Assignment 10: Software Metrics

Christian Møldrup Legaard

April 2020

1 Introduction

In today's lecture you learned about various software metrics which can be used as a measure for the quality of the code. While several such measures exists, today's exercise deals with cyclomatic complexity, that is the number of independent paths in your program.

2 Exercises

Question 1: Cyclomatic-Complexity

(a) Draw a *control flow diagram* (cfd) of an if-else statement and a while loop.

(b) Draw the cfd for the following program:

```
if(a || b)
{
    doA();
}
else
{
    doB();
}
```

(c) Describe how the diagrams are related to the cyclomatic complexity of a program.

Question 2: Complexity and design

A small toy-example is included in the handout. In its current form, the program contains a lot of branching statements which leads to a increase in complexity. Worse still, is that a lot of this complexity is contained within a single function, making it harder to comprehend and extend.

An important note here is that complexity can also be calculated for part of a program such as a function. This is useful to identify functions that "does too much" at the same time, which may then be refactored into several simpler and more cohesive functions.

A number of tool exists for automatically calculating software metrics from source code. One of these is Metrix++[1], which is written in Python. At the time of writing, the tool does not support Python 3.X, however i have created a fork of this which does[2]

To install:

- Clone the **fork**
- Open a shell and change directory into the cloned folder
- pip install .

[1] `https://github.com/metrixplusplus/metrixplusplus`
[2] `https://github.com/clegaard/metrixplusplus`

- verify the installation by calling "metrixpp –help"

If there is any issues with installing or running the tool, simply skip the parts where it is used.

(a) Examine the program to determine the complexity of the main and *Customer::serveDish* function. Here it may help to draw the cfd.

(b) Use the Metrix++ program to calculate the complexity. This is done by changing directory to where have stored the example program, then run:

```
metrixpp collect --std.code.complexity.cyclomatic
metrixpp view
```

The output should be something like:

```
./:: info: Overall metrics for 'std.code.complexity:cyclomatic' metric
        Average        : 2.0
        Minimum        : 0
        Maximum        : 11
        Total          : 14.0
        Distribution   : 7 regions in total (including 0 suppressed)
          Metric value : Ratio : R-sum : Number of regions
                     0 : 0.714 : 0.714 : 5      ||||||||||||||||||...
                     3 : 0.143 : 0.857 : 1      ||||||||||||||
                    11 : 0.143 : 1.000 : 1      ||||||||||||||

./:: info: Directory content:
        Directory      : build
        File           : example_a.cpp
```

(c) Refactor the Customer by making it an abstract base class and implement 3 new classes which implement it, *Vegan*, *Vegetarian*, and *Omnivore*. This will eliminate the need for several of the if statements.

Compute the complexity again either by hand or using the tool.