算法，最终得到模型也可能不具备太强的说服力。因此对代码维度的研究和探索，在整个项目周期中都有着重要的价值。

维度说明

目前项目组共定义出 39 个代码度量维度。这些维度绝大多数来自于NASA MDP数据集，而MDP数据集中对代码维度的定义是有其局限性的，一些论文也指出部分维度的定义存在种种问题。由于目前网络上缺少统一的，对每个维度的标准化定义，同时也为了统一组员之间计算代码维度的规则，现将维度定义列举如下。

目前的所有维度可以分为三大类： `LOC` 、 `McCabe` 和 `Halstead`

`LOC`：(line of code) 与代码行数相关的维度

`McCabe`：与程序控制流程图相关的维度

`doc/与程序控制流程图有关维度的计算方法资料1-McCabe.pdf`

`Halstead`：与代码操作符数量、操作数数量相关的维度
https://en.wikipedia.org/wiki/Halstead_complexity_measures

分类介绍如下:

LOC (7个)

- NUMBER_OF_LINES
  方法的总行数
- LOC_BLANK
  空行数
- LOC_COMMENTS
  纯注释的行数
- LOC_EXECUTABLE
  纯代码行数
- LOC_CODE_AND_COMMENT
  同时有代码和注释的行数
- LOC_TOTAL = LOC_EXECUTABLE + LOC_CODE_AND_COMMENT
- PERCENT_COMMENTS = (LOC_COMMENTS + LOC_CODE_AND_COMMENT) / (LOC_COMMENTS + LOC_CODE_AND_COMMENT + LOC_EXECUTABLE)

McCabe (20个)

- BRANCH_COUNT
  程序流程图中的 分支边 的数量。 分支边 是指从决策点（do、if、while、for、switch语句节点）分出去的边
- CALL_PAIRS
  代码中出现的Java方法调用的数量
- CONDITION_COUNT
  条件覆盖数，在条件语句中除了字面'true'/'false'以外的condition符号总数+1再乘以2。condition由"!"非, "||"或, "&&"与组成。一个条件语句的分析单位指的是do、for、if、while结构中的判断执行表达式。
- CYCLOMATIC_COMPLEXITY
  环复杂度，由程序控制流程图(CFG)计算得出(参考https://en.wikipedia.org/wiki/Control_flow_graph)，计算公式为 `V(G) = Edge-Node+2` ，Edge:CFG中边的数量 Node:CFG中节点的数量 V(G):环复杂度
- CYCLOMATIC_DENSITY
  环复杂密度，CYCLOMATIC_COMPLEXITY / LOC_TOTAL
- DECISION_COUNT 决策数量，代码中出现的决策选择的数量。当出现do语句、for语句、if语句、while语句时，每出现一处，DECISION_COUNT +2

- DECISION_DENSITY
  决策密度，DESIGN_COMPLEXITY/ CYCLOMATIC_COMPLEXITY

- DESIGN_COMPLEXITY
  模块设计复杂度，对程序流程图进行剪枝，去掉不包含方法调用的节点和与之相关联的结构，计算剪枝后图的CYCLOMATIC_COMPLEXITY.详细解释请参考 doc/与程序控制流程图有关维度的计算方法资料1-McCabe.pdf 中 7.4 节对 module design complexity 的定义

  注:经讨论，pdf中对该条定义的结构式不够合理，计算的时候请参考7.4节截图

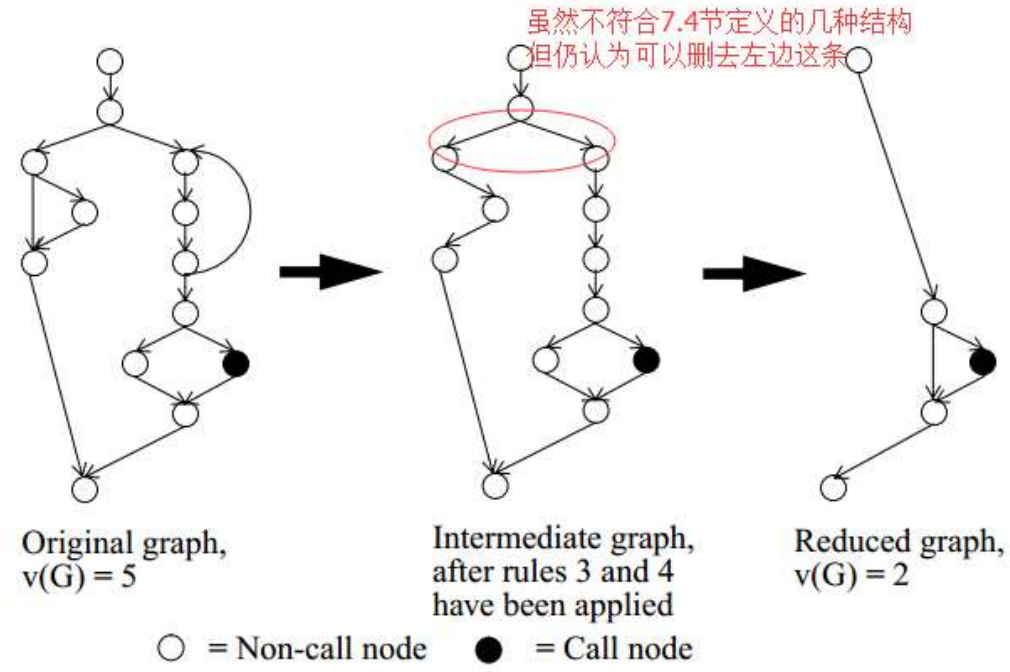

Figure 7-3. Design reduction example.

- DESIGN_DENSITY
  模块设计复杂密度，DESIGN_COMPLEXITY/ CYCLOMATIC_COMPLEXITY

- EDGE_COUNT
  程序流程图中边的数量

- ESSENTIAL_COMPLEXITY 基本复杂度，对程序流程图进行剪枝，删除图中所有结构化代码控制拓扑结构，包括if-else、if无else、switch-case、for、while、do和连续三个入度出度均为1的节点连接组成的直线结构，计算剪枝后图的CYCLOMATIC_COMPLEXITY.详细解释请参考 doc/与程序控制流程图有关维度的计算方法资料1-McCabe.pdf 中 10 节对 essential complexity 的定义

- ESSENTIAL_DENSITY
  基本复杂密度，ESSENTIAL_COMPLEXITY/ CYCLOMATIC_COMPLEXITY

- PARAMETER_COUNT 模块（方法）传入参数的数量，非static方法会计算隐式this参数，static方法不会计算此参数 模块中对类变量的直接引用相当于this.类变量名称，暂

未将这样的引用计算在内

- GLOBAL_DATA_COMPLEXITY 全局数据复杂度，对程序流程图进行剪枝，删除不包含方法参数引用的结构化拓扑结构，后计算结果的CYCLOMATIC_COMPLEXITY.详细解释请参考 doc/与程序控制流程图有关维度的计算方法资料1-McCabe.pdf 中 11.3 节对 data complexity 的定义

- GLOBAL_DATA_DENSITY
  全局数据复杂密度，GLOBAL_DATA_COMPLEXITY/CYCLOMATIC_COMPLEXITY

- MAINTENANCE_SEVERITY 维护严重度，ESSENTIAL_COMPLEXITY/ CYCLOMATIC_COMPLEXITY

- MODIFIED_CONDITION_COUNT
  修正条件数量，在条件语句中除了字面'true'/'false'以外的condition总数

- MULTIPLE_CONDITION_COUNT
  多重条件数，在条件语句中包含了字面'true'/'false'的condition总数

- NODE_COUNT
  程序流程图的节点数量

- NORMALIZED_CYCLOMATIC_COMPLEXITY
  规范化的圈复杂度，CYCLOMATIC_COMPLEXITY/ NUMBER_OF_LINES

Halstead (12个)

- HALSTEAD_VOLUME
  $$V = N \times \log_2 \eta$$
  编码程序需要的最小比特
  N=N1+N2, n=n1+n2

- NUM_OPERANDS
  操作数总数N2

- NUM_OPERATORS
  操作符总数N1

- NUM_UNIQUE_OPERANDS
  唯一操作数总数n2

- NUM_UNIQUE_OPERATORS
  唯一操作符总数n1

- HALSTEAD_DIFFICULTY

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$$ 程序困难度

- HALSTEAD_EFFORT

$$E = D \times V$$

预计人力成本

- HALSTEAD_CONTENT
HALSTEAD内容长度，等于HALSTEAD_VOLUME/ HALSTEAD_DIFFICULTY

- HALSTEAD_ERROR_SET
错误集数量，等于HALSTEAD_VOLUME/3000

- HALSTEAD_LENGTH

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

程序代码长度

- HALSTEAD_LEVEL
程序可理解程度 1/ HALSTEAD_DIFFICULTY

- HALSTEAD_PROGRAM_TIME

$$T = \frac{E}{18} \text{ seconds}$$

程序实现耗时

关于定义的改进

由于目前维度的定义和计算方式还存在不足，大家若在计算的过程中若遇到困难，或者对某些维度的定义、计算方式等有疑问，还请大家指出， **我们共同讨论，共同完善**

Appendix of Metrics

e.g.

```
public class Main {
    /**
     * This is an example function.
     */
```

```
    public int subStrCount(String parentStr, String subStr){
        //count
        int count = 0;
        //int
        int index = 0;
        while (true) {
            index = parentStr.indexOf(subStr, index + 1);
            if (index > 0) {
                count++;
            } else {
                break;
            }
        }

        return count; //I am comment 1;
    }
}
```

以下结果极可能有误，仅供参考！

LOC_BLANK	0.0
BRANCH_COUNT	4.0
CALL_PAIRS	1.0
LOC_CODE_AND_COMMENT	1.0
LOC_COMMENTS	6.0
CONDITION_COUNT	4.0
CYCLOMATIC_COMPLEXITY	3.0
CYCLOMATIC_DENSITY	0.1875
DECISION_COUNT	4.0
DECISION_DENSITY	1.0
DESIGN_COMPLEXITY	2.0
DESIGN_DENSITY	0.6666666666666666
EDGE_COUNT	14.0
ESSENTIAL_COMPLEXITY	1.0
ESSENTIAL_DENSITY	0.0
LOC_EXECUTABLE	16.0
PARAMETER_COUNT	3.0
GLOBAL_DATA_COMPLEXITY	2.0
GLOBAL_DATA_DENSITY	0.6666666666666666
HALSTEAD_CONTENT	30.859101574640846
HALSTEAD_DIFFICULTY	6.439024390243903
HALSTEAD_EFFORT	1279.450293483741
HALSTEAD_ERROR_SET	0.06623416923337548
HALSTEAD_LENGTH	32.0
HALSTEAD_LEVEL	0.1553030303030303
HALSTEAD_PROGRAM_TIME	71.08057186020784
HALSTEAD_VOLUME	198.70250770012643
MAINTENANCE_SEVERITY	0.3333333333333333
MODIFIED_CONDITION_COUNT	2.0
MULTIPLE_CONDITION_COUNT	3.0
NODE_COUNT	13.0
NORMALIZED_CYCLOMATIC_COMPLEXITY	0.13636363636363635
NUM_OPERANDS	16.0

NUM_OPERATORS	16.0
NUM_UNIQUE_OPERANDS	41.0
NUM_UNIQUE_OPERATORS	33.0
NUMBER_OF_LINES	22.0
PERCENT_COMMENTS	0.30434781312942505
LOC_TOTAL	17.0