Activity 1

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 measures the number of paths within code elements like functions or programs. It was created by Thomas J. McCabe Sr. in 1976. The higher the complexity, the more paths there are. Lower complexity is preferred (Schults, 2021). Schults (2021) states that the negative effects of high cyclomatic complexity in code are not inherent. It is possible to have code with a high cyclomatic complexity that is still easy to read and comprehend. However, in general, high cyclomatic complexity can indicate problems in the codebase or lead to future issues. One reason to reduce cyclomatic complexity is that it can contribute to cognitive complexity, which refers to how difficult it is to understand a piece of code. Cyclomatic complexity often drives up cognitive complexity, making the code harder to navigate and maintain. Additionally, high cyclomatic complexity makes code more difficult to test. More test cases are needed to test a block of code with high cyclomatic complexity thoroughly. Therefore, reducing cyclomatic complexity can make the process of writing tests easier.

A cyclomatic complexity statistic counts the number of independent routes in a piece of code. Despite some criticism, it continues to be frequently employed in the sector (Ebert et al. 2016). Ebert et al. (2016) added that cyclomatic complexity (CC) is a useful metric for analysing software. It predicts which parts of the code are more likely to have defects, helping to guide improvements in code quality. A high CC score indicates difficult-to-test code with a higher probability of defects. CC can be used with other measurements to create effective prediction models, making it a valuable tool for predicting criticality and performing static code analysis.

There is a divide between academia and day-to-day development in the software industry. Industry seeks simple metrics, like cyclomatic complexity (CC), while academia often criticises them. CC is easy to understand, collect, and interpret, making it useful for busy developers. Senior management also values metrics for accountability purposes. McCabe IQ offers an alternative to CC, providing answers about code security, quality, and testing. It appeals to management and solves a different problem than CC. Overall, the industry looks for metrics that can be communicated across levels of development stakeholders, making CC a fitting choice.

According to Imagix (N.D.), alternative measurements of the number of independent paths through a programme that have been suggested as being more indicative of the complexity of a piece of code include Woodward, Hennell and Hedley Knots Metric; Welker Maintainability Index; Chidamber and Kemerer Object Oriented Metrics; and Halstead Complexity Metrics.

- Halstead complexity metrics measure program module complexity directly from source code using operators and operands. They are strong indicators of code complexity and useful for assessing code quality in computationally-dense applications. During code development, these metrics should be considered to follow complexity trends and ensure maintainability. Introduced in 1977, they are one of the oldest measures of program complexity (Verifysoft, 2019).
- The Chidamber and Kemerer metrics suite is widely recognised as one of the most popular sets of object-oriented metrics. This suite includes six metrics, namely Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object Classes (CBO), Response for a Class (RFC), and Lack of Cohesion in Methods (LCOM)(Mcquillan & Power, N.D.). The CK metrics were developed to assess various aspects of an object-oriented design. However, only some of these metrics can accurately be measured from a UML class diagram. To measure CBO, RFC, and LCOM, for instance, implementation details such as the code within method definitions are required. Nonetheless, Mcquillan & Power (N.D.) have devised explanations that can estimate the values for these metrics based on the information provided in the UML class diagrams. Such measures can be useful, as they give both upper and lower bounds for metrics calculated later in the design or implementation process.
- The Maintainability Index (MI) is a helpful way to estimate how easy it will be to maintain code. It is calculated using formulas considering the lines of code and

McCabe and Halstead measurements. By measuring and tracking maintainability, we can avoid the risk of a system's code becoming less reliable over time, or "code entropy". MI can also indicate when it might be more cost-effective and less risky to rewrite the code instead of making further changes (Verifysoft, 2019).

According to Jaime (2021), code complexity measurement has four main advantages plus an additional one:

- Improved Tests: Programmers may be able to determine how many independent paths there are through a piece of code and how many of those paths need to be tested. Programmers can discover how many different ways to test by being aware of the number of code pathways. As a result, programmers can estimate the number of tests necessary to ensure the code is covered.
- Reduced Risk: The quality of the code a developer maintains, not the number of lines they've written (or updated), should be used to judge how competent a developer is. Given this, programmers can lower the danger of introducing minor or major flaws that could cause them to go bankrupt by simplifying their code.
- Lower Costs: Fewer defects can be detected and fixed when the danger of potential flaws is decreased. We decrease the possibility of introducing defects by lowering complexity. That permeates every phase of a software's lifecycle.
- Greater Predictability: We can develop more predictably by minimising software complexity. Based on this information, the company or organisation can more effectively determine its objectives and goals, particularly those directly related to the programme. Setting realistic budgets, predictions, and other purposes is simpler when this occurs.
- Understanding why a developer's code is regarded as complex has one last benefit: it aids in developing that developer.

Activity 2

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

As a software metric, Cyclomatic Complexity (CYC) assesses program complexity by counting the number of decisions in the source code. A higher count indicates more intricate code, making CYC crucial in limiting code complexity and determining the necessary test cases. Nonetheless, CYC can be a challenging software quality metric, making accurate calculations difficult. Therefore, it is essential to understand software quality metrics like Cyclomatic Complexity and its precise measurement (Britton, 2016).

According to Garg (2014), several methods are used to measure software complexity, including LOC, Halstead's measure, and McCabe's cyclomatic complexity. However, these methods must consider the interaction between object classes in object-oriented programming. To address this, an algorithm has been proposed that measures external class references. Measuring software complexity is crucial for minimising maintenance costs, ensuring quality, and reducing testing efforts.

Meeting the needs of complex software requires careful consideration of software complexity measurement. The line of code, Halstead's measure, and cyclomatic complexity are the other three proposed and investigated metrics. Cyclomatic complexity is one of the most popular and well-known metrics. Although cyclomatic complexity is very well-liked, it also helps with some of the issues found and displayed in a tabular format in this research endeavour. It simply determines complexity based on conditional statements and does not evaluate the connectivity between object classes like in object-oriented programming. Therefore, it is necessary to enhance the coupling-based theory of cyclomatic complexity (Ankita, N.D.).

Activity 3

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

```
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;
```

}

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

A code section's cyclomatic complexity can be calculated as the sum of all its linearly independent pathways. It is a software statistic that shows how complex a programme is. It is calculated using the program's Control Flow Graph. The graph's nodes represent the smallest possible collection of programme commands, and a directed edge between two nodes indicates whether the second command will necessarily come after the first (GeeksforGeeks, 2018).

In the code above, there are three possible paths through the code:

- The first path runs the if (age >= 18) statement but skips the if (name.Length > 7) statement.
- The second path carries out the if (name.Length > 7) but not the if (age >= 18) statement.
- The third path that runs the if (age >= 18) and if (name.Length > 7) statements.

As a result, the code's cyclomatic complexity is 3.

Activity 4

Extend the following program to test the 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)
```

Source: Progamiz. (n.d.) **Python Strings**.

Assertions are statements in Python that are used to set sanity tests when developing. When debugging code, assertions allow you to verify that the code is proper by seeing if a predetermined criterion still holds. The assertion condition should always be true unless your programme contains a bug. The assertion raises an exception and stops your program's execution if it turns out that the circumstance is incorrect (Ramos, 2022).

Assertions can be implemented in a programme using Python's built-in assert statement. For the programme to continue running, the assert statement in Python requires that an argument, such as a condition or an expression, be true. The assert statement will throw an AssertionError if the condition or expression is untrue, which stops the programme from running as usual (Study tonight, N.D.).

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

# using +
print('str1 + str2 = ', str1 + str2)
assert str1 + str2 == 'HelloWorld!'

# using *
print('str1 * 3 =', str1 * 3)
assert str1 * 3 == 'HelloHelloHello'
```

The assert statement has been used in the code above to verify the precision of the two-string operations. The first assert statement confirms that the outcome of adding str1 and str2 is "HelloWorld!" The second assert statement verifies that str1 * 3 produces the string "HelloHelloHello."

The programme will carry on running normally if the assert statements are true. However, the programme will end if any assert statements are untrue and an AssertionError exception is produced.

References:

Schults, C. (2021). *Cyclomatic Complexity Defined Clearly, With Examples | LinearB*. [online] Available at: https://linearb.io/blog/cyclomatic-complexity/#:~:text=Cyclomatic%20complexity%20is%20a%20metric [Accessed 9 Jul. 2023].

Ebert, C., Cain, J., Antoniol, G., Counsell, S. & Laplante,P. (2016). Cyclomatic Complexity. *IEEE Software*, 33(6), pp.27–29. doi:https://doi.org/10.1109/ms.2016.147.

www.imagix.com. (N.D.). *Software Quality Metrics Resources - Imagix*. [online] Available at: https://www.imagix.com/links/software-metrics.html [Accessed 9 Jul. 2023].

Verifysoft (2019). *Verifysoft → Halstead Metrics*. [online] Available at: https://www.verifysoft.com/en_halstead_metrics.html.

Mcquillan, J. & Power, J. (n.d.). *A Definition of the Chidamber and Kemerer Metrics suite for UML*. [online] Available at: https://mural.maynoothuniversity.ie/2391/1/JM_NUIM-CS-TR-2006-03.pdf [Accessed 9 Jul. 2023].

Verifysoft (2019). *Verifysoft → Maintainability Index*. [online] Available at: https://www.verifysoft.com/en_maintainability.html.

Britton, J. (2016). *What Is Cyclomatic Complexity?* [online] Available at: https://www.perforce.com/blog/qac/what-cyclomatic-complexity#:~:text=Cyclomatic%20complexity%20%28CYC%29%20is%20a%20software%20metric%20used.

Garg, A. (2014). *An approach for improving the concept of Cyclomatic Complexity for Object-Oriented Programming*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.1412.6216.

Jaime (2021). *An In-Depth Explanation of Code Complexity*. [online] Codacy | Blog. Available at: https://blog.codacy.com/an-in-depth-explanation-of-code-complexity/#:~:text=In%201976%2C%20Thomas%20McCabe%20Snr [Accessed 10 Jul. 2023].

Ankita (N.D.). *An approach for improving the concept of Cyclomatic Complexity for Object-Oriented Programming*. Available at: https://arxiv.org/ftp/arxiv/papers/1412/1412.6216.pdf.

GeeksforGeeks. (2018). *Cyclomatic Complexity - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/cyclomatic-complexity/.

Ramos, L. P. (2022). *Python's assert: Debug and Test Your Code Like a Pro – Real Python*. [online] realpython.com. Available at: https://realpython.com/python-assert-statement/#what-are-assertions [Accessed 10 Jul. 2023].

Study tonight (N.D.). *Python Assert Statement | Studytonight*. [online] Available at: https://www.studytonight.com/python/python-assert [Accessed 10 Jul. 2023].