Object-Oriented Metrics

Instructor: Mehroze Khan

Object-Oriented Metrics

- CK Metrics
  - Proposed by Chidamber and Kemerer
  - Class-based metrics
- LK Metrics
  - Proposed by Lorenz and Kidd
  - Class-based and operation-based
- MOOD Metrics
  - Proposed by Harrison, Counsell, and Nithi
  - Class-based

CK Metrics

- CK metrics, also known as **Chidamber and Kemerer metrics**, are a suite of object-oriented metrics designed to assess the **quality of software design**, particularly for object-oriented systems.
- These metrics help in identifying **design flaws** and predicting the **maintainability**, **reliability**, and **reusability** of a system.

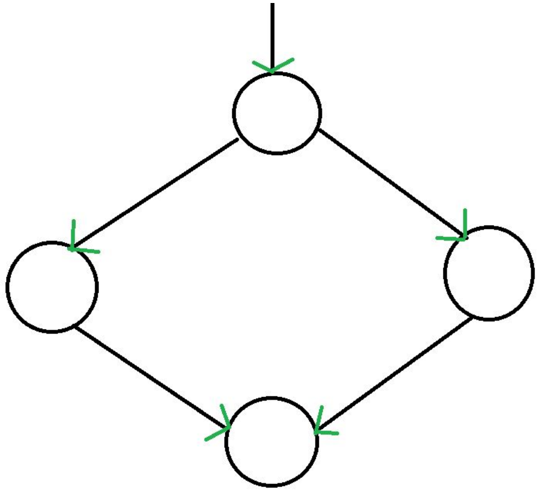
CK Metrics

- **Weighted methods per class (WMC)**
  - WMC is the **sum of complexities of all methods** within a class.
  - If every method has a complexity of 1, WMC is simply the count of methods in the class.
  - They do not specify the specific complexity metric to use (e.g., cyclomatic complexity).
  - High WMC indicates a **high level of functionality and complexity**, which may make a class harder to understand, test, and maintain.
  - WMC should be kept **low**.
  - Example: Suppose a Customer class in an e-commerce application has methods like addToCart, removeFromCart, checkout, and updateProfile. If each method is of equal complexity, and there are four methods, WMC = 4.

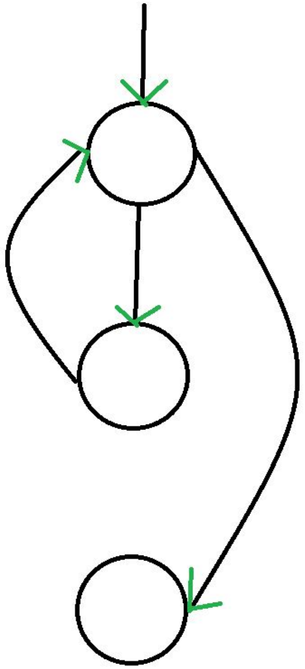
CK Metrics

- **Weighted methods per class (WMC)**
  - WMC is a predictor of how much **time** and **effort** is required to develop and maintain the class. The higher the value, the more effort required.
  - A large number of methods also means a greater potential impact on derived classes, since the derived classes inherit the methods of the base class.
  - Search for high WMC values to spot classes that could be restructured into several smaller classes.

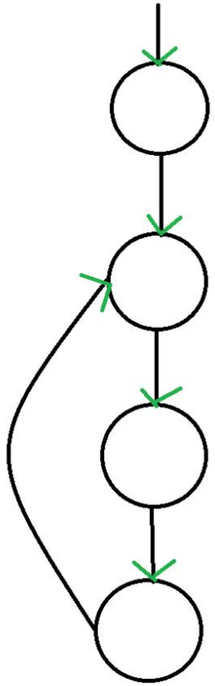
Control Flow Graphs



If-then-else



while



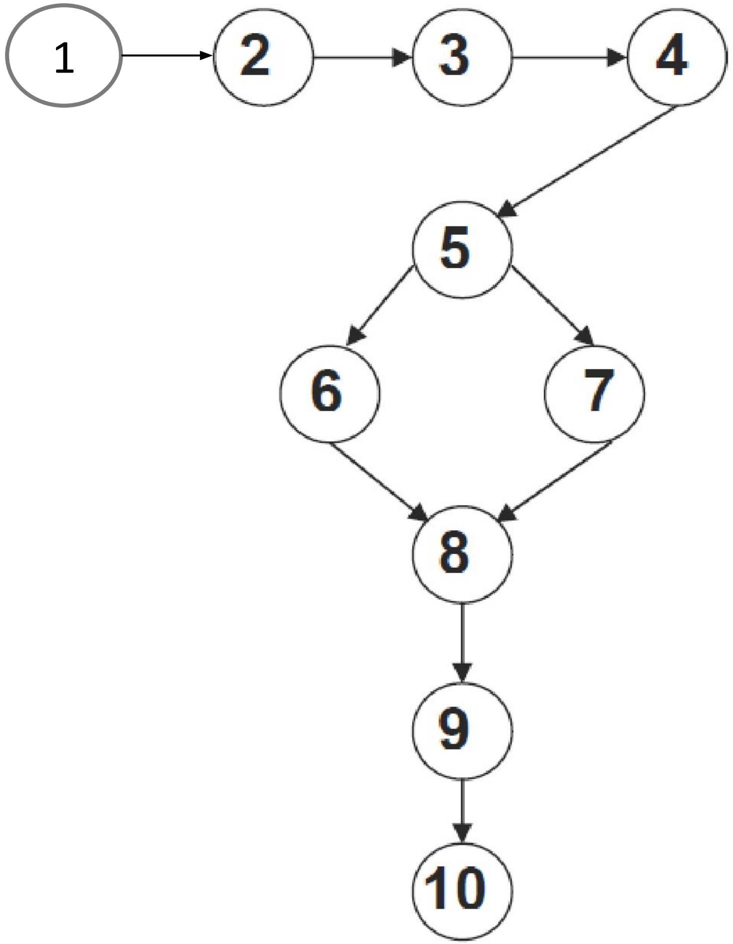
for

Control Flow Graphs

Program

1. *Program 'Simple Subtraction'*
2. *Input (x, y)*
3. *Output (x)*
4. *Output (y)*
5. *If $x > y$ then DO*
6. *$x - y = z$*
7. *Else $y - x = z$*
8. *EndIf*
9. *Output (z)*
10. *Output "End Program"*

Control Flow Graph



Cyclomatic Complexity

- **Cyclomatic Complexity** is the quantitative measure of the number of linearly independent paths in it.
- It is a software metric used to describe the complexity of a program.
- Complexity is computed as:
  - Cyclomatic complexity $V(G)$ for a flow graph G is defined as

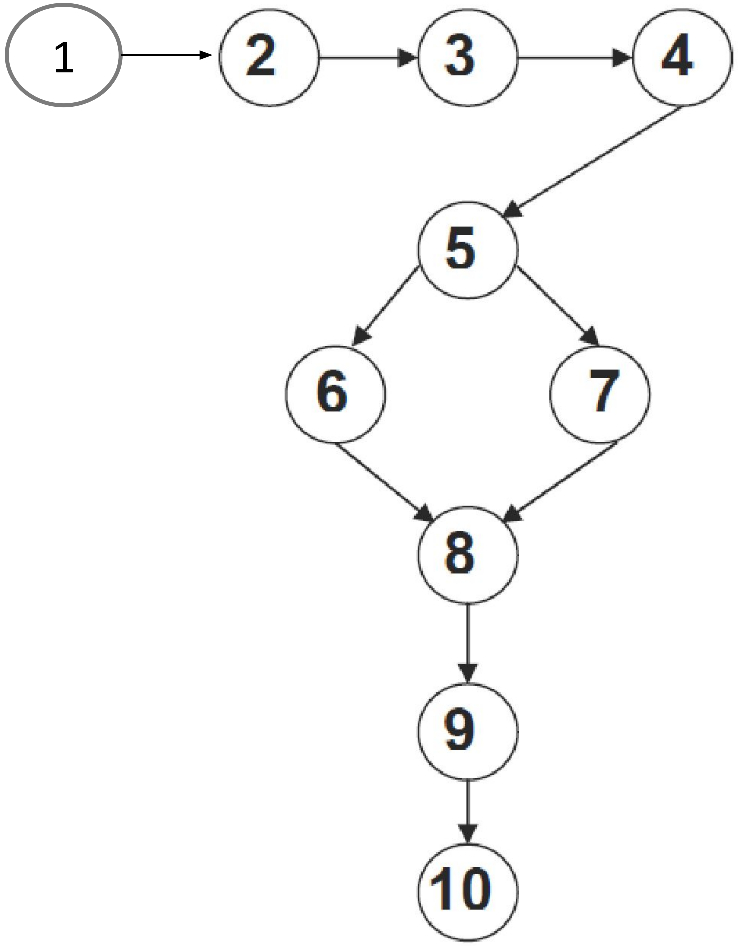
    $V(G) = E - N + 2$

    where E is the number of flow graph edges and N is the number of flow graph nodes.

Cyclomatic Complexity

- *Cyclomatic Complexity:*

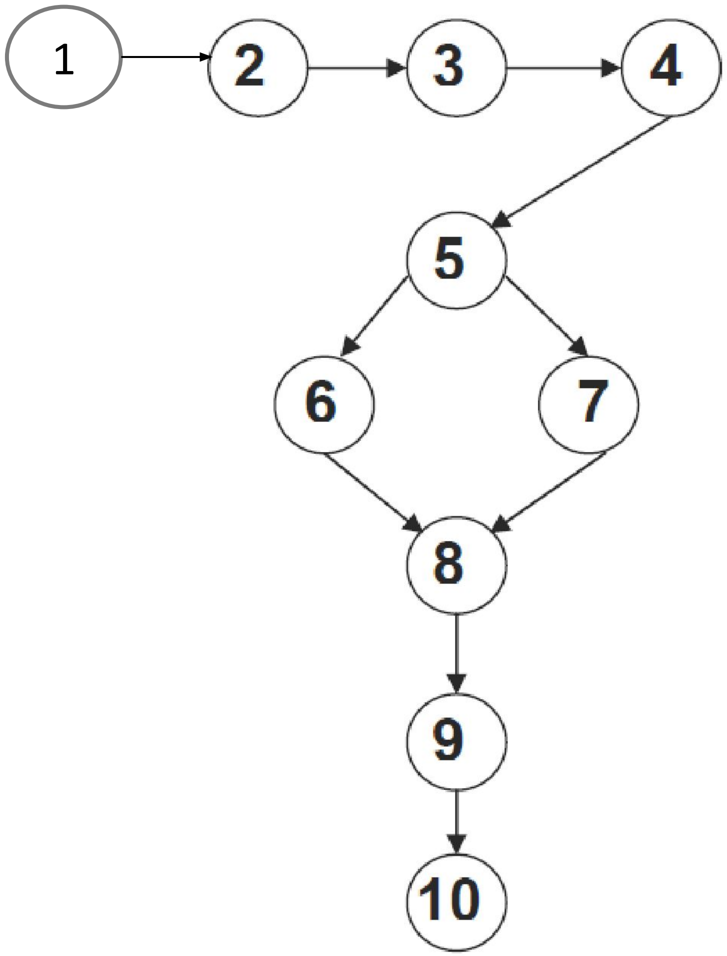
$V(G)$ = 10 edges − 10 nodes + 2 = 2.



White Box Testing

Control Flow Testing

- There are two independent paths:
  - Path 1: 1-2-3-4-5-6-8-9-10
  - Path 2: 1-2-3-4-5-7-8-9-10



CK Metrics

- **Depth of inheritance tree (DIT)**
  - DIT is the **length of the longest path** from a given class to the root class in the inheritance hierarchy.
  - As DIT **increases**, the lower classes in the hierarchy inherit a greater number of data and methods, thus making their **behavior** more difficult to understand and causing **testing** to require more effort.
  - A large DIT value implies greater **design complexity**, but also greater **reuse**.
  - Example: Consider a class hierarchy where Animal is the root class, Mammal inherits from Animal, Dog inherits from Mammal, and Bulldog inherits from Dog. Here, Bulldog has a DIT of 3.

CK Metrics

- **Number of children (NOC)**
  - NOC is the **number of immediate subclasses** inheriting from a particular class.
  - **Higher NOC** values suggest more **complexity**, as the class must support more inheriting classes.
  - High NOC can be a sign of **over-generalization** or it may indicate that a class is designed for extensibility.
  - Example: Suppose we have a Shape class, and three classes inherit from it: Circle, Square, and Triangle. Here, NOC for Shape is 3.

CK Metrics

- **Coupling between object classes (CBO)**
  - CBO measures the **number of other classes to which a given class is coupled**, either by using their methods or properties.
  - **High coupling** makes the code **less modular** and **harder to maintain or test**, as changes in one class may impact other classes.
  - Example: If a Customer class in a banking application calls methods from Account, Loan, and Transaction classes, CBO for Customer would be 3.

CK Metrics

- **Response for a class (RFC)**
  - The number of methods that can potentially be executed in response to a message received by an object of a given class.

$$RFC = M + R$$

  - M = number of methods in the class
  - R = number of remote methods directly called by methods of the class
  - As the RFC value increases, testing effort and design complexity also increase.
  - RFC should be kept low.

CK Metrics

- **Lack of cohesion in methods (LCOM)**
  - Lack of Cohesion in Methods (LCOM) measures **how closely related the methods in a class are**, based on the instance variables (attributes) they access.
  - One common approach to calculate LCOM is based on **method pairs**, which calculates cohesion by examining how frequently methods share attributes.

CK Metrics

- **Lack of cohesion in methods (LCOM)**
  - Identify pairs of methods.
  - For each pair:
    - Count it as "connected" if they access at least one common attribute.
    - Count it as "disconnected" if they do not share any attributes.
  - Calculate LCOM as follows:

LCOM = Number of disconnected method pairs – Number of connected method pairs

  - If LCOM is positive, it indicates a lack of cohesion (i.e., methods are disconnected).
  - If LCOM is zero or negative, the class has good cohesion.