e-Portfolio Activities for Unit 9

Activity 1

Cyclomatic complexity is a widely used metric for measuring the complexity of procedural code, focusing on the number of linearly independent paths through a program. However, object-oriented systems might not fully capture the complexity due to the additional layers introduced by inheritance, polymorphism, and encapsulation. Cyclomatic complexity primarily measures the control flow of a single method or function, but it does not account for the interactions between methods or classes.

Alternative metrics that might be more reflective of the complexity of object-oriented systems include:

Weighted Methods per Class (WMC): This metric measures a class's complexity by summing the complexity of its methods. It provides insight into the potential difficulty of maintaining a class.

Depth of Inheritance Tree (DIT): This metric indicates the level of inheritance in a class hierarchy. A deeper inheritance tree might indicate more complexity due to inherited behavior.

Coupling Between Object Classes (CBO): This metric measures how many other classes a class is coupled with. High coupling can indicate a higher complexity due to dependencies.

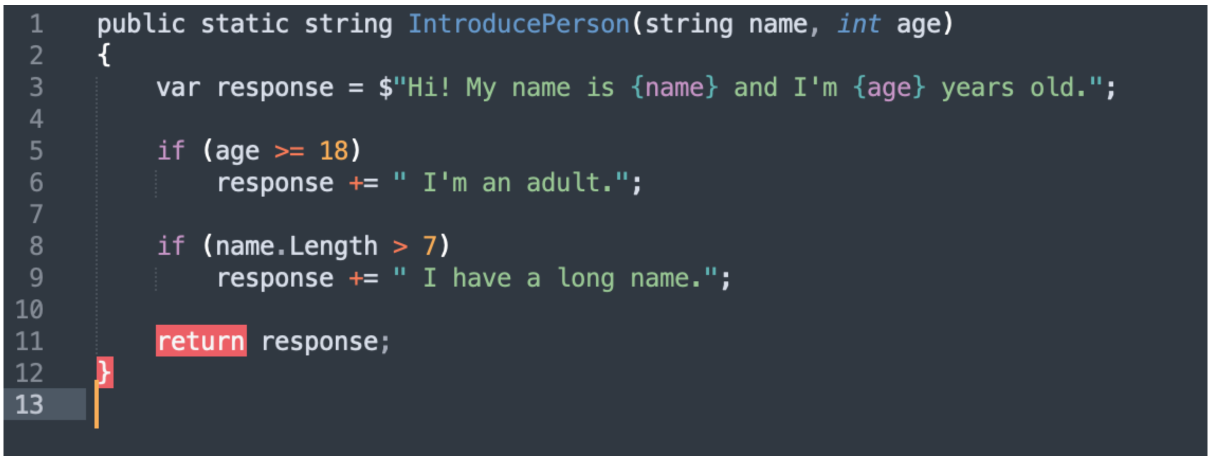
Lack of Cohesion in Methods (LCOM): This metric measures the degree of cohesion within a class. Lower cohesion indicates higher complexity due to the class having multiple responsibilities.

According to Basili et al. (1996), these object-oriented metrics provide a more comprehensive view of the complexity and maintainability of object-oriented software compared to cyclomatic complexity alone.

Activity 2

Cyclomatic complexity remains relevant when developing object-oriented code, but it should not be the sole metric used to gauge complexity. While it effectively measures the complexity of individual methods, it does not account for the interactions between objects, inheritance hierarchies, or polymorphic behavior inherent in object-oriented design. Cyclomatic complexity can still provide valuable insights into the complexity of the logic within methods and help identify areas that might require refactoring to improve readability and maintainability. However, it should be complemented with other object-oriented metrics to understand the system's complexity comprehensively.

Activity 3



```
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 of the given code is calculated by counting the number of decision points (if statements) and adding 1 to it.

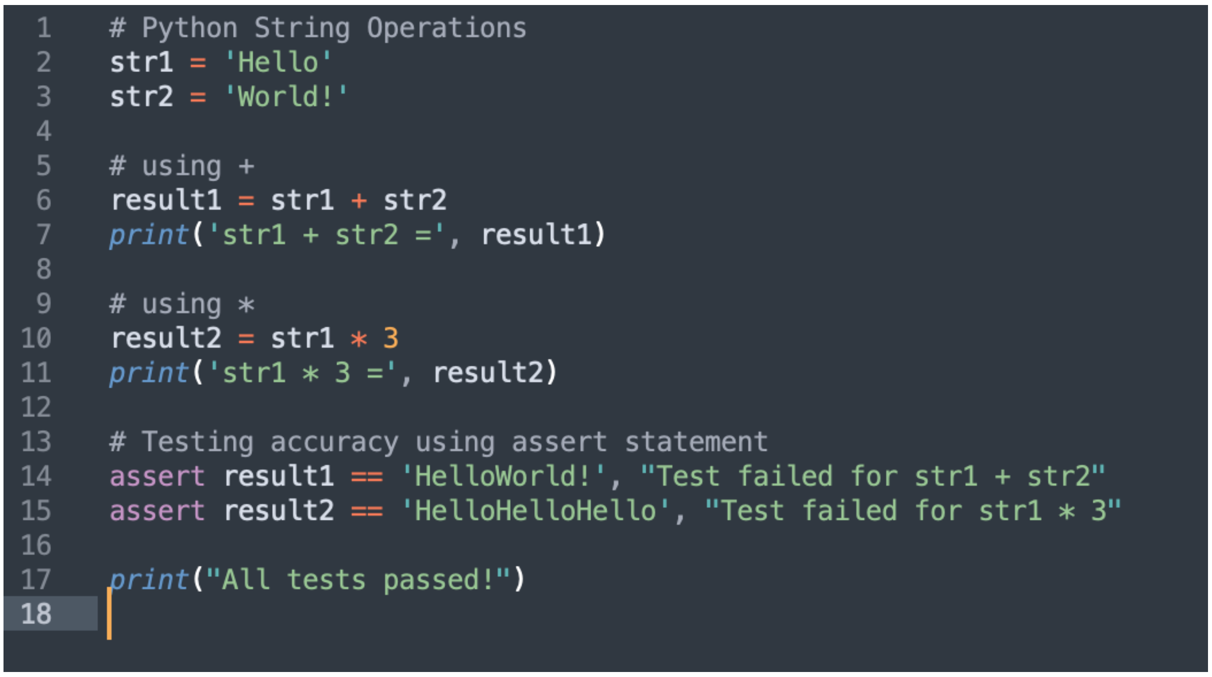
There are two if statements, each representing a decision point.

Cyclomatic Complexity = Number of decision points + 1

Cyclomatic Complexity = 2 + 1 = 3

Thus, the cyclomatic complexity of the provided code is 3.

Activity 4



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

# using +
result1 = str1 + str2
print('str1 + str2 =', result1)

# using *
result2 = str1 * 3
print('str1 * 3 =', result2)

# Testing accuracy using assert statement
assert result1 == 'HelloWorld!', "Test failed for str1 + str2"
assert result2 == 'HelloHelloHello', "Test failed for str1 * 3"

print("All tests passed!")

```

The assert statements are used to verify that the operations produce the expected results.

assert result1 == 'HelloWorld!' checks if the concatenation of str1 and str2 is correct.

assert result2 == 'HelloHelloHello' checks if the repetition of str1 three times is correct.

If any of the assertions fail, an error message will be displayed, indicating which test failed. If all assertions pass, "All tests passed!" is printed.

Reference:

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering, 22(10), 751-761.