Metrics exercises

Fan-in and Fan-out

- The **Fan-in** of a module is the amount of information that “enters” a module
- The **Fan-out** of a module is the amount of information that “exits” a module
- We assume all the pieces of information with the same size
- Fan-in and Fan-out can be computed for functions, modules, objects, and also non-code components

What does it tell us?

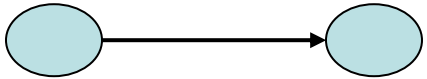
- Low fan-in, low fan-out
  - a module with little dependencies in either direction. All good.
- High fan-in, low fan-out
  - a module that's highly depended upon, but itself doesn't depend on much. Like a low-level utility library.
- Low fan-in, high fan-out
  - a module that depends on lots of other modules, but a few if any modules depend on it. You really can't avoid having one top-level module to tie your whole application together, and naturally this module will depend on each and every other module in the system.
- High fan-in, hight fan-out
  - a very problematic module that can break / need changes whenever one of its many dependencies changes, and it'll in turn break many other parts in the system that rely on it.

McCabe's Complexity Measures

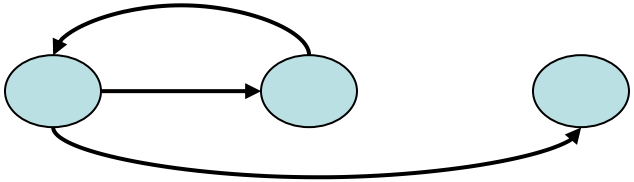
- McCabe's metrics are based on a control flow representation of the program
- A **control flow graph** is used to depict control flow
- Nodes represent processing tasks (one or more code statements)
- Edges represent control flow between nodes

Flow Graph Notation

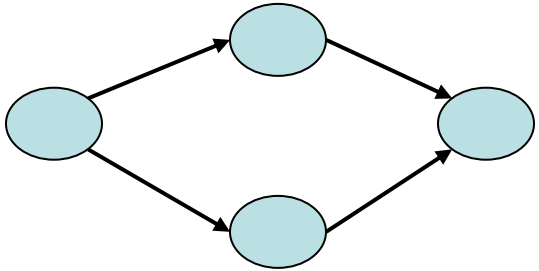
Sequence



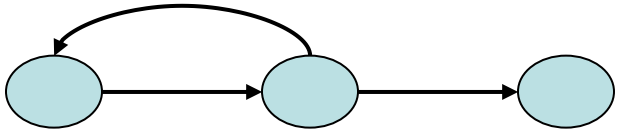
While



If-then-else



Until



Cyclomatic complexity

V(G) = Independent Paths in the Graph

It also can be calculated as the number of regions in the Graph.

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

where:

- *E* = number of edges
- *N* = number of nodes

$$V(G) = P + 1$$

where:

- *P* = number of predicated nodes (if, case, while, for, do, ...)

Example: Code

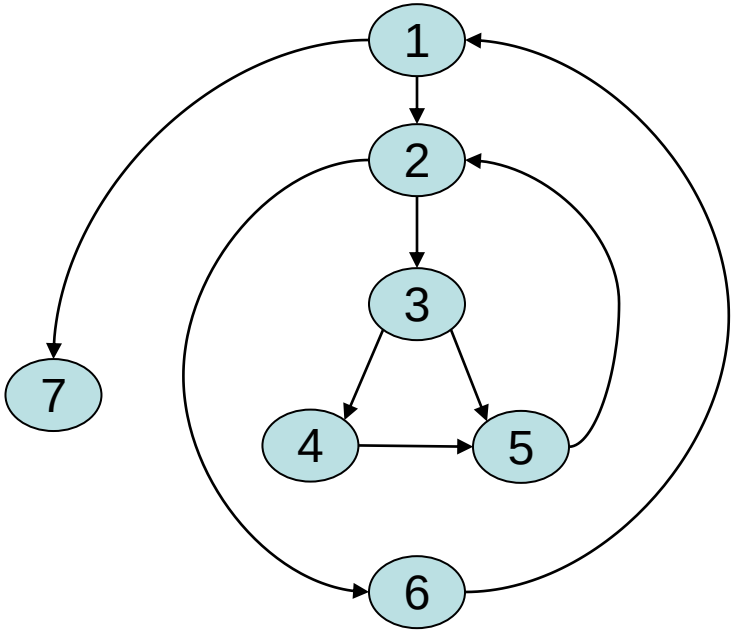
```
i = 0;
while (i < n - 1) do
  j = i + 1;
  while (j < n) do
    if A[i] < A[j]
     then
      swap(A[i], A[j]);
  end do;
```

Find:

- the complexity of the code
- basic paths

Example: Flow Graph

```
i = 0;
while (i < n - 1) do
  j = i + 1;
  while (j < n) do
    if A[i] < A[j]
     then
      swap(A[i], A[j]);
  end do;
```

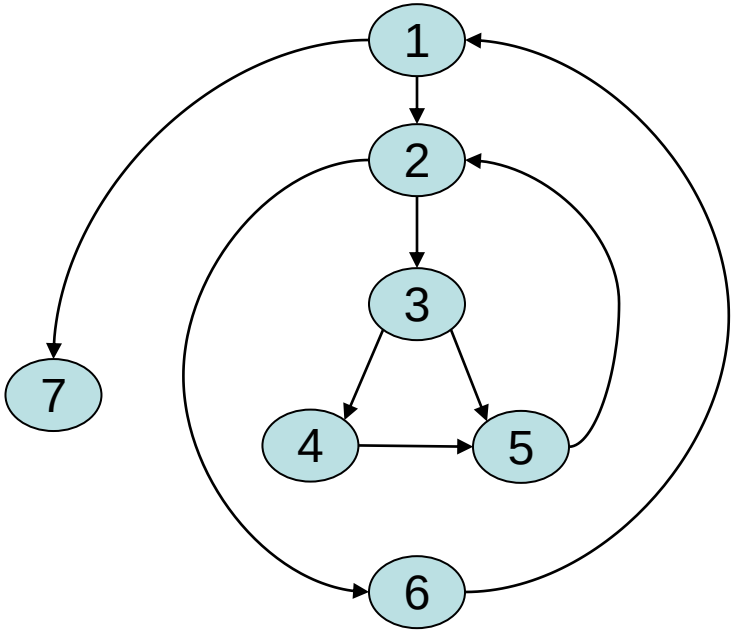


Example: Computing V(G)

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4

Basic paths are:

- 1, 7
- 1, 2, 6, 1, 7
- 1, 2, 3, 4, 5, 2, 6, 1, 7
- 1, 2, 3, 5, 2, 6, 1, 7



Meaning of V(G)

- Complexity increases with the number of decision paths and loops
- *V(G)* is a quantitative measure of the **testing difficulty** and, ultimately, an indication of reliability
- Experimental data shows value of *V(G)* should be no more than 10. Testing is very difficult above this value

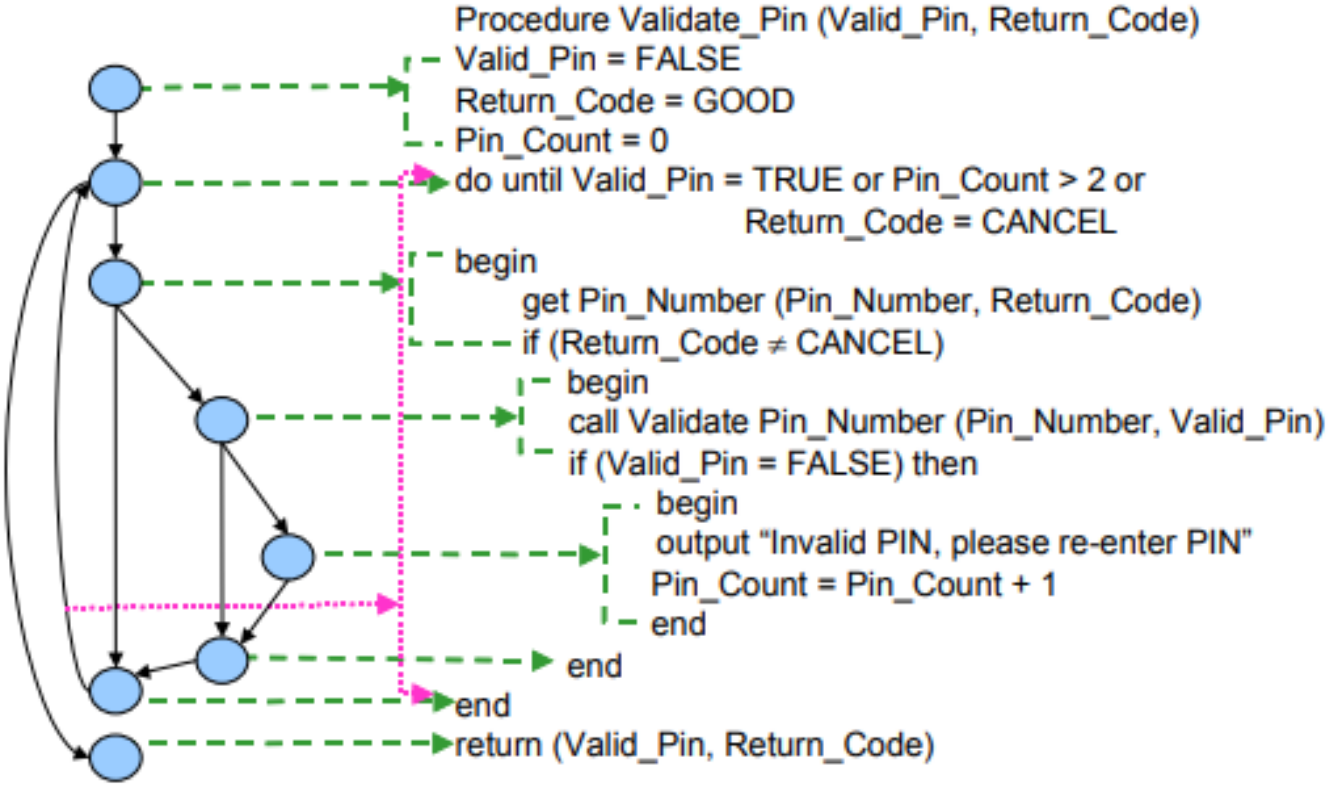
Meaning of V(G)

Complexity	What IT MEANS
1-10	Structured and well written code that is easily testable.
10-20	Fairly complex code that could be a challenge to test. Depending on what you are doing these sorts of values are still acceptable if they're done for a good reason.
20-40	Very complex code that is hard to test. You should look at refactoring this, breaking it down into smaller methods, or using a design pattern.
>40	Crazy code, that is not at all testable and nearly impossible to maintain or extend. Something is really wrong here and needs to be scrutinised further.

Calculate V(G)

```
Procedure Validate_Pin (Valid_Pin, Return_Code)
Valid_Pin = FALSE
Return_Code = GOOD
Pin_Count = 0
do until Valid_Pin = TRUE or Pin_Count > 2 or
                        Return_Code = CANCEL
begin
      get Pin_Number (Pin_Number, Return_Code)
      if (Return_Code ≠ CANCEL)
          begin
          call Validate Pin_Number (Pin_Number, Valid_Pin)
          if (Valid_Pin = FALSE) then
                  begin
                  output "Invalid PIN, please re-enter PIN"
                  Pin_Count = Pin_Count + 1
                  end
          end
end
return (Valid_Pin, Return_Code)
```

Solution


Procedure Validate_Pin (Valid_Pin, Return_Code)
Valid_Pin = FALSE
Return_Code = GOOD
Pin_Count = 0
do until Valid_Pin = TRUE or Pin_Count > 2 or
Return_Code = CANCEL
begin
get Pin_Number (Pin_Number, Return_Code)
if (Return_Code ≠ CANCEL)
begin
call Validate Pin_Number (Pin_Number, Valid_Pin)
if (Valid_Pin = FALSE) then
begin
output "Invalid PIN, please re-enter PIN"
Pin_Count = Pin_Count + 1
end
end
end
return (Valid_Pin, Return_Code)

Exercises (page 3)

Find:

- LOC
- MCC
- Fan-in & Fan-out

Chidamber & Kemerer (CK) Object-Oriented Suite

Includes 6 measures:

- Weighted Method for a class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of children (NOC)
- Coupling between object (CBO)
- Response for a class (RFC)
- Lack of cohesion (LCOM)

CK metrics

- Weighted Method for a class (WMC)
  - a weighted sum of the number of methods of a class
- Depth of Inheritance Tree (DIT)
  - the longest path from the class to the most remote ancestor
- Number of children (NOC)
  - the count of all the direct children of a class
- Coupling between object (CBO)
  - a measure of the dependencies that an object has with other objects
- Response for a class (RFC)
  - the number of methods of a class than can be invoked in response to a call to a method of a class
- Lack of cohesion (LCOM)
  - the absence of cohesion among the methods of a given class

Exercises (page 1-2)

Find:

- WMC, DIT, NOC, CBO, RFC