Path Testing

Mohammad Mousavi

Halmstad University, Sweden

http://bit.ly/TAV16

Testing and Verification,
February 12, 2016

Outline

Functional Testing: Pros and Cons

Pros:

- Straightforward test-case generation
- Based on specification (early test-case generation)

Cons:

- No use of program information
- Gaps and redundancies

Structural Testing

Idea

- Derive structural abstractions from programs
  Example: flow graphs
- Use them to measure the adequacy of the test-set

Structural Testing (Example from the 1st Lecture)

Spec.: input: an integer x $[1..2^{16}]$
output: x incremented by two, if x is less than 50,
x decremented by one, if x is greater than 50, and
50, otherwise.

```
if x < 50 then
   x = x + 2;
end if
if x > 50 then
   x = x - 1;
end if
return x
```

Structural Testing

if $x < 50$ **then**
  $x = x + 2;$
end if
if $x > 50$ **then**
  x = x - 1;
end if
return x

Adequacy criterion: test until all statements are at least executed once (subject of today's lecture: DD-path coverage).

Structural Testing

if x $<$ 50 **then**
  x $=$ x $+$ 2;
end if
if x $>$ 50 **then**
  x $=$ x - 1;
end if
return x

Adequacy criterion: test until all statements are at least executed once (subject of today's lecture: DD-path coverage).

Input	Output	Pass/Fail
3222	3221	P
30	32	P

Outline

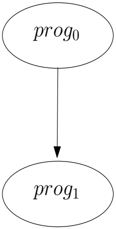
Flow Graphs

- Nodes: program statements
- Edges: $p \rightarrow q$ iff q may execute immediately after p

From Programs to Flow Graphs: Examples

Flow Graph for simple statements

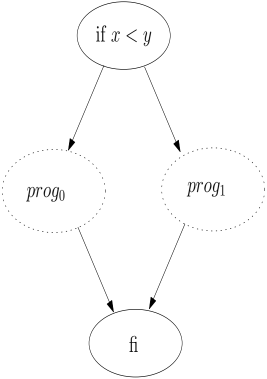
- Sequential composition: $prog_0; prog_1$,



From Programs to Flow Graphs: Examples

Flow Graph for simple statements

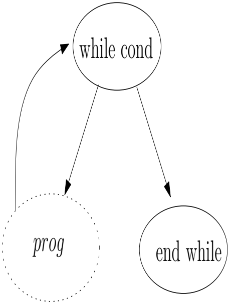
- Sequential composition: $prog_0; prog_1$,
- Conditional: $if (cond) then\ prog_0\ else\ prog_1\ fi$,



From Programs to Flow Graphs: Examples

Flow Graph for simple statements

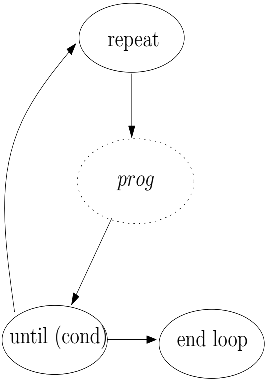
- Sequential composition:
  $prog_0; prog_1$,
- Conditional:
  $if(cond)then\ prog_0\ else\ prog_1\ fi$,
- While loop:
  $while(cond)do\ prog\ endwhile$,



From Programs to Flow Graphs: Examples

Flow Graph for simple statements

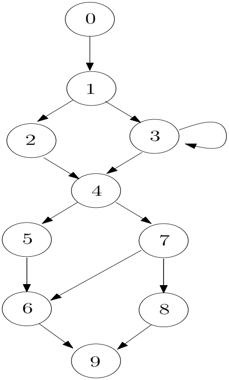
- Sequential composition:
  $prog_0; prog_1$,
- Conditional:
  $if(cond) then\ prog_0\ else\ prog_1\ fi$,
- While loop:
  $while(cond) do\ prog\ endwhile$,
- Repeat-until loop:
  $repeat\ prog\ until(cond)$,



Test Adequacy Criteria

The test-set covers, in the flow graph,

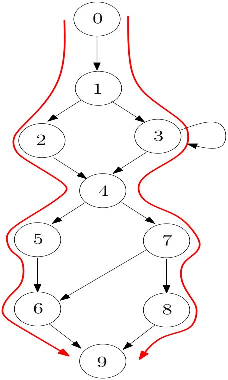
1. all nodes (statement coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

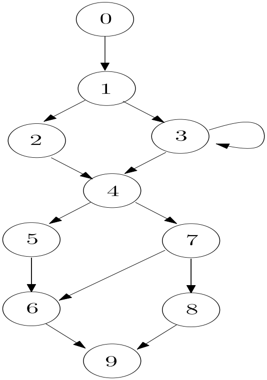
1. all nodes (statement coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

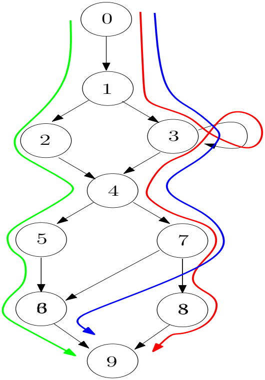
1. all nodes (statement coverage)
2. all edges (DD-path coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

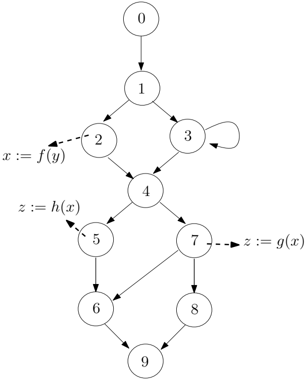
1. all nodes (statement coverage)
2. all edges (DD-path coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

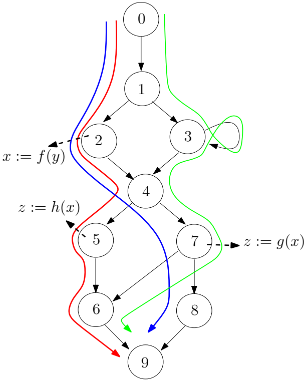
1. all nodes (statement coverage)
2. all edges (DD-path coverage)
3. all prime paths (single-loop coverage)
4. all edges + all combinations of data-flow dependent edges (dependent pairs coverage: next lecture)



Test Adequacy Criteria

The test-set covers, in the flow graph,

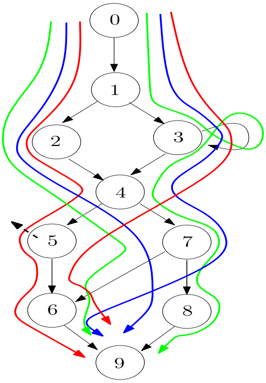
1. all nodes (statement coverage)
2. all edges (DD-path coverage)
3. all prime paths (single-loop coverage)
4. all edges + all combinations of data-flow dependent edges (dependent pairs coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

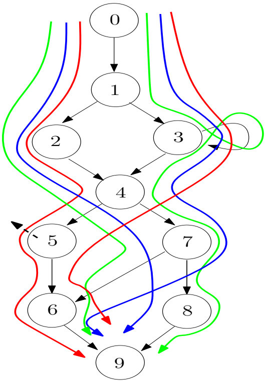
1. all nodes (statement coverage)
2. all edges (DD-path coverage)
3. all prime paths (single-loop coverage)
4. all edges + all combinations of data-flow dependent edges (dependent pairs coverage)
5. all edges + all combinations of condition edges (multiple-condition coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

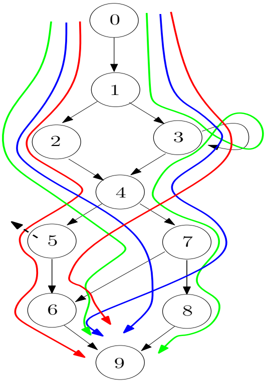
1. all nodes (statement coverage)
2. all edges (DD-path coverage)
3. all prime paths (single-loop coverage)
4. all edges + all combinations of data-flow dependent edges (dependent pairs coverage)
5. all edges + all combinations of condition edges (multiple-condition coverage)



Test Adequacy Criteria

The test-set covers, in the flow graph,

1. all nodes (statement coverage)
2. all edges (DD-path coverage)
3. all prime paths (single-loop coverage)
4. all edges + all combinations of data-flow dependent edges (dependent pairs coverage)
5. all edges + all combinations of condition edges (multiple-condition coverage)
6. all paths (full path coverage)



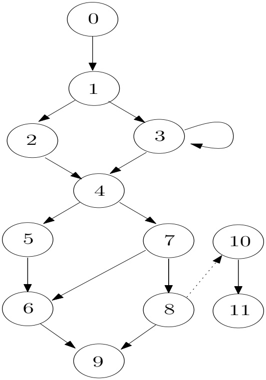
Finite Feasibility

An adequacy criteria should be satisfiable by some finite test-set.

Question: Which of the aforementioned criteria are finitely feasible?

Finite Feasibility

An adequacy criteria should be satisfiable by some finite test-set.



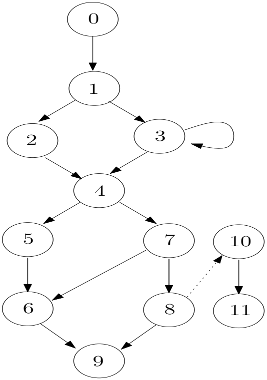
Finite Feasibility

An adequacy criteria should be satisfiable by some finite test-set.

Solution: Adding feasibility:

1. all reachable nodes (feasible statement coverage)
2. all reachable edges (feasible DD-path coverage)
3. all reachable ...

Problem solved?



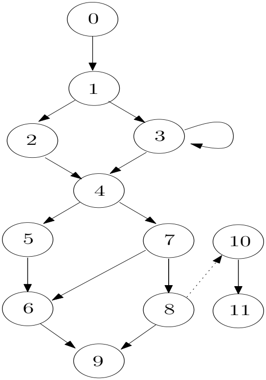
Finite Feasibility

An adequacy criteria should be satisfiable by some finite test-set.

Solution: Adding feasibility:

1. all reachable nodes (feasible statement coverage)
2. all reachable edges (feasible DD-path coverage)
3. all reachable ...

Problem solved? No, checking reachability is undecidable in general!



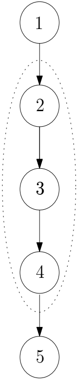
Outline

Chain: Definition

A chain $n_0, \ldots, n_i$, with $0 \leq i$, is a list of nodes s.t.

1. $n_j \rightarrow n_{j+1}$ for each $j < i$,
2. $indeg(n_j) = outdeg(n_j) = 1$, for each $0 \leq j \leq i$,

A chain $n_0, \ldots, n_i$ is maximal when neither $n', n_0, \ldots, n_i$ nor $n_0, \ldots, n_i, n'$ (for any n') are chains. Each node is a member of at most one maximal chain.



DD-Path: Definition

A DD-Path is a set of nodes satisfying one of the following:

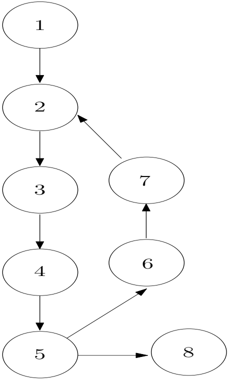
1. $\{n\}$ s.t. $indeg(n) = 0$ (staring node) or $outdeg(n) = 0$ (terminal node),
2. $\{n\}$ s.t. $outdeg(n) \geq 2$ or $indeg(n) \geq 2$ (branch or merge nodes)
3. $\{n_0, \ldots, n_i\}$ with $i \geq 0$ s.t.
   $n_0 \rightarrow \ldots \rightarrow n_i$ is a maximal chain

Property: each node belongs to precisely one DD-path

DD-Path: Simplified Definition

A DD-Path is a set of nodes satisfying one of the following:

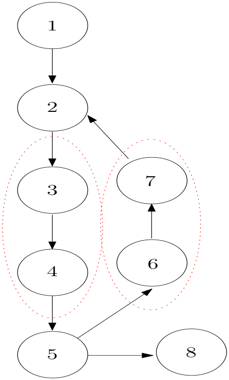
1. $\{n\}$ s.t. $indeg(n) \neq 1$ or $outdeg(n) \neq 1$,
2. $\{n_0, \ldots, n_i\}$ with $i \geq 0$ s.t. $n_0 \rightarrow \ldots \rightarrow n_i$ is a maximal chain



DD-Path: Simplified Definition

A DD-Path is a set of nodes satisfying one of the following:

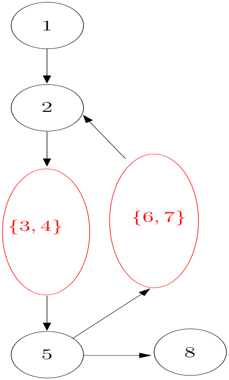
1. $\{n\}$ s.t. $indeg(n) \neq 1$ or $outdeg(n) \neq 1$,
2. $\{n_0, \ldots, n_i\}$ with $i \geq 0$ s.t. $indeg(n_j) = outdeg(n_j) = 1$ and $n_0 \rightarrow \ldots \rightarrow n_i$ is a maximal chain



DD-Path Graph

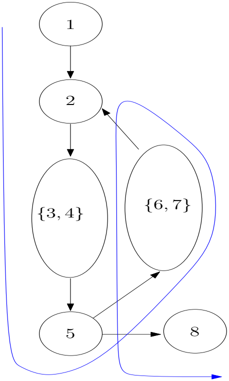
In a DD-Path graph:

1. nodes: DD-Paths as
2. edges: $\{n_i \mid i \in I\} \rightarrow \{m_j \mid j \in J\}$ when $\exists_{i' \in I, j' \in J}$ s.t. $n_{i'} \rightarrow m_{j'}$.



DD-Path Coverage

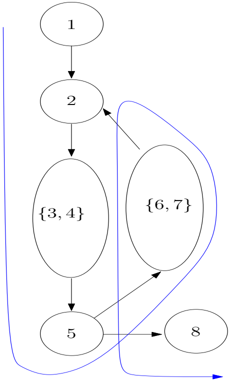
A test-set is adequate when for each node or edge in the DD-path graph, there exists a test-case covering it.



DD-Path Coverage

A test-set is adequate when for each node or edge in the DD-path graph, there exists a test-case covering it.

This is equivalent to edge coverage, but requires less checks.

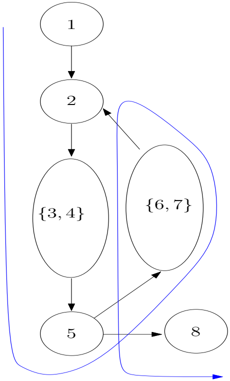


DD-Path Coverage

A test-set is adequate when for each node or edge in the DD-path graph, there exists a test-case covering it.

This is equivalent to edge coverage, but requires less checks.

This subsumes node coverage.



DD-Path Testing: Complete?

```
if x < 50 then
   x = x + 2;
end if
if x > 50 then
   x = x - 1;
end if
return x
```

Input	Output	Pass/Fail
3222	3221	P
30	32	P
49	51	F
50	50	P

DD-Path: Complete?

Solutions:

1. Use stronger adequacy criteria: prime paths, dependent pairs testing, multiple condition coverage testing
2. Problems: more test-sets; even sometimes: not that many more faults detected
3. Use more switch statements instead of sequential conditions.

DD-Path Testing

Pros:

1. DD-paths instead of statements: more efficient coverage measuring
2. DD-paths coverage: a practical measure of test adequacy
3. implemented in many tools

Cons:

1. infeasible paths must be tested!
2. some important paths left untested
3. no test-case generation technique
4. main reason: ignoring specification and data-dependencies: dependent pairs testing (see the next lecture)

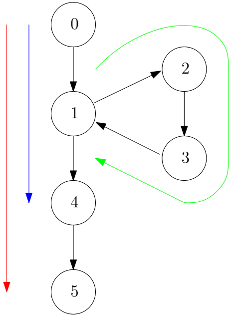
Outline

Simple Path: Definition

A simple path $n_0, \ldots, n_t$, with $0 \leq t$, is a list of nodes s.t.

1. $n_j \rightarrow n_{j+1}$ for each $j < t$,
2. for each $0 \leq i < j \leq i$, $n_i \neq n_j$ or $(n_i = n_0$ and $n_j = n_t)$

Informally: a simple path visits a node at most once, except that the start and the ending node may be the same.

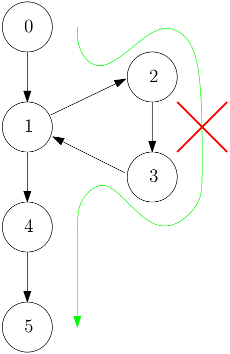


Simple Path: Definition

A simple path $n_0, \ldots, n_i$, with $0 \leq i$, is a list of nodes s.t.

1. $n_j \rightarrow n_{j+1}$ for each $j < i$,
2. for each $0 < i < j \leq i$, $n_i \neq n_j$

Informally: a simple path visits a node at most once, except that the start and the ending node may be the same.

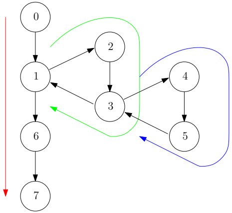


Prime Path: Definition

A prime path is:

- a simple path that
- does not appear as a proper sub-path of any other simple path.

Informally: a prime path is a complete path from start to end, or a complete and simple iteration of a loop (infeasibility issue set aside)

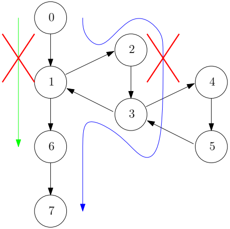


Prime Path: Definition

A prime path is:

- a simple path that
- does not appear as a proper sub-path of any other simple path.

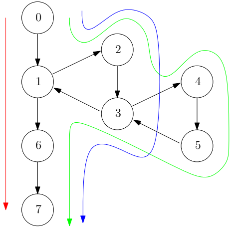
Informally: a prime path is a complete path from start to end, or a complete and simple iteration of a loop



Prime Path Coverage

A test set is adequate if for each prime path, there is a test case covering it (as a sub-path).

Informally: all complete simple paths and up to one iteration of each loop

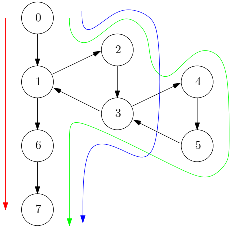


Prime Path Coverage

A test set is adequate if for each prime path, there is a test case covering it (as a sub-path).

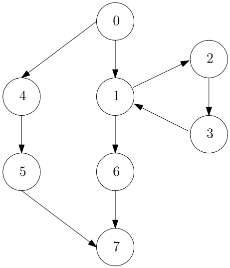
Informally: all complete simple paths and up to one iteration of each loop

Variants with tours, detours and side-trips



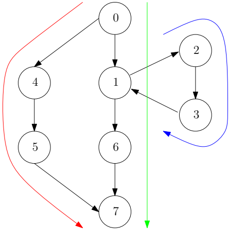
Prime Path Coverage: Exercise

Propose a set of test cases that is adequate for prime path coverage.



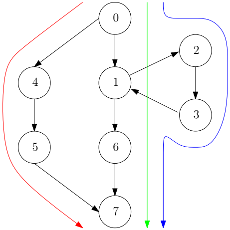
Prime Path Coverage: Solution

Prime paths



Prime Path Coverage: Solution

Prime paths



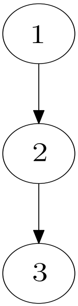
Outline

Testability: Cyclomatic number

1. Idea (very informal):
   Take one path from start to exit,
   count the number of alternatives by
   flipping one condition at a time.
2. Also called: nullity, first Betti number, dimension of cycle space

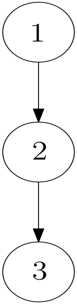
Cyclomatic number: Examples

Cyclomatic number:



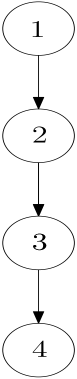
Cyclomatic number: Examples

Cyclomatic number: 1



Cyclomatic number: Examples

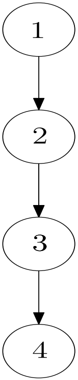
Cyclomatic number:



Cyclomatic number: Examples

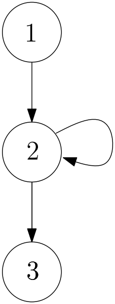
Cyclomatic number: 1

Observation: Cyclomatic nr. is size independent...



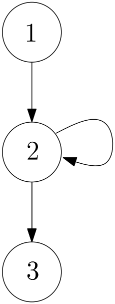
Cyclomatic number: Examples

Cyclomatic number:



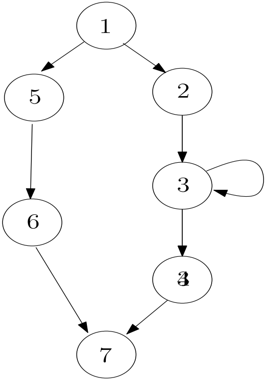
Cyclomatic number: Examples

Cyclomatic number: 2



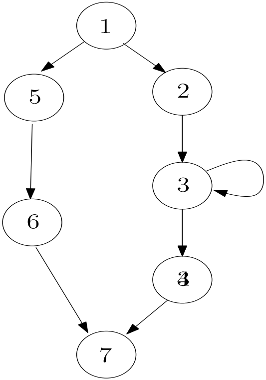
Cyclomatic number: Examples

Cyclomatic number:



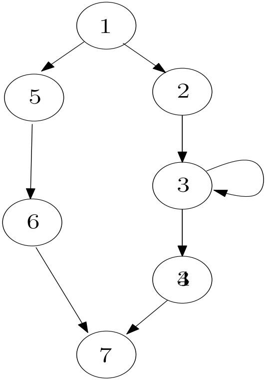
Cyclomatic number: Examples

Cyclomatic number: 3



Cyclomatic number: Examples

Cyclomatic number:



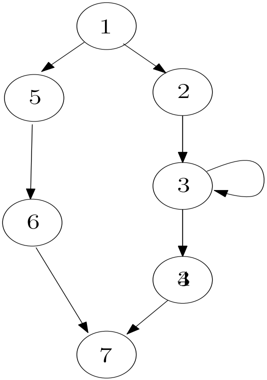
Cyclomatic number: Examples

Cyclomatic number: 3

Only for programs with:

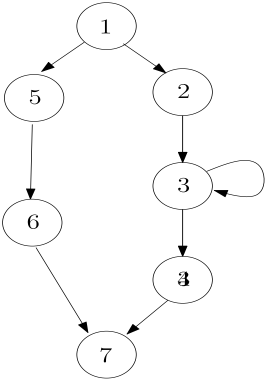
1. one connected component,
2. one starting state, and
3. one terminal state:

$V(G) = \#edges - \#nodes + 2$



Cyclomatic number: Calculation

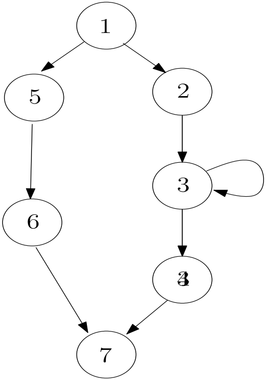
Cyclomatic number:



Cyclomatic number: Calculation

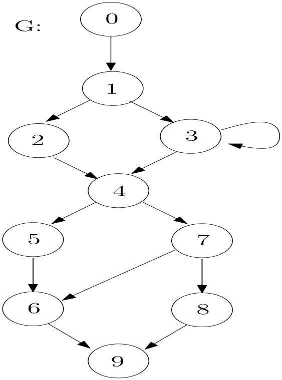
Cyclomatic number: 3

For (connected) planar graphs:
$V(G) = \#regions\ in\ the\ plane$



Cyclomatic number: Example

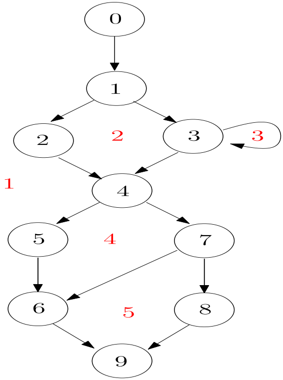
1. the cyclomatic number of G?
2. the cyclomatic number of G sequentially composed with itself?
3. the cyclomatic number of G *n*-times sequentially composed with itself?



Cyclomatic number: Example

What is

1. the cyclomatic number of G? 5
2. the cyclomatic number of G sequentially composed with itself? $9 = 5 + 5 - 1$
3. the cyclomatic number of G *n*-times sequentially composed with itself? $4 * n + 1$



Cyclomatic number: Implications

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
>50	untestable program (very high risk)

Conclusions

1. Cyclomatic number: a measure for software complexity and testability
   watch out for programs with $V(G) > 10$!
2. implemented in several tools (see the course web-site for examples)
3. a measure of test-set adequacy (originally invented for this purpose!)

Conclusions

1. Cyclomatic number: a measure for software complexity and testability
   watch out for programs with $V(G) > 10$!
2. implemented in several tools (see the course web-site for examples)
3. a measure of test-set adequacy (originally invented for this purpose!)
   Let's have a better look...