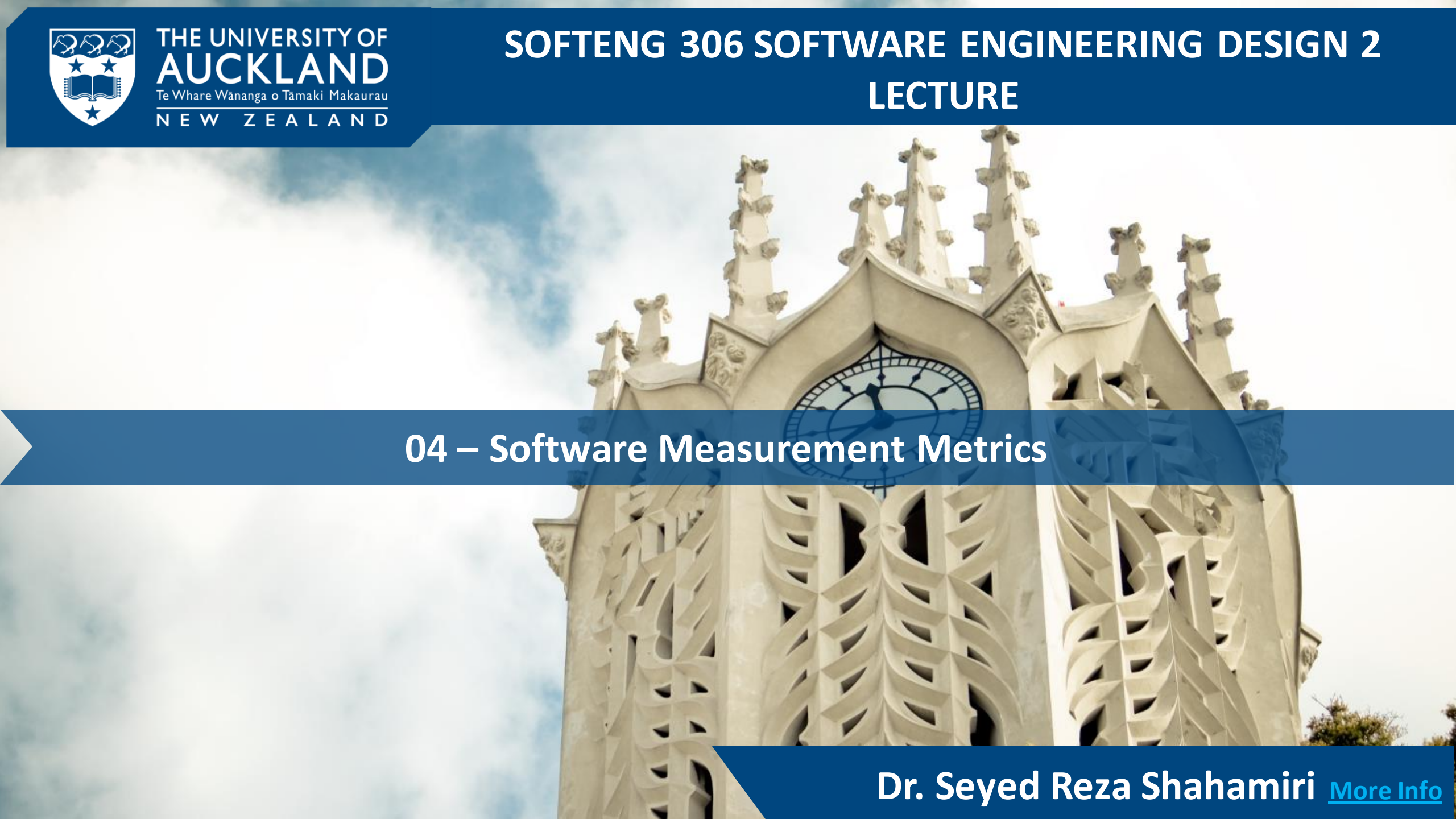


THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

SOFTENG 306 SOFTWARE ENGINEERING DESIGN 2 LECTURE

04 – Software Measurement Metrics

Dr. Seyed Reza Shahamiri More Info

Software Measurement

- Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system, or process.
- By comparing these values to each other and to the standards that apply across an organization, you may be able to draw conclusions about the quality of software, or assess the effectiveness of software processes, tools, and methods.

Software Measurement

Software Metrics

- Using software measurement, a system could ideally be assessed using a range of metrics and, from these measurements, a value for the quality of the system could be inferred.
- A software metric is a characteristic of a software system, system documentation, or development process that can be objectively measured.

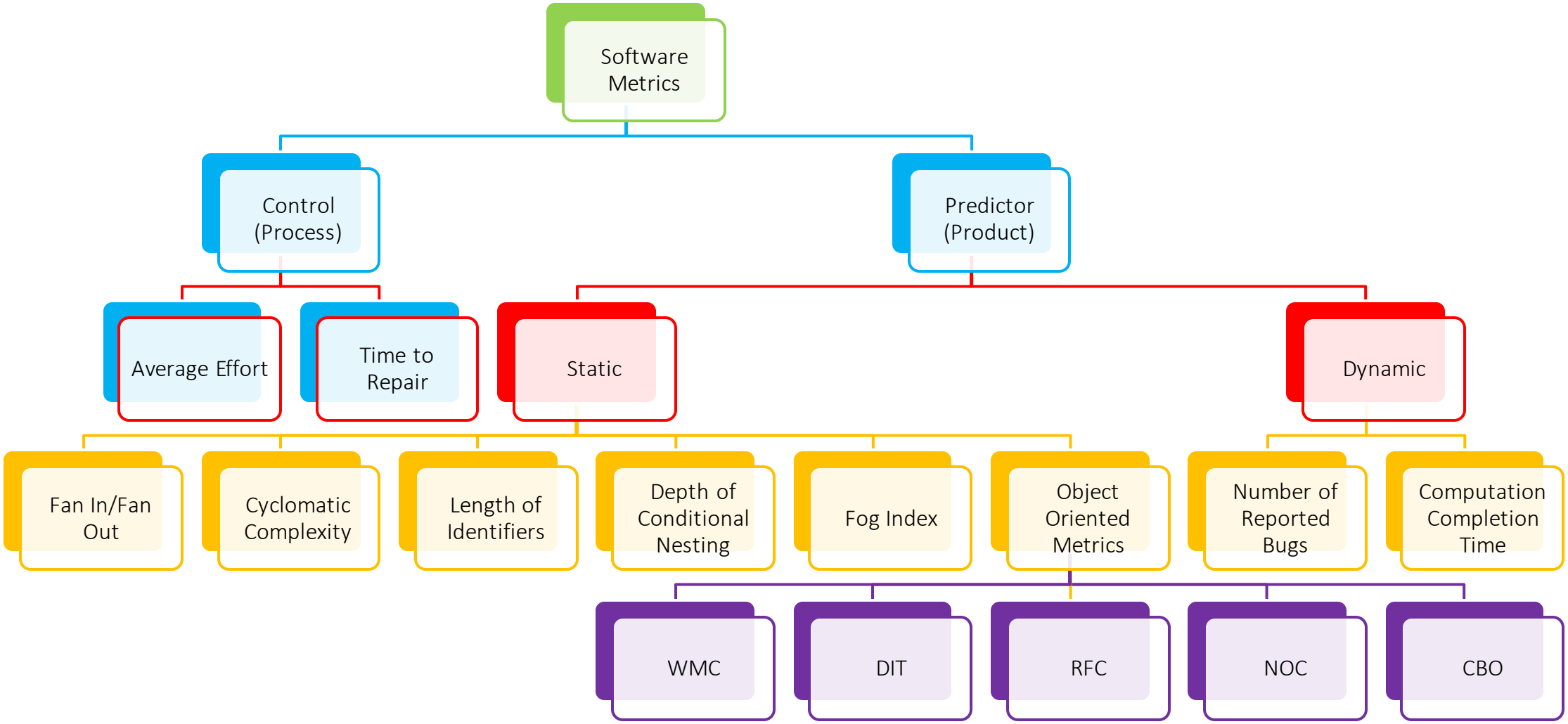
Software Measurement

Software Metrics

- Examples of metrics:
  - The size of a product in lines of code
  - The number of reported faults in a delivered software product
  - The number of person-days required to develop a system component
- Software metrics may be either control metrics or predictor metrics.

Software Measurement

Software Metrics



Software Measurement

Software Metrics

Control Metrics

- Control metrics or process metrics support process management and are usually associated with software processes.
  - Examples of control or process metrics are the average effort and the time required to repair reported defects.
- Managers use process measurements to decide if process changes should be made.

Software Measurement

Software Metrics

Predictor Metrics

- Predictor metrics help you predict characteristics of the software and are associated with the software itself and are sometimes known as product metrics.
  - Examples of predictor metrics are:
    - The Cyclomatic Complexity of a module
    - The average length of identifiers in a program
    - The number of attributes and operations associated with object classes in a design.
- Managers use predictor metrics to help estimate the effort required to make software changes.

Software Measurement

Software Metrics

Product Metrics

- Product metrics are predictor metrics that are used to measure internal attributes of a software system.
- Product metrics fall into two classes:
  1. Dynamic metrics
  2. Static metrics

Software Measurement

Software Metrics

Product Metrics

Dynamic metrics:

- They are collected by measurements made of a program in execution.
- These metrics can be collected during system testing or after the system has gone into use.
- Dynamic metrics help to assess the efficiency and reliability of a program.
- An example might be the number of bug reports or the time taken to complete a computation.

Software Measurement

Software Metrics

Product Metrics

Static metrics:

- They are collected by measurements made of representations of the system, such as the design, program, or documentation.
- Static metrics help assess the complexity, understandability, and maintainability of a software system or system components.
- Examples of static metrics are the code size and the average length of identifiers used.

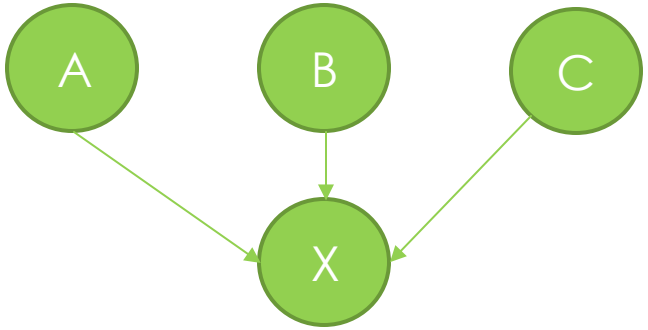
Software Measurement

Software Metrics

Static Metrics

Fan-in:

- Fan-in is a measure of the number of functions or methods that call another function or method (say X).
- To put it simple, Fan-in of function X is the number of other functions that call X.
- A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects.



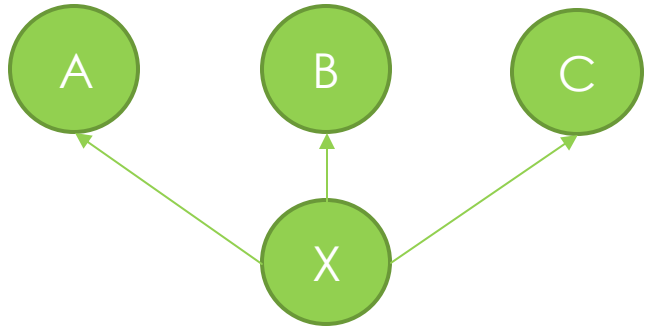
Software Measurement

Software Metrics

Static Metrics

Fan-out:

- Fan-out is the number of functions that are called by function X.
- A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
- It may also be an indication that X implements multiple functionalities that is a violation of Single Responsibility Principle.



Software Measurement

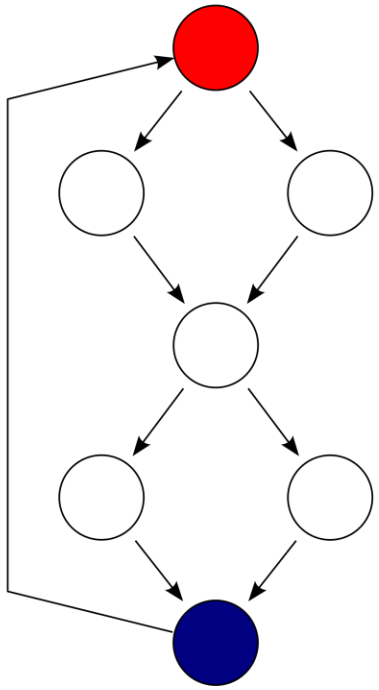
Software Metrics

Static Metrics

Cyclomatic complexity

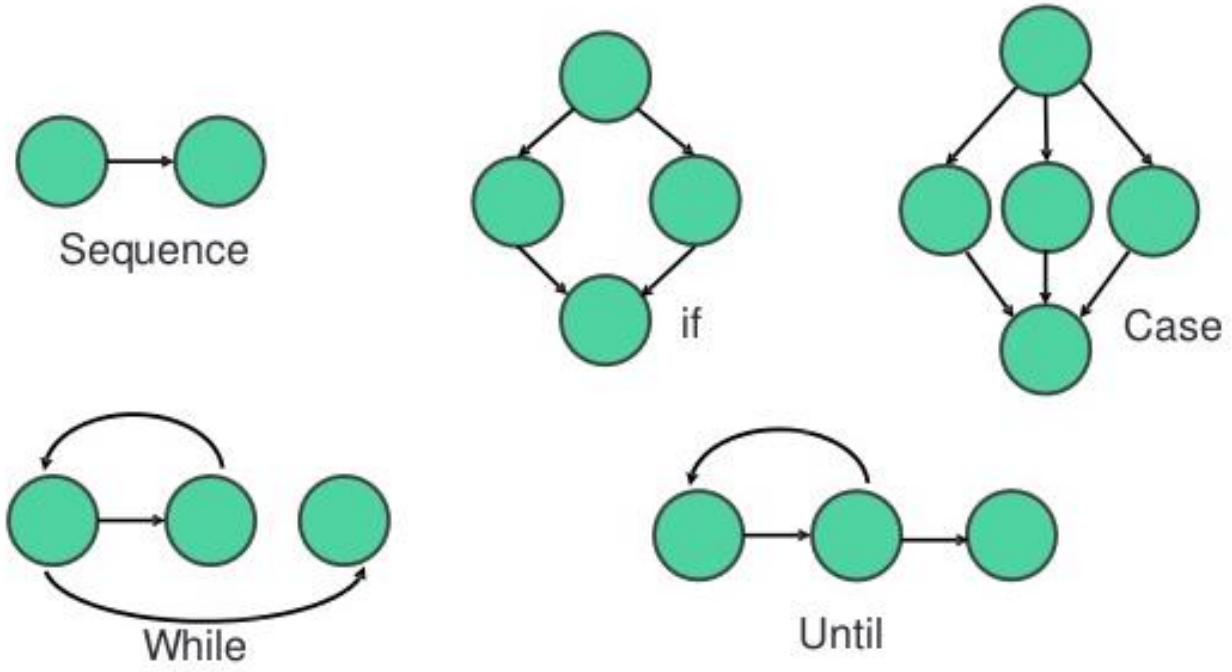
- This is a measure of the control complexity of a program based on the program's control graph.
- It directly measures the number of linearly independent paths through a program's source code.

Control Flow Graph



- Nodes: Statement of the program
  - Linear sequences of statements (block, segment) can be presented as a single node.
    - If the first statement of the sequence is executed, all further statements of the sequence will also be executed.
- Edges: Possible execution sequences of the statements, represented by the nodes.
  - Nodes with several outgoing edges represent the statements that control the flow.
  - IF-statements and loop statements: Two outgoing edges.
  - CASE-statements: several outgoing edges.
- There is exactly one start node at the beginning of a program, or head of the method and exactly one end node at the end of the program or method.

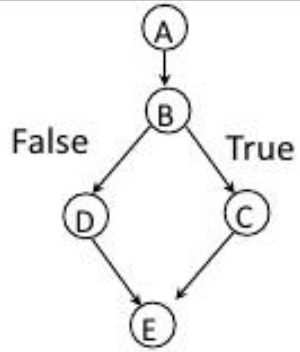
Control Flow Graph



Control Flow Graph

Example

```
/*A*/  float z = InOut.readFloat(),
          w = InOut.readFloat();
          t = InOut.readFloat();
/*B*/  if (z  == 0 || w > 0)
/*C*/    w = w/z;
/*D*/  else w = w + 2/t;
/*E*/  System.out.println(z+' '+w+' '+t);
```

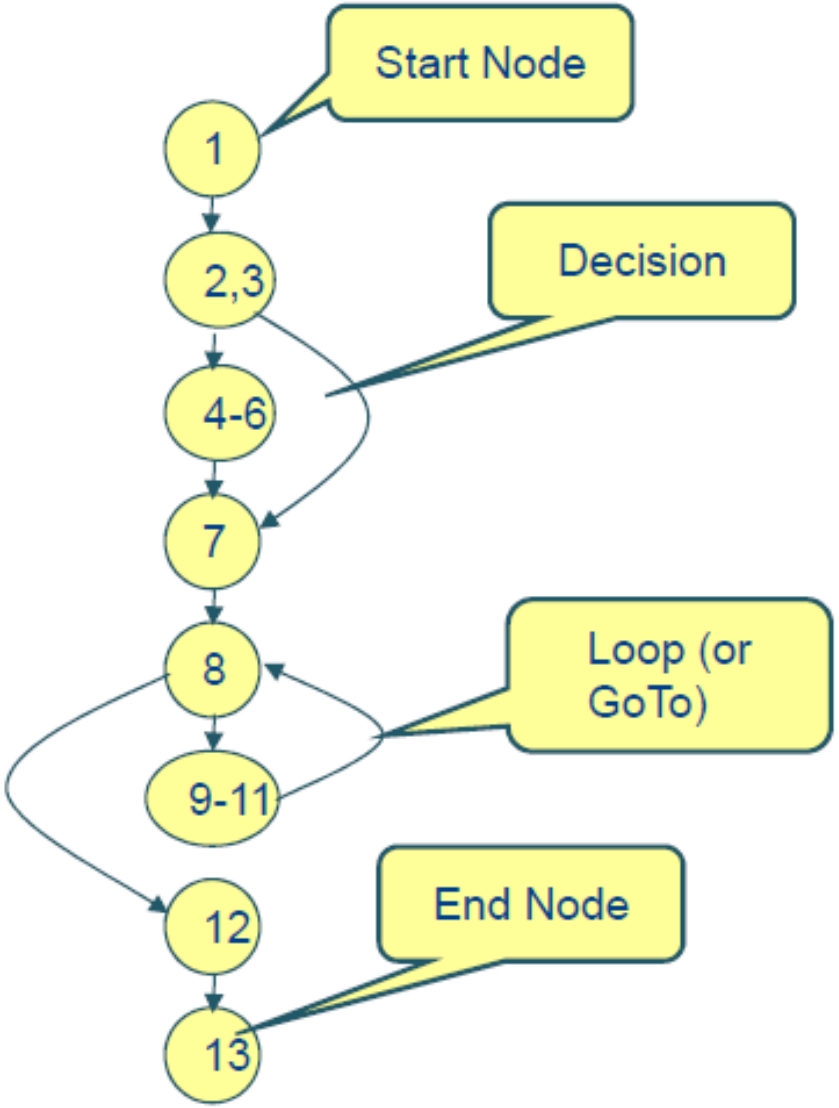


Control flow graph
Without conditions

Control Flow Graph

Activity 1

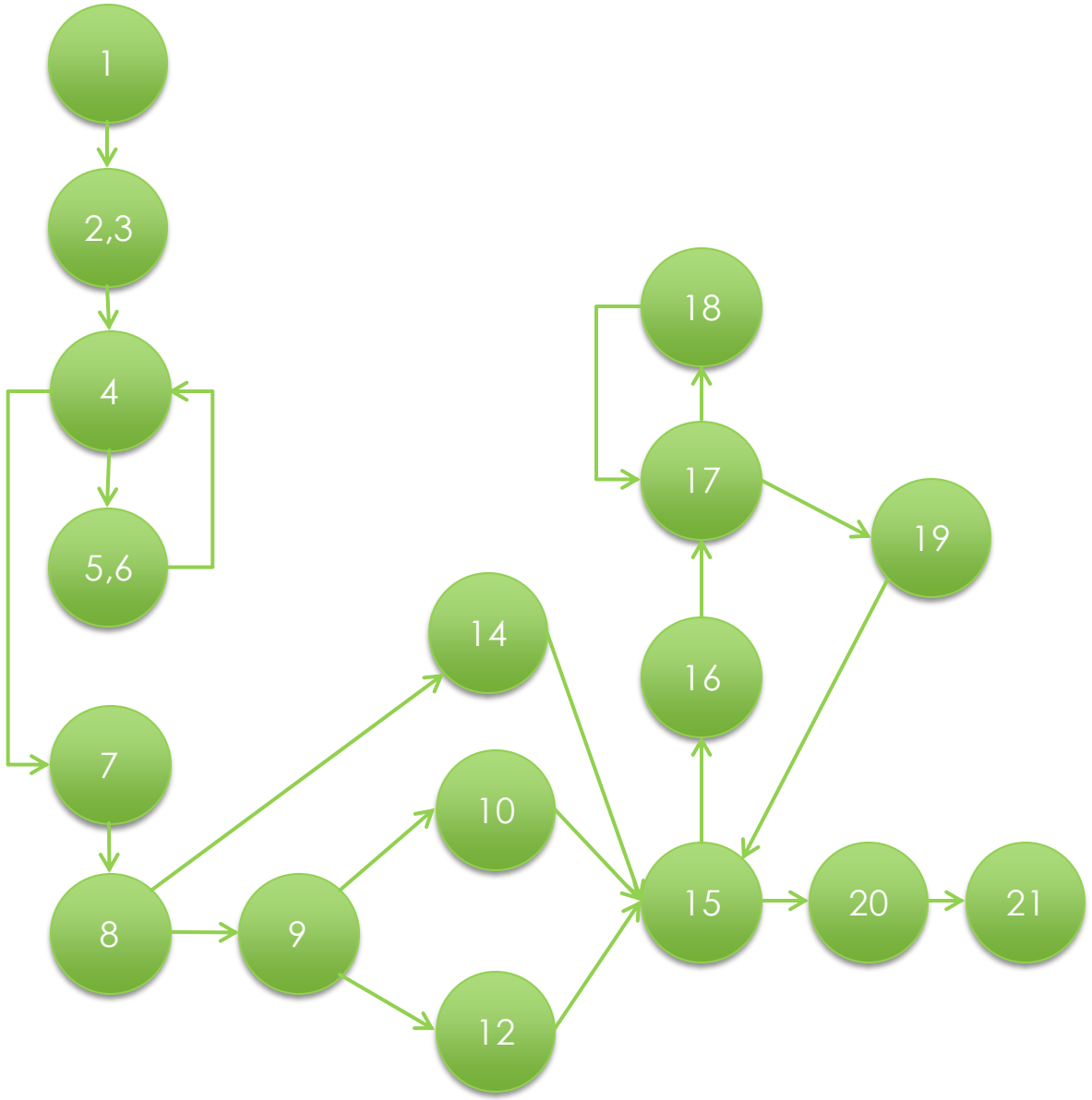
```
1. public int euclid1 (int m, int n) {
2. int r;
3. if (n > m) {
4.   r = m;
5.   m = n;
6.   n = r;
     }
7. r = m % n;
8. while (r != 0) {
9.   m = n;
10.      n = r;
11.      r = m % n;
     }
12. return n;
13. }
```



Control Flow Graph

Activity 2

```
1. public int euclid2 (int m, int n) {
2. int r;
3. double d=0;
4. for (int i=0;i<=10;i++) {
5.     d+=m+n;
6.     d*=i;
       }
7. r = m % n;
8. if ( (d/2)>=m)
9.     if ( (d/2)<n)
10.          r=1;
11.    else
12.          r=0;
13. else
14.    r=-1;
15. while (r != 0) {
16.    m = n;
17.    for (int j=0; j<m;j++)
18.          n += r+j;
19.    r = m % n;
       }
20. return n;
21. }
```



Software Measurement

Software Metrics

Static Metrics

Cyclomatic complexity

- It shows how many independent paths are in a program. More paths means the software is more complex.

  $G = (V,E)$

  $e = |E(G)|$, number of nodes

  $v = |V(G)|$, number of edges

- Cyclomatic Number: $v(G) = v-e+2$
- In test-slang: "McCabe-Metric"

Software Measurement

Software Metrics

Cyclomatic Complexity

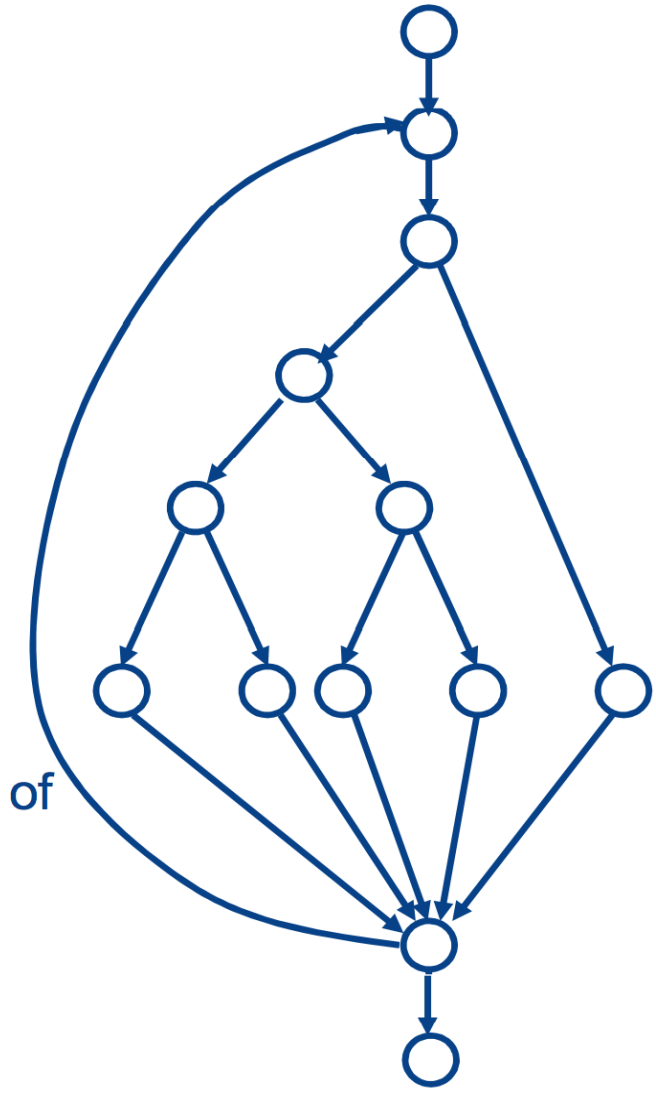
$e = |E(G)| = 13$

$v = |V(G)| = 17$

Cyclomatic Number:

$v(G) = 17\text{-}13\text{+}2 = 6$

- ✧ Cyclomatic number is also the number of decisions nodes in the control flow graph of a structured program +1, here: 5 (+ 1)
- ✧ Example 2: What is the Cyclomatic complexity of the graph shown in slide 17?

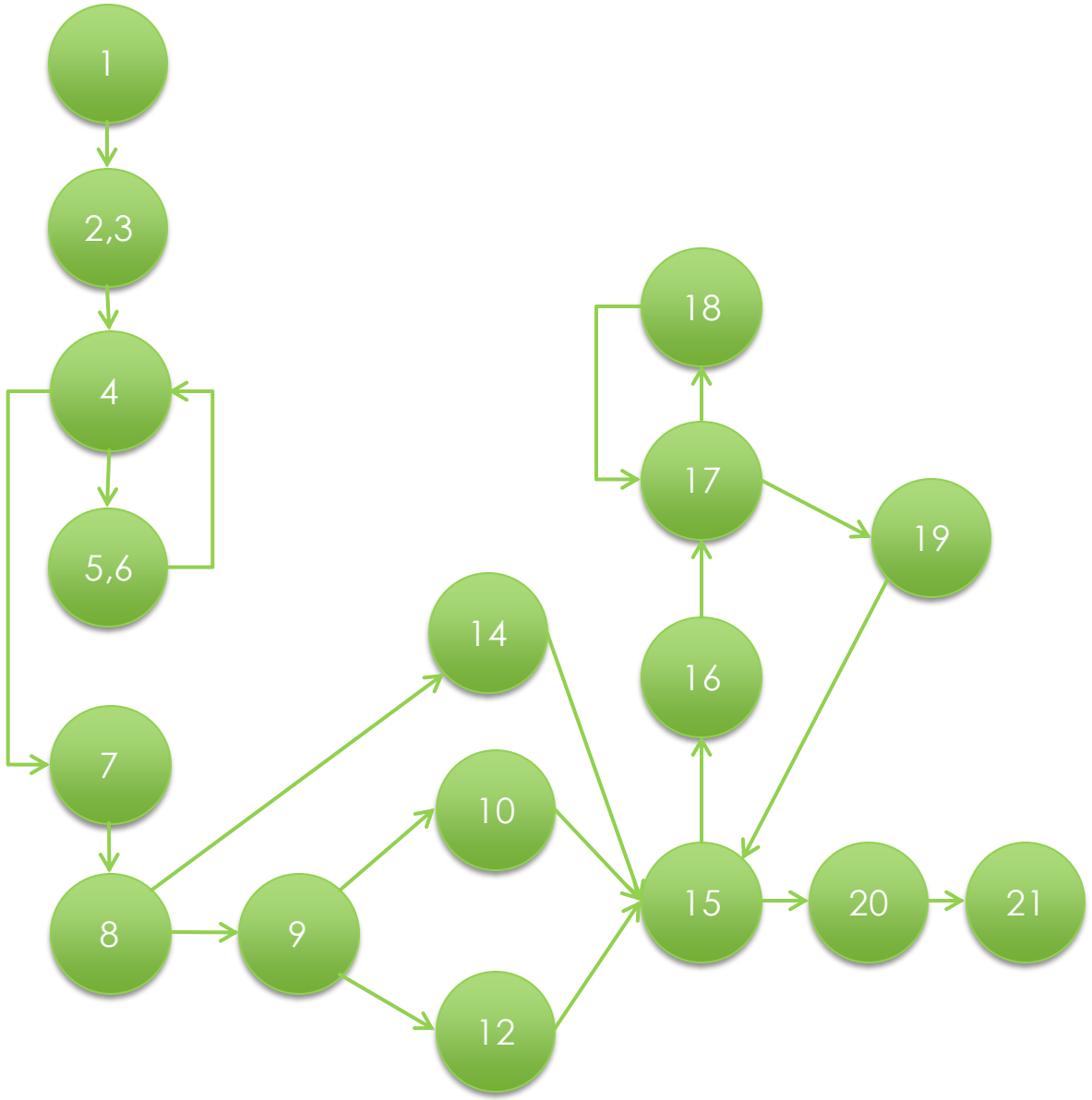


Software Measurement

Software Metrics

Cyclomatic Complexity

What is the Cyclomatic complexity of this graph?



Software Measurement

Software Metrics

Evaluating Cyclomatic Complexity

- A Cyclomatic number of a method higher than 10 is, according to McCabe, not tolerable because it represents excessive complexity.
- The understandability of a program piece is of vital importance for the maintainability.
  - Rework the program part!
  - The measured value of 6 in the example is in the acceptable area, according to McCabe.
  - The higher the determined cyclomatic number is, the more difficult it is to trace the application flow of the program piece and the more difficult it becomes to understand it.

Software Measurement

Software Metrics

Static Metrics

Length of identifiers:

- This is a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program.
- The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.

Software Measurement

Software Metrics

Static Metrics

Depth of conditional nesting:

- This is a measure of the depth of nesting of if-statements in a program.
- Deeply nested if-statements are hard to understand and potentially error-prone.

Software Measurement

Software Metrics

Static Metrics

Fog index:

- This is a measure of the average length of words and sentences in documents.
- The higher the value of a document's Fog index, the more difficult the document is to understand.

Software Measurement

Object Oriented Software Metrics

Weighted Methods per Class

- Weighted Methods per Class (WMC) is an object-oriented metric to measure complexity in a class based on the weighted sum of complexity of each method in the class.
- Consider a Class *C1*, with methods $M_1,...M_n$ that are defined in the class. Let $c_1,...c_n$ be the complexity of the methods (for example Cyclomatic Complexity, or other metrics discussed here). Then :

$$WMC = \sum_{i=1}^{n} c_i$$

Software Measurement

Object Oriented Software Metrics

Weighted Methods per Class

- For example suppose Class A has the following three methods with the given Cyclomatic Complexities (CCs):

  $MethodA_{CC} = 5$

  $MethodB_{CC} = 8$

  $MethodC_{CC} = 3$

- Then WMC_{CC} (i.e. WMC in terms of Cyclomatic Complexity) of Class A is:

$$WMC_{CC}(A) = 5 + 8 + 3 = 16$$

Software Measurement

Object Oriented Software Metrics

Weighted Methods per Class

- The lower limit for WMC is 1 assuming a class has at least one function and the upper default limit is 50 (if Cyclomatic Complexity is the basis).
- If the complexity seems disproportionately high compared to the complexity of other classes, look at ways to split the class, by moving some of the methods into another class.
  - It can also be an indication that the implementation violates one or several of SOLID principles.

Software Measurement

Object Oriented Software Metrics

Weighted Methods per Class

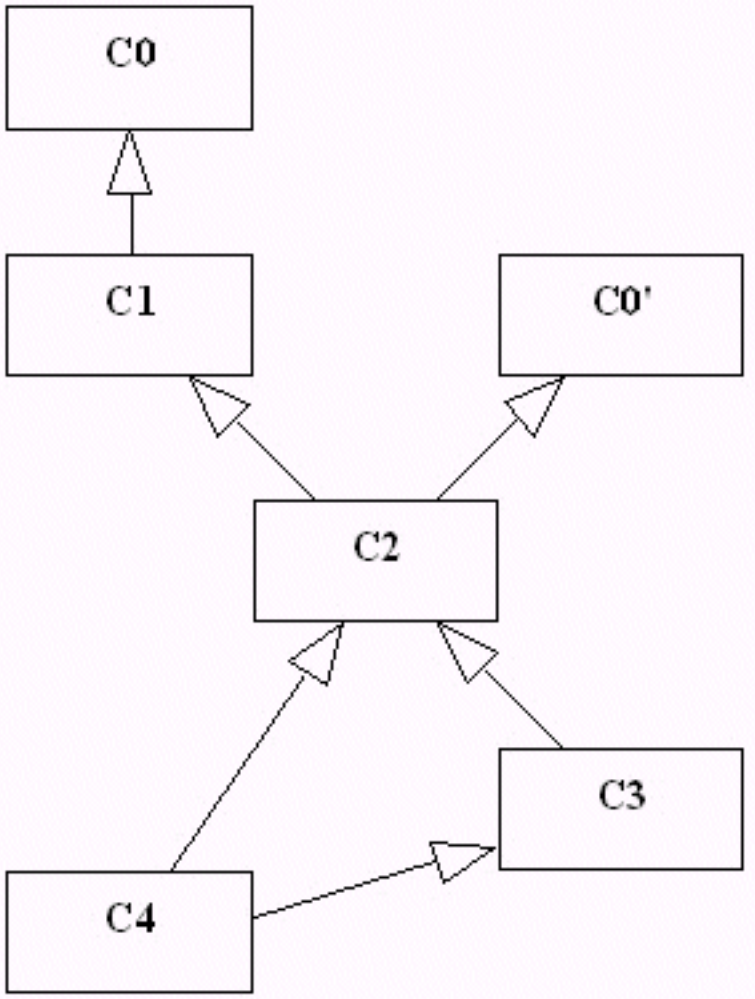
- WMC can be measured for other base metrics as well such as Fan In, Fan Out, etc.
- WMC is an indicator of how much effort is required to develop, verify, and/or maintain a particular class.
- A class with a disproportionately unusual WMC can indicate that the class:
  - is more likely to be prone to errors,
  - has more impact on the children of the class,
  - is more likely to become application-specific and therefore less reusable.

Software Measurement

Object Oriented Software Metrics

Depth in Inheritance Tree

- The Depth in Inheritance Tree (DIT) is the depth of inheritance of the class (i.e. number of ancestors in direct lineage) in the object-oriented paradigm. For example:
  - DIT (C0) = 0
  - DIT (C0') = 0
  - DIT (C1) = 1
  - DIT (C2) = 2
  - DIT (C3) = 3
  - DIT (C4) = 4



Software Measurement

Object Oriented Software Metrics

Depth in Inheritance Tree

- DIT is predicated on three fundamental assumptions:
  1. The deeper a class in the hierarchy, the greater the number of methods it will probably inherit which makes it harder to predict its behaviour.
  2. Deeper trees involve greater design complexity since more classes and methods are involved.
  3. Deeper classes in the tree have a greater potential for reusing inherited methods.
- Assumptions 1 and 2 tell us that having a higher number for depth is bad.
- If that is where it ended we would be in good shape; however assumption 3 indicates that a higher number for depth is good for potential code reuse.

Software Measurement

Object Oriented Software Metrics

Depth in Inheritance Tree

- A low number for depth implies less complexity but also the possibility of less code reuse through inheritance.
- A high number for depth implies more potential for code reuse through inheritance but also higher complexity with a higher probability of errors in the code.

Software Measurement

Object Oriented Software Metrics

Number of Children

- Number of Children (or Number of Direct Subclasses) NOC measures the number of direct subclasses of a class.
- The size of NOC approximately indicates how an application reuses itself.
  - It is assumed that the more children a class has, the more responsibility there is on the maintainer of the class not to break the children's behaviour. As a result, it is harder to modify the class and requires more testing.
- The upper recommended NOC limit for a class is 10 and the lower limit is 0.
  - If NOC exceeds 10 children for a class, this may indicate a misuse of sub-classing.

Software Measurement

Object Oriented Software Metrics

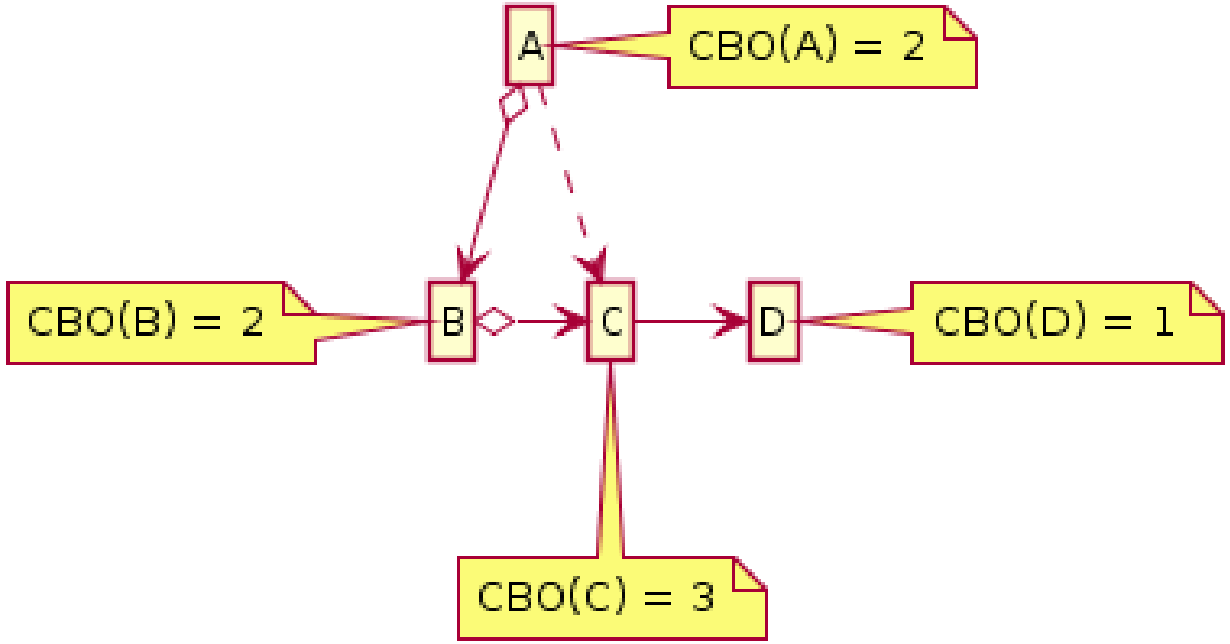
Coupling Between Objects

- Coupling Between Objects (CBO) is a count of the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the attributes of the other.
  - These calls need to be counted in both directions so the CBO of class A is the size of the set of classes that class A references and those classes that reference class A.
  - Since this is a set - each class is counted only once even if the reference operates in both directions i.e. if A references B and B references A, B is only counted once. To put it differently, it is not the number of times A references B.
  - Note that inheritance associations not counted in CBO.

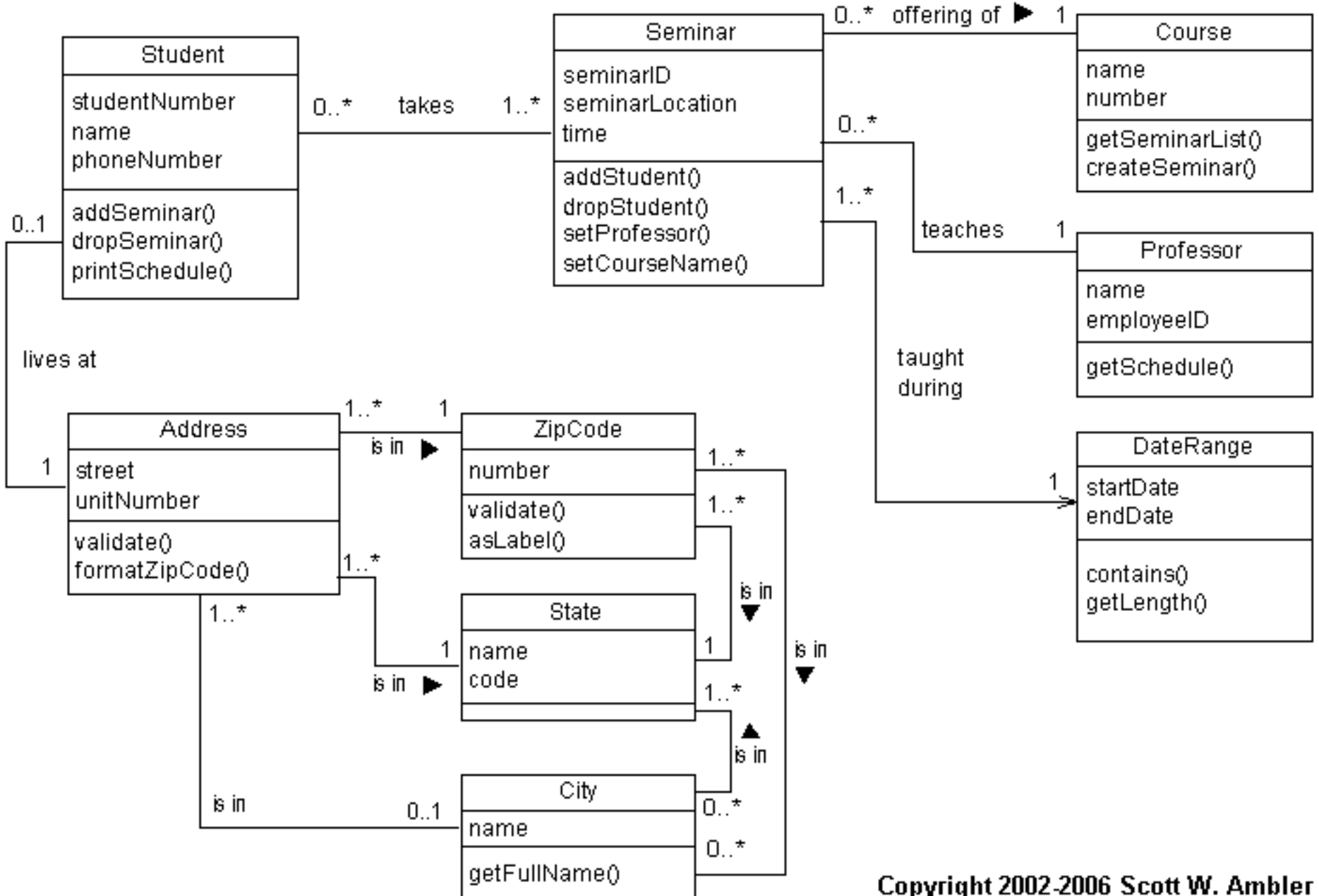
Software Measurement

Object Oriented Software Metrics

Coupling Between Objects



Class Activity: calculate CBOs for each class.

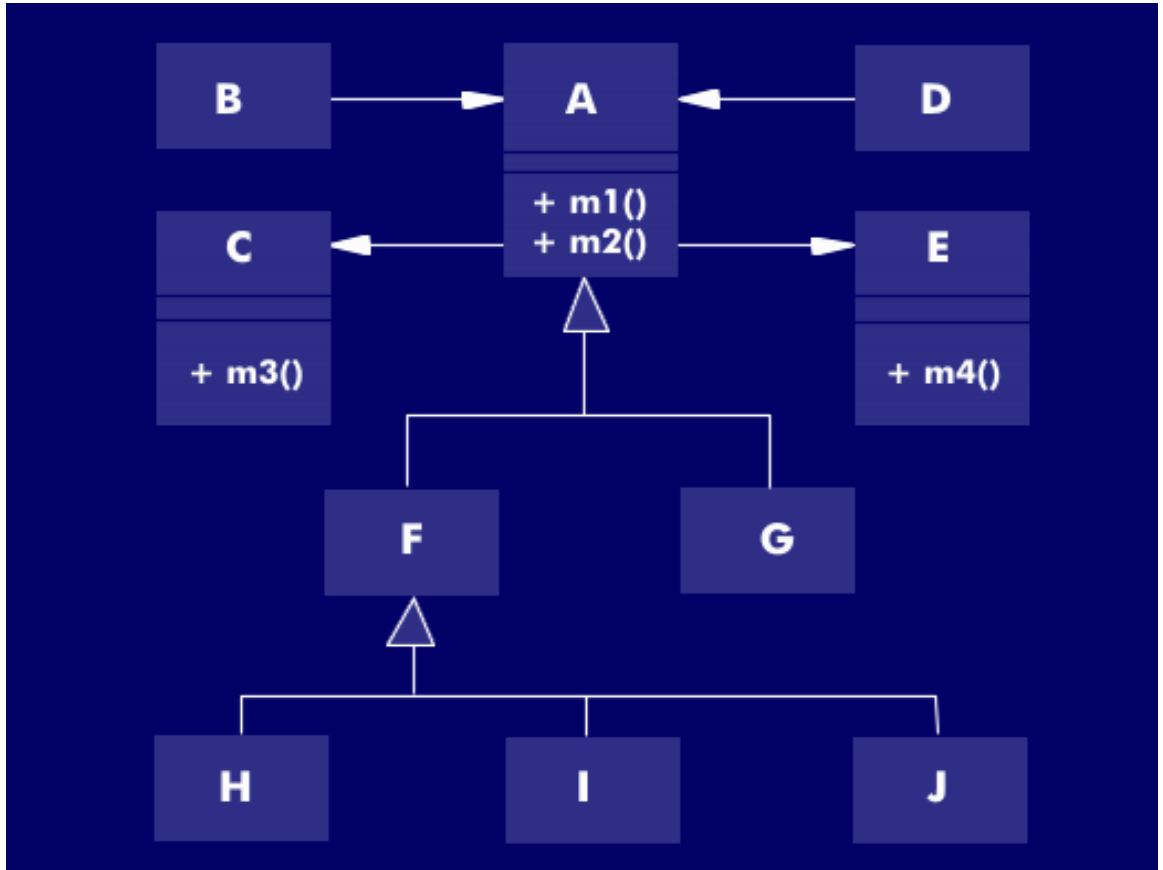


Software Measurement

Object Oriented Software Metrics

Coupling Between Objects

Class Activity: calculate the CBOs for A, B, D, E, C, and F.



Software Measurement

Object Oriented Software Metrics

Coupling Between Objects

- CBO should be as low as possible for two reasons:
  1. Increased coupling increases interclass dependencies, making the code less modular and less suitable for reuse. In other words if you want to package up the code for reuse you might end up having to include code that was not really fundamental to the core functionality.
  2. More coupling means that the code becomes more difficult to maintain since an alteration to code in one area runs a higher risk of affecting code in another (linked) area. The more links between classes the more complex the code and the more difficult it will be to test.
- The only consideration in calculating this metric is whether only the classes written for the project should be considered or whether library classes should be included too.

Software Measurement

Object Oriented Software Metrics

Response for Class

- The Response for Class (RFC) is the "Number of Distinct Methods and Constructors invoked by a Class."
- The response set of a class is the set of all methods and constructors that can be invoked as a result of a message sent to an object of the class. This set includes the methods in the class, inheritance hierarchy, and methods that can be invoked by other objects.

Software Measurement

Object Oriented Software Metrics

Response for Class

- If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.
- The larger the number of methods that can be invoked from a class, the greater the complexity of the class.
- A worst case value for possible responses will assist in appropriate allocation of testing time.
- The RFC for a class should usually not exceed 50 although it is acceptable to have a RFC up to 100.

Software Measurement

Object Oriented Software Metrics

Object-oriented metric	Description
Weighted methods per class (WMC)	This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand. They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.
Coupling between object classes (CBO)	Classes are coupled when methods in one class use methods or instance variables defined in a different class. CBO is a measure of how much coupling exists. A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.

Software Measurement

Software Metrics

Applications

There are two ways in which measurements of a software system may be used:

1. To assign a value to system quality attributes:
    - By measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
    - Note that not all software quality attributes can be directly measured because sometimes they are affected by subjective factors such as user experience and education.

Software Measurement

Software Metrics

Applications

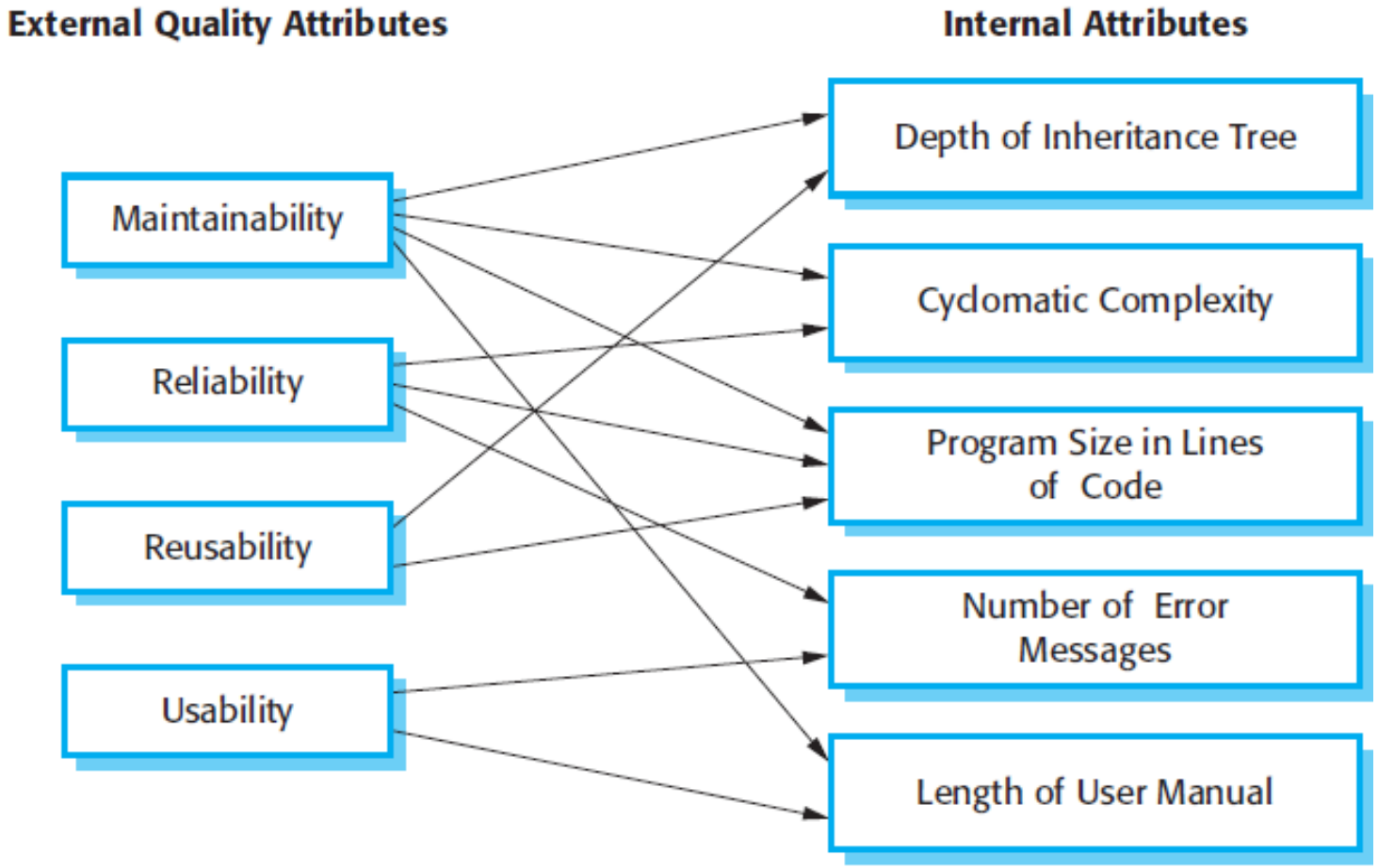
2. To identify the system components whose quality is substandard:

- Measurements can identify individual components with characteristics that deviate from the norm.
- For example, you can measure components to discover those with the highest complexity. These are most likely to contain bugs because the complexity makes them harder to understand.

Software Measurement

Software Metrics

Relationships between internal attributes (metrics) and software quality attributes are shown here.



Software Measurement

Software Metrics

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
ConsoleApplication2 (Release)	68	29	1	21	74
Test	68	29	1	21	74
testURL	63	21	1	19	56
TestURL() : void	51	11		14	21
SendEmail(string) : void	54	3		7	17
Log(string, TextWriter, st	65	3		4	7
main(string, string) : voi	72	1		2	5
main() : void	79	1		2	3
Log(string, TextWriter, st	81	1		2	2
testURL()	100	1		0	1
mainThread	73	8	1	3	18

Code Metrics Results Error List Output