e-Portfolio Activities for unit 9

Activity 1

How relevant is the cyclomatic complexity in object oriented systems? Which alternative metrics do you consider to be more reflective of the complexity of a piece of code, in comparison to the number of independent paths through a program? Support your response using reference to the related academic literature.

Cyclomatic complexity indicates the number of possible execution paths through a piece of code. The higher this number is, the more complex the code is and the harder it is to understand. When it comes to testing code, cyclomatic complexity is the deciding factor in how difficult it is to test code. In other words, cyclomatic complexity is a predictor for assessing the difficulty level in testing code. There are several metrics besides cyclometric complexity to calculate the code complexity and identify potential areas for improvement:

- Lines of Source Code
- Lines of Executable Code
- Coupling/Depth of Inheritance
- Maintainability Index
- Cognitive Complexity
- Halstead Volume
- Rework Ratio (Computing Department, 2022)

Activity 2

To what extent is cyclomatic complexity relevant when developing object-oriented code?

As mentioned in Activity 1, the cyclomatic complexity score indicates the difficulty of the code review. Maintenance and navigation also become more complicated when the number of cyclomatic complexity is high. However, for object-oriented programs, cyclomatic complexity results only refer to the basis of the conditional statements, as the calculation of the coupling between the objects is missing. Moreover, inheritance affects complexity but should not affect cyclomatic complexity. However, it is unclear whether the measure has ever considered the abstraction of complexity through polymorphism, such as inheritance, where there is an implicit decision point based on the actual type of a reference to resolve a function call. If inheritance were included, the complexity of a coding unit would no longer be a local measure, as it would depend on the externally determined number of possible types, which could change over time as the inheritance tree is changed (Schults, 2021a).

Activity 3

What is the cyclomatic complexity of the following piece of code?

There are basic rules to calculate the cyclomatic complexity. With the help of a graph (Figure 1) and a formula the cyclomatic complexity can be calculated.

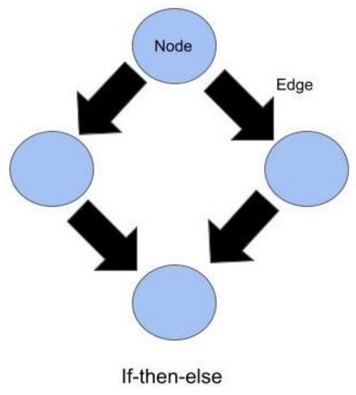

Figure 1: Flow Graph for calculating the cyclomatic complexity (Fetzner, 2021)

The formula is:

M = E – N + 2P

Where,

M = cyclomatic complexity

E = number of edges of the graph

N = number of nodes of the graph

P = number of connected components (Schults, 2021b)

```
public static string IntroducePerson(string name, int age)
{
    var response = $"Hi! My name is {name} and I'm {age} years old.";

    if (age >= 18)
        response += " I'm an adult.";

    if (name.Length > 7)
        response += " I have a long name.";

    return response;
```

}

The cyclomatic complexity calculation of the function above is more accessible understandable with the following graph (Figure 2)

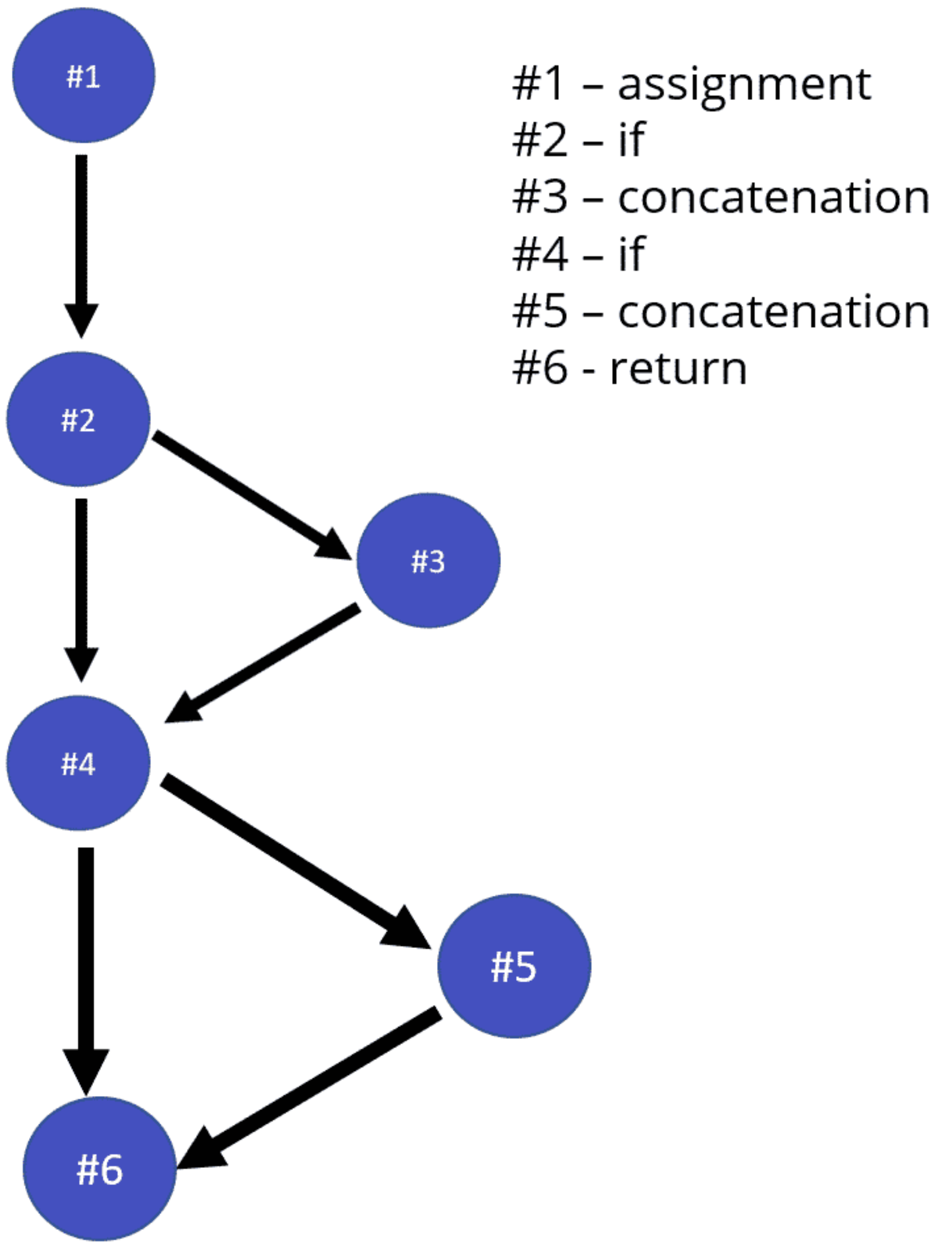

Figure 2: Flow Graph for calculating the function (Schults, 2021b)

Of course, the function will look different if values are inserted into the parameters. However, the code will always go to the first assignment. Without the if statements, the second node would be the absolute path because it would return the function. Nevertheless, the code will go to the first if statement and check whether the statement is true or false. Whether true or false, the code must choose the correct path, and two more nodes are assigned to the graph. Due to the second if statement, the code must choose which path is the right way. Finally, six nodes are added to the graph. In addition, the edges indicate which path the code can take.

After counting the edges and nodes, the formula looks like this:

M = 7 – 6 + 2 = 3

The result of the cyclomatic complexity is 3. (Schults, 2021b)

Source: Schultz, C. (2021) **Cyclomatic Complexity Defined Clearly, With Examples.** LinearB.

Activity 4

Extend the following program to test accuracy of operations using the assert statement.

```
# Python String Operations
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
```

```
print('str1 * 3 =', str1 * 3)
```

With assert statement:

```
#if condition returns True, then nothing happens!
#if condition returns False, AssertionError is raised:

str1 = 'Hello '
str2 ='World!'

# using +
assert str1 == 'Hello ', "x should be 'hello with a space character'"
print('str1 + str2 = ', str1 + str2)

# using *
assert str1 == 'Hello ', "x should be 'hello with a space character'"
print('str1 * 3 =', str1 * 3)
```

References:

Fetzner, W. (2021) What Is Code Complexity? What It Means and How to Measure It. Available from: https://linearb.io/blog/what-is-code-complexity/ [Accessed 16 August 2022].

Schults, C. (2021a) How to Reduce Cyclomatic Complexity: A Complete Guide. Available from: https://linearb.io/blog/reduce-cyclomatic-complexity/#:~:text=The%20more%20decision%20structures%20you,branch%20coverage%20of%20that%20function. [Accessed 15 August 2022].

Schults, C. (2021b) Cyclomatic Complexity Defined Clearly, With Examples. Available from: https://linearb.io/blog/cyclomatic-complexity/ [Accessed 16 August 2022].

Computing Department (2022) Packaging and Testing [Lecturecast]. OOP_PCOM7E June 2022 Object-Oriented System. University of Essex Online