e-Portfolio Activities:

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 is an essential software metric used to quantify the complexity of a program by examining the number of linearly independent paths through its source code. It helps in assessing the program's maintainability, understandability, and potential error-proneness8. This metric provides significant insights into the complexity of the control flow within a program, aiding developers and testers in evaluating code quality and efficiency (Bruneaux, N.D.).

While cyclomatic complexity serves as a valuable tool, it is essential to recognize that it is not the sole indicator of software complexity and quality. Several alternative metrics offer a more holistic view of code complexity compared to just counting independent paths within a program (Bruneaux, N.D.). These metrics complement cyclomatic complexity and provide a more comprehensive understanding of software quality.

One such alternative metric is the McCabe IQ Complexity Metric, which encompasses a broader range of software complexity analysis techniques8. This metric considers factors like module design complexity, integration complexity, object integration

complexity, actual complexity, realizable complexity, essential complexity, and data complexity to provide a more nuanced evaluation of software complexity and reliability. Moreover, qualitative factors such as readability, maintainability, testability, scalability, performance, robustness, and compatibility play a crucial role in assessing software quality alongside cyclomatic complexity (Bruneaux, N.D.). These qualitative aspects emphasize the human experience of code quality, focusing on simplicity, maintainability, and effectiveness over purely numerical metrics.

Therefore, while cyclomatic complexity remains a valuable metric for measuring software complexity, it should be used in conjunction with alternative metrics and qualitative assessments to provide a comprehensive evaluation of a piece of code's complexity and quality. This balanced approach, considering both quantitative and qualitative factors, enables developers to make informed decisions about code structure, testing strategies, and overall software maintainability.

References:

Bruneaux, T. (N.D.) *Cyclomatic complexity: Definition and limits in Understanding code quality*, *DX*. Available at: https://getdx.com/blog/cyclomatic-complexity/ (Accessed: 27 May 2024).

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

Cyclomatic complexity is relevant when developing object-oriented code as it provides insights into the complexity and potential challenges of the codebase. In object-oriented programming, the interrelationships between classes, methods, and objects can impact the overall complexity of the system. By considering cyclomatic complexity, developers can assess the maintainability, testability, and overall quality of their object-oriented code. High cyclomatic complexity values may indicate potential areas for refactoring or optimization to improve code readability and maintainability in object-oriented systems.

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;
}
```

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)
```

Answer:

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

# using +
result_concatenation = str1 + str2
print('str1 + str2 = ', result_concatenation)
assert result_concatenation == 'HelloWorld!', "Concatenation result is
incorrect"

# using *
result_repetition = str1 * 3
print('str1 * 3 =', result_repetition)
assert result_repetition == 'HelloHelloHello', "Repetition result is
incorrect"
```