# Verification and Validation เน้น Testing

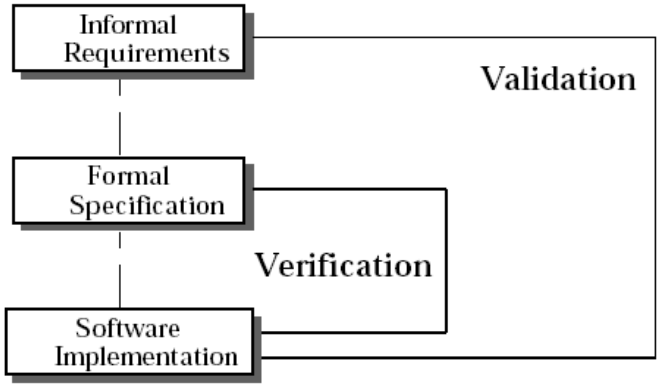
อ.ทรงศักดิ์ รองวิริยะพานิช ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ศูนย์รังสิต

rongviri@yahoo.com,

rongviri@cs.tu.ac.th

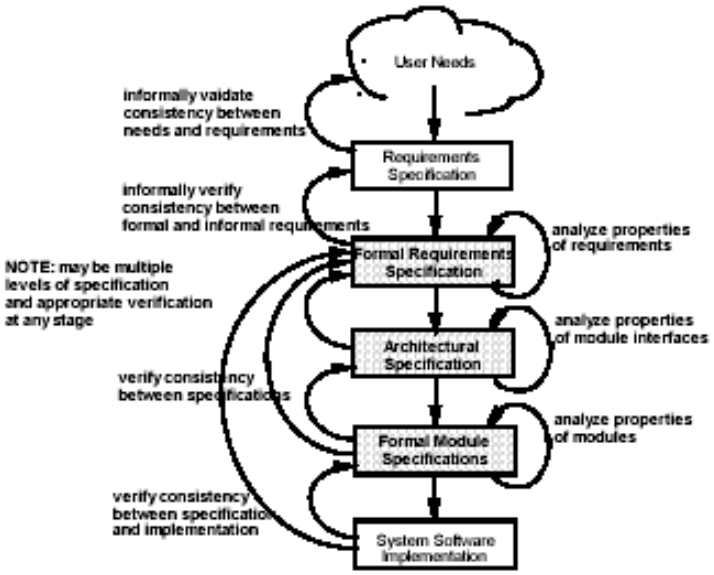


## Verification and Validation

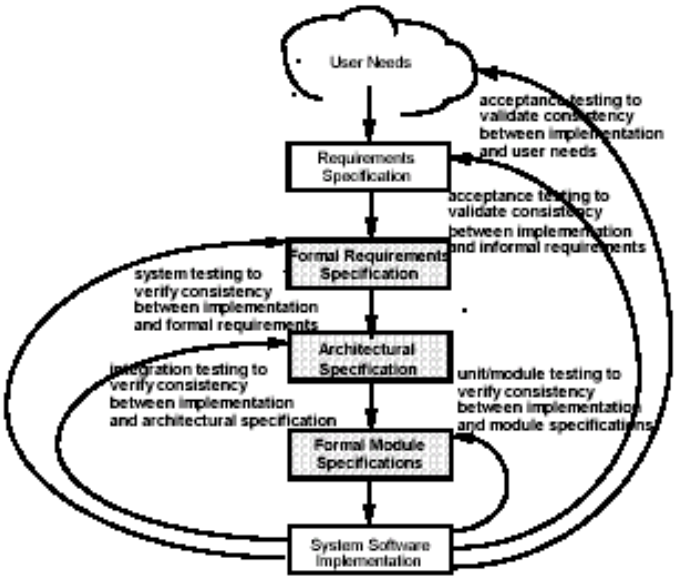



*Verification: is implementation consistent with requirements specification?*
*Validation: does the system meet the customer's/user's needs?*

# Verification with Formal Specifications



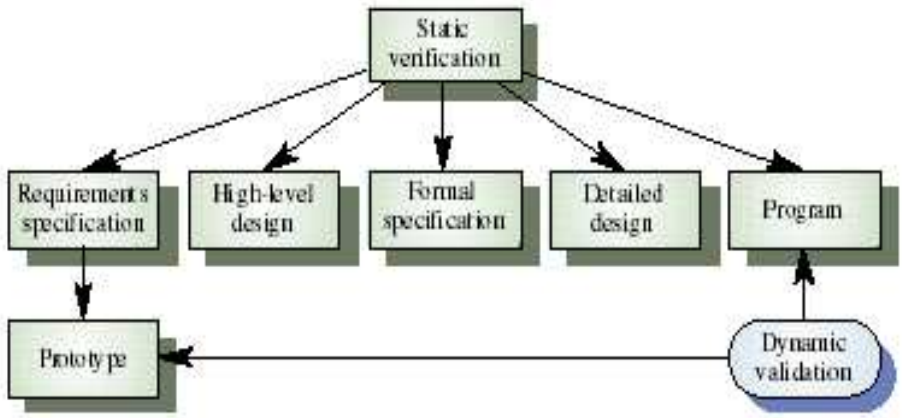

# Testing with Formal Specifications



## Dynamic and static verification

- *Dynamic V & V* Concerned with exercising and observing product behaviour (testing)
- *Static verification* Concerned with analysis of the static system representation to discover problems

## Static and dynamic V&V



## Software Testing

- Exercising a system or component
  - on some predetermined input data
  - capturing the behavior and output data
  - comparing with test oracle
  - For the purpose of
    - identifying inconsistencies
    - providing confidence in consistency with specification and/or measurable qualities
    - demonstrating satisfaction of user needs and/or subjective qualities

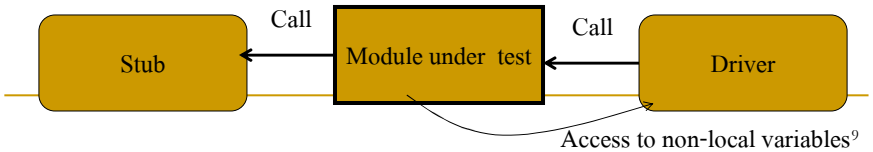
*Where are the limitations on effectiveness introduced?*

## Failures, Errors, Faults

- Failure: incorrect/unexpected behavior or output
  - incident is the symptoms revealed by execution
  - failures are usually classified
- Potential Failure: incorrect internal state
  - sometimes also called an error, state error, or internal error
- Fault: anomaly in source code
  - may or may not produce a failure
  - a “bug”
- Error: inappropriate development action
  - action that introduced a fault

## คุณลักษณะการทำTest

- Test should be organized to help locate errors, not just detect error.
  การ locate error ได้ทำให้debug ได้ง่าย
- Testing should be repeatable เหมือนการทดลองทางวิทยาศาสตร์ที่ให้ผลลัพธ์เดิมเสมอถ้าทำtest ในcondition เดิม input ค่าเดิมและ environment ของการ execute เดิม
- การทำ test เช่นเดียวกันต้องมีตัวที่ควบคุม environment/condition ของการ execute test ให้มีสภาพเหมือนกันทุกครั้งก่อนการทำ test จึงจะให้ผลลัพธ์เหมือนเดิม เรียกตัวควบคุมว่า **Test driver** ขณะที่ **Test Stub** แทน Module อื่นๆที่จำเป็นของระบบที่ทดสอบแต่ไม่ใช่ Module ส่วนที่ทดสอบ




## Fundamental Testing Questions

- Test Case:
  How is test described / recorded?
- Test Criteria:
  What should we test?
- Test Oracle:
  Is the test correct?
- Test Adequacy:
  How much is enough?
- Test Process:
  Is testing complete and effective?

*How to make the most of limited resources?*

## Test Case

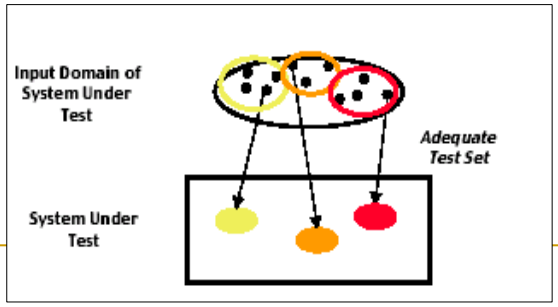
- Specification of
  - identifier
  - test items
  - input and output specs
  - environmental requirements
  - procedural requirements
- Augmented with history of
  - actual results
  - evaluation status
  - contribution to coverage

## Test Criterion

- Test Criterion provides the guidelines, rules, strategy by which test cases are selected
  - requirements on test data ->
    - conditions on test data ->
    - actual test data
- Equivalence partitioning is hoped for
  - a test of any value in a given class is equivalent to a test of any other value in that class
  - if a test case in a class reveals a failure, then any other test case in that class should reveal the failure
  - some approaches limit conclusions to some chosen class of faults and/or failures

# Subdomain-Based Test Adequacy Criteria

- A test adequacy criterion C is subdomain-based if it induces one or more subsets, or subdomains, of the input domain D
- A subdomain-based criterion C typically does not partition D (into a set of non-overlapping subdomains whose union is D)






# Theoretical Foundation of Testing

- Let P Program to test, D domain of all possible input for P, R range of all possible result from P
- พิจารณาให้ P เป็น (partial)function จาก D ไป R
- P(d) แทน output ของโปรแกรมสำหรับ input d
- OR เป็น output requirement ของโปรแกรม P
- โปรแกรม P correct สำหรับ requirement OR ก็ต่อเมื่อ **for a given d in D, P(d) satisfies OR**
- **error หรือ defect** ถ้ามี P(d) ที่ไม่satisfied OR
- **test case** คือ d = element ใน D
- **test set** คือ a finite set of test cases D1 $\subseteq$ D
- ideal test set คือ ถ้า P incorrect จะต้องมี d ใน test set ที่P(d) ไม่ satisfies OR

# Theoretical Foundation of Testing

- Test selection criterion C : เป็น condition ที่ Test set ต้องมี เช่น
  - C{<X1,X2, …,Xn> | n >=3 and exists i, j, k ($X_i$ < 0, $X_j$ =0, $X_k$>0)}
- Test selection criterion C is consistent
  - Test sets T1, T2 ที่อยู่ใน C  T1 เป็น test set ของP แล้ว T2 เป็นด้วย
- Test selection criterion is complete
  - ถ้า P error แล้วมี Test set ที่ detect error ได้ที่อยู่ใน C



## Complete Coverage principle

- Testing criterion มีหน้าที่group elements of input ให้เป็นกลุ่มโดยให้สมาชิกในกลุ่มให้outputเหมือนๆกัน
- เราสามารถเลือกตัวแทน(representative)จากแต่ละกลุ่ม Di
- ถ้าเรามี testing selection criterion ที่ทำให้ $UD_i$ = D เราเรียกว่า testing criterion satisfies the “complete coverage principle”
- Complete Coverage Principle เน้นที่การแบ่งกลุ่ม input ให้ โดยเมื่อรวมกลุ่มแล้วต้องได้เท่ากับ set ของ input ทั้งหมดที่โปรแกรมยอมรับ
- มี Coverage ประเภทอื่นอีกที่ไม่ใช่การ coverage input เช่น statement coverage หรือ edge coverage หรือ path coverage

## ตัวอย่างการ apply testing criterion ที่satisfies Complete Coverage principle

If the input value n is < 0, then an appropriate error message must be printed. If 0<= n < 20, then the exact value of n! must be printed. If 20 <= n <= 200, then an approximate value of n! must be printed in floating point format, e.g. using some approximate method of numerical calculus. The admissible error is 0.1% of the exact value. Finally, if n>200, the input can be rejected by printing an appropriate error message.

**เราแบ่งเป็น classes ได้ {n<0},{0<=n<20},{20<=n<=200},{n>200}**
**test set เช่น {-10,5,175, 250} ถือว่าprogram ถูกต้องสำหรับinputตัวอื่นในทั้ง4classes**

17

## Test Oracle

- A test oracle is a mechanism for verifying the behavior of test execution
  - extremely costly and error prone to verify
  - oracle design is a critical part of test planning
- Sources of oracles
  - input/outcome oracle
  - tester decision
  - regression test suites
  - standardized test suites and oracles
  - gold or existing program
  - formal specification

## Test Adequacy

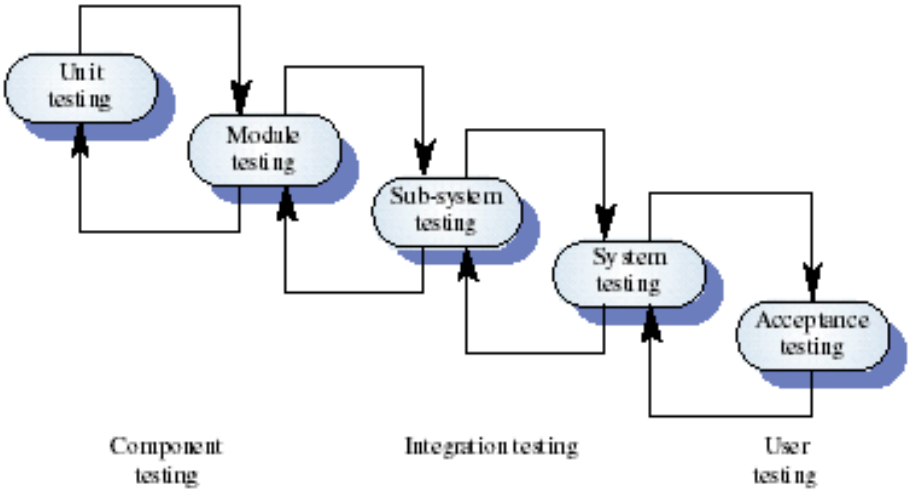
- Theoretical notions of test adequacy are usually defined in terms of adequacy criteria
  - Coverage metrics (sufficient percentage of the program structure has been exercised)
  - Empirical assurance (failures/test curve flatten out)
  - Error seeding (percentage of seeded faults found is proportional to the percentage of real faults found)
  - Independent testing (faults found in common are representative of total population of faults)
- Adequacy criteria are evaluated with respect to a test suite and a program under test

## Testing throughout the Process

- Unit testing: testing of code unit (subprogram, module, subsystem)
  - Usually requires use of test drivers
- Integration testing: testing of interfaces between integrated units
  - Incremental or "big bang"
- System testing: testing complete system for satisfaction of requirements specification
- Acceptance testing: comparing to user requirements
- Regression testing: testing after change

*What are Component Testing? Architecture Testing?*

## The testing process






## Testing is a creative process

- Test case execution is only a (relatively small) part of the process
- Test case generation is difficult, but extremely important
- Planning is essential
  - To achieve early and continuous *visibility*
  - To choose appropriate techniques at each stage
  - To build a testable product
  - To coordinate complementary analysis and testing
- Measuring answers many managerial and technical problems

# Testing activities before coding

- Planning
  - acceptance test planning (requirements elicitation)
  - system test planning (requirements specifications)
  - Integration & unit test planning (architectural design)
- Generation
  - create functional system tests (requirement specifications)
  - generate test oracles (detailed design)
  - generate black box unit tests (detailed design)

# Testing activities after coding

- Generation
  - create scaffoldings (unit coding)
- Execution
  - unit test execution (unit coding)
  - integration test execution (integration and delivery)
  - system test execution (integration and delivery)
  - acceptance test execution (integration and delivery)
  - regression test execution (maintenance)
- Measuring
  - coverage analysis (unit coding)
- Generation
  - deliver regression test suites (integration and delivery)
  - revise regression tests (maintenance)

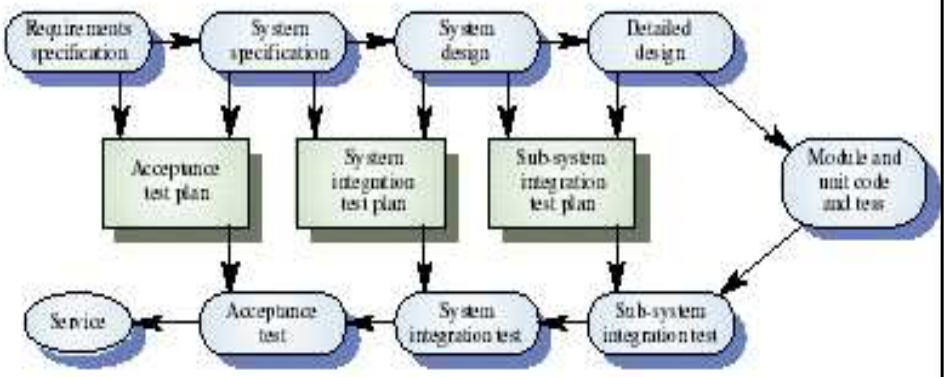
## Test planning and scheduling

- Describe major phases of the testing process
- Describe tracability of tests to requirements
- Estimate overall schedule and resource allocation
- Describe relationship with other project plans
- Describe recording method for test results

## The test plan

- The testing process
- Requirements traceability
- Tested items
- Testing schedule
- Test recording procedures
- Hardware and software requirements
- Constraints

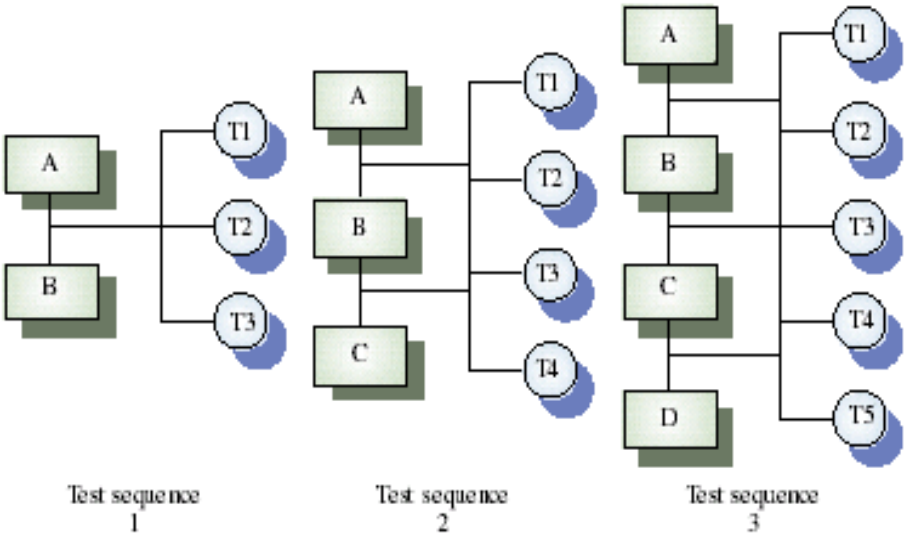
# The V-model of development



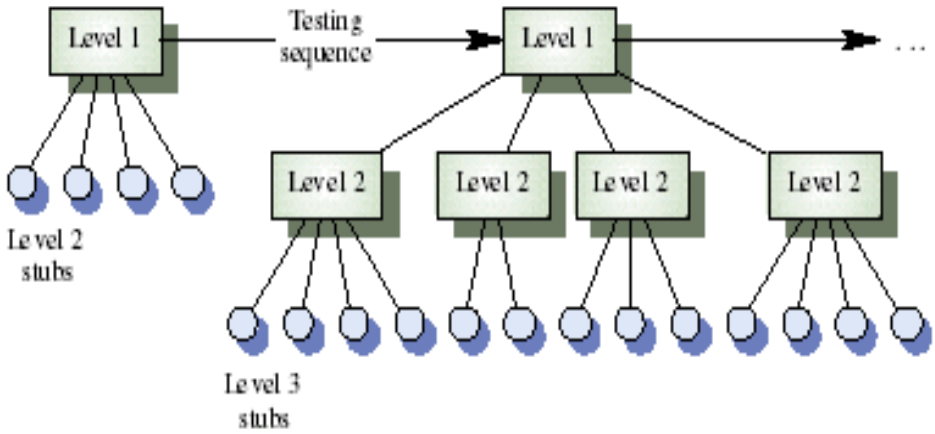

# Testing strategies

- Tetsing strategies are ways of approaching the testing process
- Different strategies may be applied at different stages of the testing process
- Strategies covered
  - Top-down testing
  - Bottom-up testing
  - Thread testing
  - Stress testing
  - Back-to-back testing

# Incremental testing




# Top-down testing

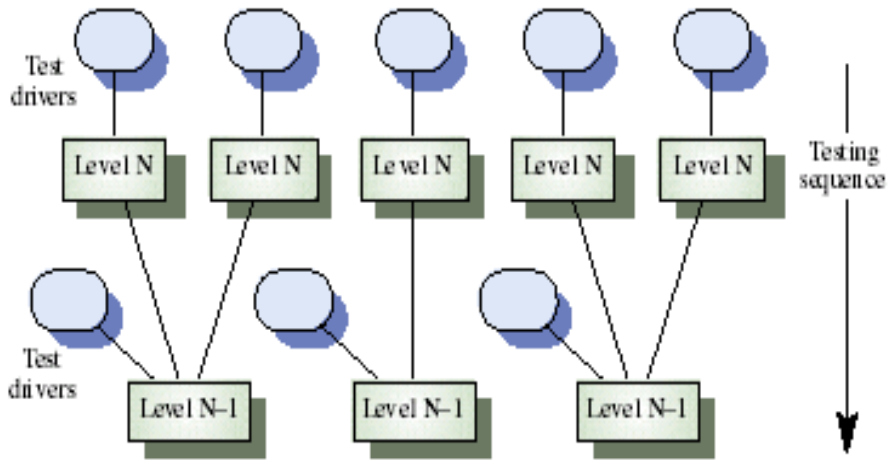


# Top-down testing

- Start with the high-levels of a system and work your way downwards
- Testing strategy which is used in conjunction with top-down development
- Finds architectural errors
- May need system infrastructure before any testing is possible
- May be difficult to develop program stubs



# Bottom-up testing



## Bottom-up testing

- Necessary for critical infrastructure components
- Start with the lower levels of the system and work upward
- Needs test drivers to be implemented
- Does not find major design problems until late in the process
- Appropriate for object-oriented systems

## Stress testing

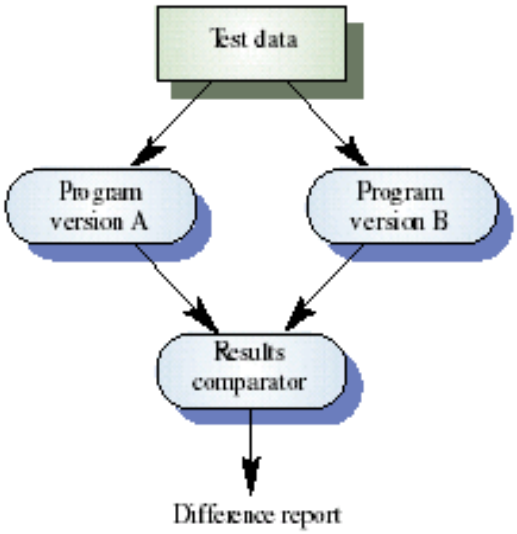
- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light
- Stressing the system test faulure behaviour.. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data
- Particularly relevant to distributed systems which can exhibit severe degradation as a network becomes overloaded

## Back-to-back testing

- Present the same tests to different versions of the system and compare outputs. Differing outputs imply potential problems
- Reduces the costs of examining test results. Automatic comparison of outputs.
- Possible twhen a prototype is available or with regression testing of a new system version



## Back-to-back testing



## Testing in the Small

- แยกเป็น 2 วิธี(approach)
  - **White-box testing** :
    - คือการทำtest โดยสนใจ ว่า internal structure ของ software ที่จะ test
    - ตัวอย่าง โปรแกรมการทดสอบหา Max ระหว่างค่า 2 ค่า

    if x > y then max:=x else max:=x end if;  แล้วกำหนด test sets เป็น {x>y}, {x<=y} เป็น white-box testing
  - **Black-box testing** :
    - คือการทำทดสอบโดยไม่รู้รายละเอียดเกี่ยวกับdesign และ code ของ software ที่กำลัง test
    - พิจารณาจาก Specification หา test set ดู results ที่ได้ เทียบกับ specification
    - ตัวอย่างหน้า 17 เป็น black-box testing

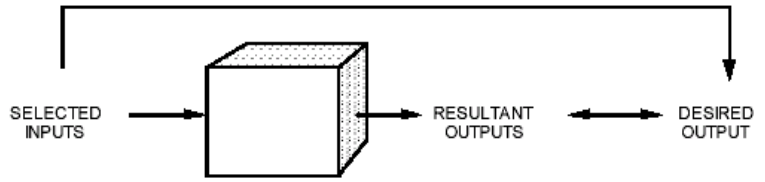


## Functional and Structural Testing

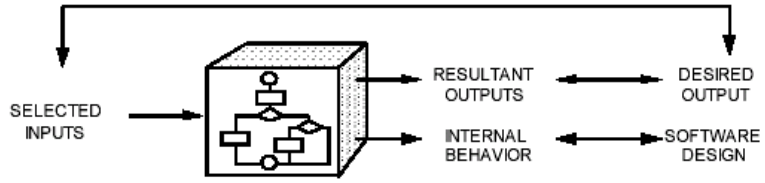
- Functional Testing
  - Test cases selected based on specification
  - Views program/component as *black box*
- Structural Testing
  - Test cases selected based on structure of code
  - Views program /component as *white box* (also called *glass box* testing)

***Can do black-box testing of program by white-box testing of specification***

## Black Box vs. White Box Testing




"BLACK BOX"




"WHITE BOX"

## Structural (White-Box) Test Criteria

- Criteria based on
  - control flow
  - data flow
  - expected faults
- Defined formally in terms of flow graphs
- Metric: percentage of coverage achieved
- Adequacy based on metric requirements for criteria

*Objective:* ***Cover*** *the software structure*

## Flow Graphs

- Control Flow
  - The partial order of statement execution, as defined by the semantics of the language
- Data Flow
  - The flow of values from definitions of a variable to its uses

*Graph representation of control flow and data flow relationships*



## White-box testing

- บางครั้งเรียก Structural testing เนื่องจากใช้ internal structure ในการคำนวณหา test data
- ตัวอย่างการทำwhite-box testing กับโปรแกรม หรม (gcd)

```
begin
   read(x);  read(y) ;
   while x <> y loop
     if x > y then x:=x-y;
     else y:=y-x ;
     end if ;
   end loop ;
   gcd := x ;
end ;
```

{<x = 3, y = 3>, <x = 4, y = 3>, <x = 3, y = 4>}
เราจะหา test case ที่รับประกัน
Statement Coverage Criterion คือ ทุก Statement ถูก execute อย่างน้อยหนึ่งครั้ง

## Statement coverage criterion

- มาจากข้อสังเกตว่า “Error cannot be discovered if the parts of the program containing the error are not executed”
- ข้อเสีย คือ execute once ไม่รับประกันว่าไม่ผิด
- ตัวอย่าง : จงหา test set (x, y) ที่ทำให้ ทุก statement ถูก execute อย่างน้อย 1 ครั้ง

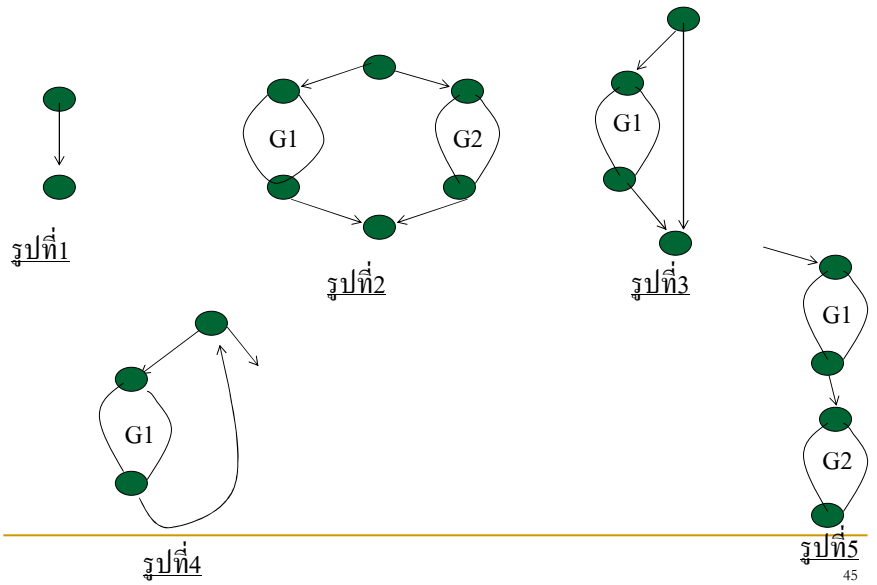
```
read(x); read(y);
if x > 0 then write(1)
else write(2)
end if ;
if y > 0 then write(3)
else write(4)
end if ;
```



## Model Program to be tested by Graph

- การทำ white box testing โดยนำ control flow ของโปรแกรมที่จะ test มาเขียนเป็น graph
- Instruction แต่ละประเภทมีวิธีการสร้าง Graph ต่างออกไป
  - รูปที่1 assignment instruction
  - รูปที่ 2 if-then-else
  - รูปที่ 3 if-then
  - รูปที่ 4 while
  - รูปที่ 5 แทน 2 instruction blocks ต่อกัน
- แต่ละ instruction มี node แทน จุดเริ่มต้น(entry point) และจุดสิ้นสุด(exit point) ของ instruction
- เราใช้ instruction แทน label ของ edge

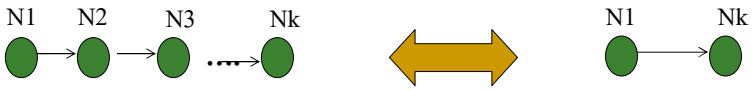
## Model Program to be tested by Graph






# Model Program to be tested by Graph

- ให้ S1 S2 แทน statements ชุดที่1 ชุดที่2 และ G1 G2 แทน graph ของ S1 และ S2
  - รูปที่ 2 แทน instructions : if cond then S1 ; else S2 end if ;
  - รูปที่ 3 แทน instructions : if cond then S1 ; end if ;
  - รูปที่ 4 แทน instructions : while cond loop S1 ; end loop ;
  - รูปที่ 5 แทน instructions : S1 ; S2 ;



# Edge coverage criterion

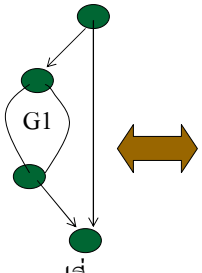
- หลังจากสร้าง Graph ของโปรแกรมที่จะ test แล้ว ใช้หลัก Edge coverage criterion เพื่อหา test set
- การหา test set ที่มีคุณสมบัติตาม Edge coverage criterion คือ select a test set T such that by executing P for each d in T, each edge of P's control flow graph is traversed at least once.
- ความแตกต่างที่ชัดเจนระหว่าง statement coverage criterion กับ edge coverage criterion คือ if instruction ที่ไม่มี else
- Edge coverage จะต้องหา test case ที่ execute edge ที่ if-condition เป็นจริง และ test case ที่ if-condition ไม่เป็นจริง ขณะที่ statement coverage ไม่จำเป็น



# ตัวอย่าง program แสดงข้อบกพร่องของ statement coverage

```
If x < 0 then
        x:= -x
end if ;
z: = x ;
```

**กรณี Statement coverage**
**ต้องการ test case กรณี x negative**




รูปที่3

```
If x < 0 then
        x:= -x
else  null
end if ;
z: = x ;
```

**กรณี Edge coverage**
**ต้องการ test case กรณี x negative และ x positive หรือ 0**

## ตัวอย่างการหา Edge coverage

```
found:=false ;
if number_of_items <> 0 then counter:=1 ;
        while (not found) and (counter < number_of_items) loop
                if table(counter) = desired_element then
                        found:=true;
                end if ;
                counter:=counter + 1;
        end loop ;
end if ;
if found then write("the desired element exists in the table") ;
else write ("the desired element does not exists in the table") ;
end if ;
```



## Discussion about Example

- กรณีนี้มี error ที่ชัดเจนคือใช้ < แทนที่จะเป็น <=
- test set ที่มี test case 2 กรณี คือ
  - กรณี number_of_items = 0
  - กรณี table มี 3 items และ ค่าที่ต้องการหาอยู่ที่ช่องที่ 2

  test set นี้มีคุณสมบัติ edge coverage แต่ detect error ไม่ได้ เนื่องจากเราไม่ได้สนใจกรณี test condition : counter >= number_of_items
- ข้อบกพร่องคือ การไม่ให้ความสำคัญกับ compound condition ใน loop
- ได้แนวคิด criterion เพิ่มสำหรับ edge coverage criterion ที่จะทำให้หา test set ที่มีประสิทธิภาพเพิ่มคือ "Condition coverage criterion"

# Condition coverage criterion

- Select a test set T such that, by executing P for each element in T, each edge of P's control flow graph is traversed, and all possible values of the constituents of compound conditions are exercises at least once.
- ถ้า condition เป็น compound condition ที่ประกอบจากcondition องค์ประกอบย่อย(constituents) C1 and C2 เราต้องทดสอบ ทุกค่าที่ C1และ C2 จะเป็นไปได้ คือ 4กรณี คือ C1 เป็น True หรือ False C2 เป็น True หรือ False
- จากตัวอย่างเราต้องทดสอบกรณี counter >= number_of_items



# Condition coverage enhances Edge coverage โดยการกระจาย conditionให้ หมด(explicit condition of compound condition)

```
If C1 and C2 then
        St ;
else
        Sf ;
end if ;
```

```
If C1 then
        if C2 then
                St ;
        else
                Sf ;
        end if ;
else
        Sf ;
end if ;
```

## ตัวอย่างแสดงข้อบกพร่องของ Edge Coverage Criterion

```
If x <> 0 then  y:= 5 ;
else z:= z - x ;
end if ;
if z > 1 then
        z:= z / x  ;
else z:=0 ;
end if ;
```

- Test set ที่ มีคุณสมบัติตามcondition coverage หรือ edge coverage เช่น {(x=0,z=1),(x=1,z=2)} ไม่สามารถ ครอบคลุมข้อผิดพลาด
- การ test set มีคุณสมบัติ edge coverage อาจได้จากการที่ test case ผ่านเส้นทาง(Path) ใดมาก็ได้
- ในProgram มีหลายเส้นทาง(Paths) ที่เป็นไปได้ Path บาง Path เป็นกรณี error Path ที่ Edge coverage ผ่านอาจจะไม่ใช่ Error Path ดังนั้น Edge Coverage จึงอาจไม่ได้ได้ผ่านกรณี Error
- น.ศ.ลองวาดรูปPathของ ตัวอย่างดู



## Path coverage criterion

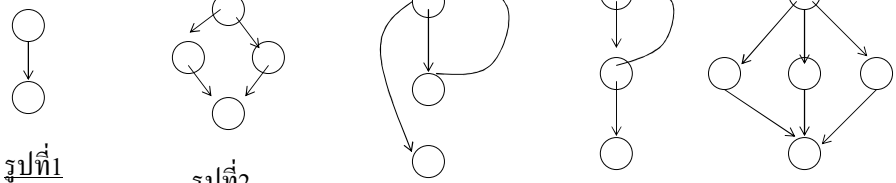
- Path coverage criterion คือ select a test set T such that, by executing P for each d in T, all paths leading from the initial to the final node of P's control flow graph are traversed.
- จากตัวอย่าง เรา execute test case ที่ผ่าน Path else-then ของตัวอย่าง จะdetect error ได้
- ข้อเสียของ Path coverage criterion คือ จำนวน Path มีมากเกินกว่าที่จะ test

# Basis Path Testing

- ทำในลักษณะคล้ายการหา test set ด้วย edge coverage criterion คือ สร้าง graph แต่เน้นที่ node แทน ที่จะเน้นที่ edge
- instruction แต่ละประเภทมีการสร้าง graph ต่างกัน
- สามารถเขียน control flow ของโปรแกรม เป็น flow chart ก่อนแปลงเป็น flow graph ได้
- Node แทน One or More Procedural Statements
- Edge แทน Flow of control
- กรณี compound condition แตกเป็น Nodes หลาย Nodes
- Sequence of process boxes and a decision diamond can map into a single node.



# การสร้าง graph สำหรับ instruction แต่ละประเภท



รูปที่1
SEQUENCE

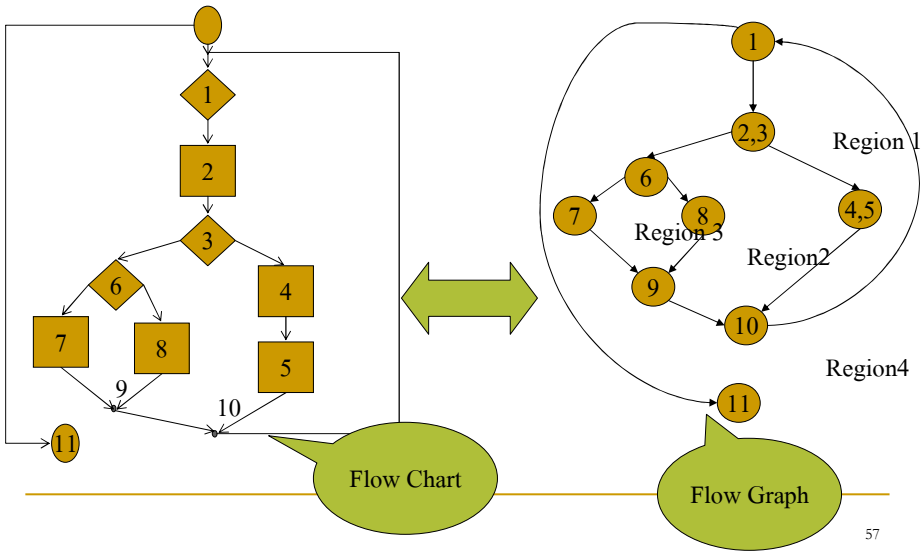
รูปที่2
IF-THEN-ELSE

รูปที่3:
WHILE

รูปที่4:
UNTIL

รูปที่5 :
CASE

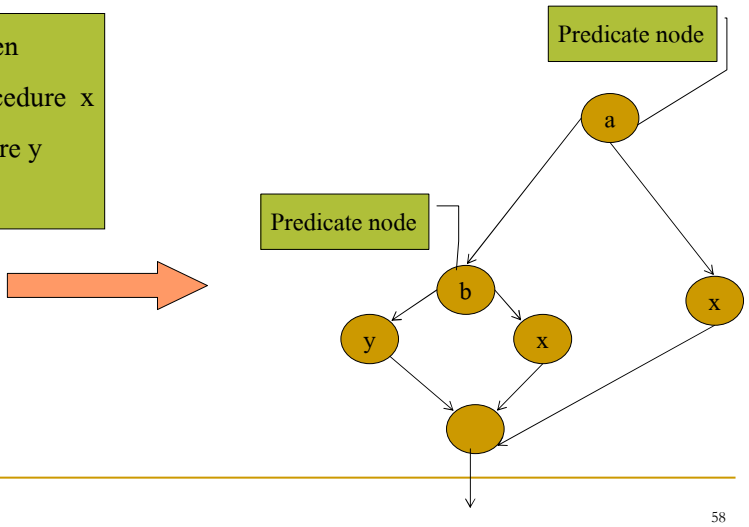
## การสร้าง Flow graph จาก Flow chart





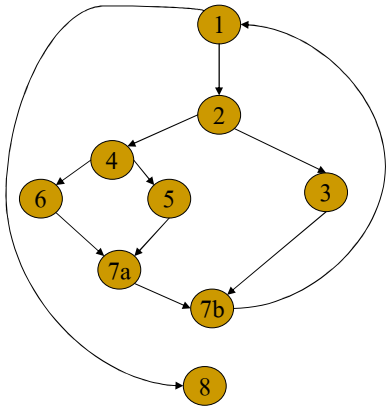

## การสร้าง Graph สำหรับ Compound condition

```
If a OR b then
        procedure  x
else procedure y
end if ;
```



## ตัวอย่างการสร้าง Flow graph จากโปรแกรม sort

```
Procedure : sort
1: do while records remain
          read  record ;
2:        if record  field 1 = 0
3:         then  process record ;
                 store in buffer ;
                 increment counter ;
4:        else if record field 2 = 0 ;
5:           then  reset counter ;
6:           else   process record ;
7a:       end if  end if
7b: end do
8: end
```

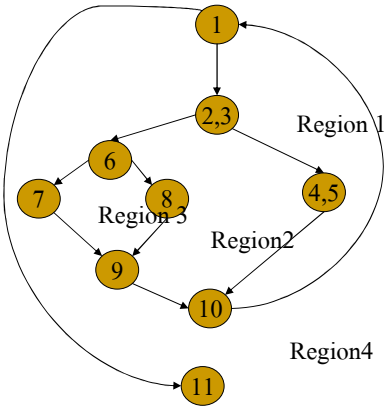





## Cyclomatic Complexity

- Cyclomatic complexity = จำนวนของ test cases ที่ต้องทำเพื่อประกันว่า statement ทุก statement และ ค่าtrue และ false ของทุก condition ถูก execute อย่างน้อยหนึ่งครั้ง
- Cyclomatic complexity ใช้เป็น software metric วัด logical complexity ของ program ได้
- ค่านี้มีในทฤษฎีกราฟ (Graph Theory)
- Cyclomatic Complexity นำมาใช้กับ Basis path testing แทน จำนวนของ independent paths ในโปรแกรม
- Independent Path ของโปรแกรมคือ Path through the program และ introduces at least one new set of processing statements or a new condition.
- Path ที่ไม่ผ่านedge ใหม่ๆไม่ถือเป็น independent path

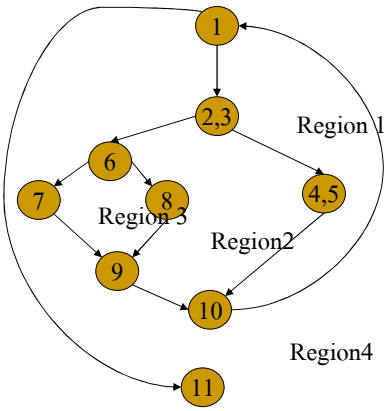
## ตัวอย่างการหา independent paths จาก flow graph




- Independent Paths จากFlow graph ด้านข้าง มี 4 paths คือ
  - path 1: 1-11
  - path 2: 1-2-3-4-5-10-1-11
  - path 3: 1-2-3-6-8-9-10-1-11
  - path 4: 1-2-3-6-7-9-10-1-11
- path: 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 ไม่ถือเป็น independent path



## การคำนวณค่า Cyclomatic Complexity จาก Flow graph




- การคำนวณหาค่า Cyclomatic Complexity ทำได้ 3 วิธี
  - จาก Number of regions of the flow graph
  - จำนวน Edge - จำนวน Node + 2
  - จำนวน Predicate Node + 1
- ใช้วิธีที่ 3 มั่นใจมากกว่า เนื่องจาก จำนวน Node และ Edge อาจเปลี่ยนถ้าเราวาด Flow Graphไม่ดี
- จากตัวอย่าง Cyclomatic complexity = 4 = 4 regions = 11 edges - 9nodes + 2 = 3 predicate nodes + 1

## สรุปขั้นตอนการ derive test case ด้วยวิธี Basis path testing

- Using the design or code as a foundation, draw a corresponding flow graph.
- Determine the cyclomatic complexity of the resultant flow graph.
- Determine a basis set of independent paths.
- Prepare test cases that will force execution of each path in the basis set.



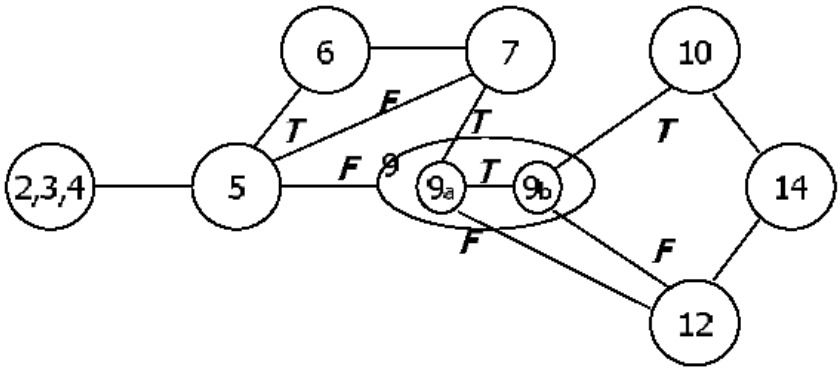
## หลักการการหา test set ที่ได้จาก criterion ต่าง ๆ

- Test loop ต้องทดสอบกรณี
  - ไม่วน loop เลย (zero times)
  - วน loop เป็นจำนวนครั้งที่มากที่สุด (maximum number of times)
  - วน loop เป็นจำนวนครั้งโดยเฉลี่ย (an average number of times)
- Test กรณีมี alternatives หรือ compound condition
  - ทดสอบ path ที่ได้จากค่าที่แตกต่างของ conditions องค์ประกอบย่อย

## A Sample Program to Test

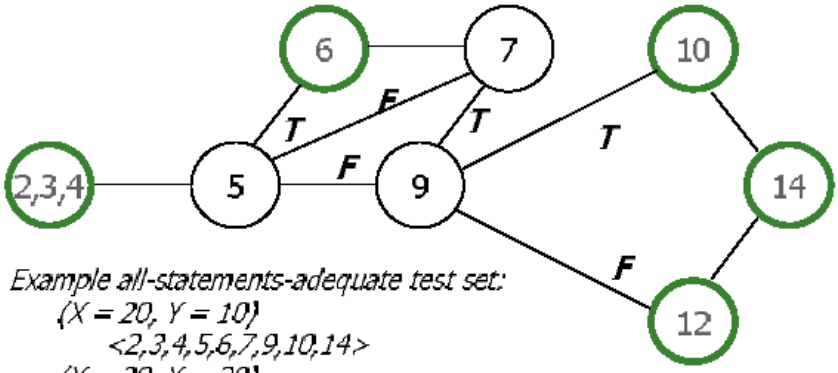
```
1    function P return INTEGER is
2    begin
3        X, Y: INTEGER;
4        READ(X); READ(Y);
5        while (X > 10) loop
6            X := X – 10;
7            exit when X = 10;
8        end loop;
9        if (Y < 20 and then X mod 2 = 0) then
10           Y := Y + 20;
11       else
12           Y := Y – 20;
13       end if;
14       return 2*X + Y;
15   end P;
```

## P's Control Flow Graph (CFG)



## All-Statements Coverage of P

*At least 2 test cases needed*

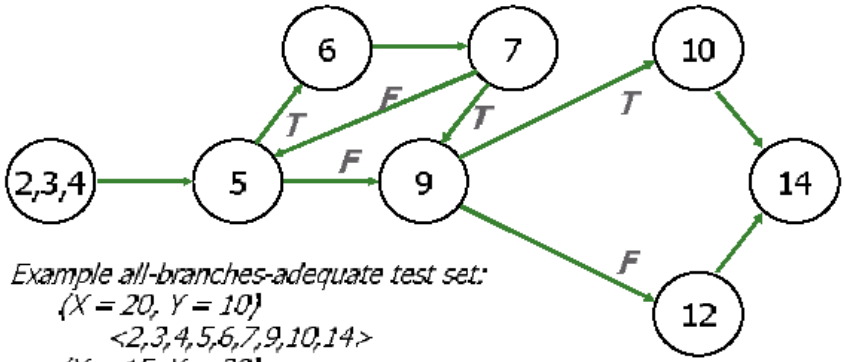



*Example all-statements-adequate test set:*

*(X = 20, Y = 10)*
*<2,3,4,5,6,7,9,10,14>*
*(X = 20, Y = 30)*
*<2,3,4,5,6,7,9,12,14>*

## All-Branches Coverage of P

*At least 2 test cases needed*

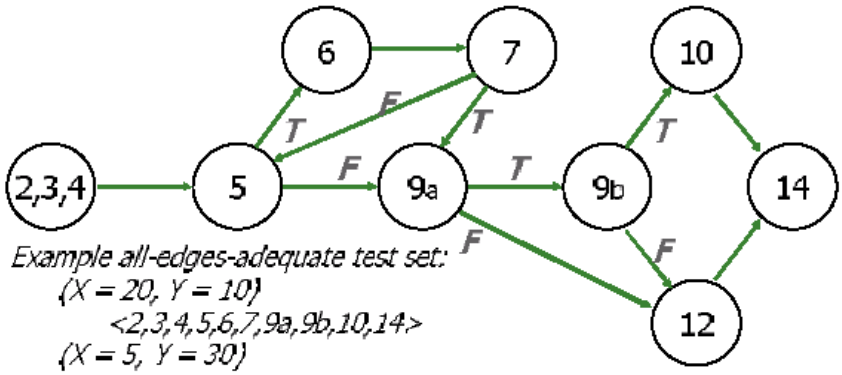



*Example all-branches-adequate test set:*

*(X = 20, Y = 10)*
*<2,3,4,5,6,7,9,10,14>*
*(X = 15, Y = 30)*
*<2,3,4,5,6,7,5,9,12,14>*

# All-Edges Coverage of P

*At least 3 test cases needed*

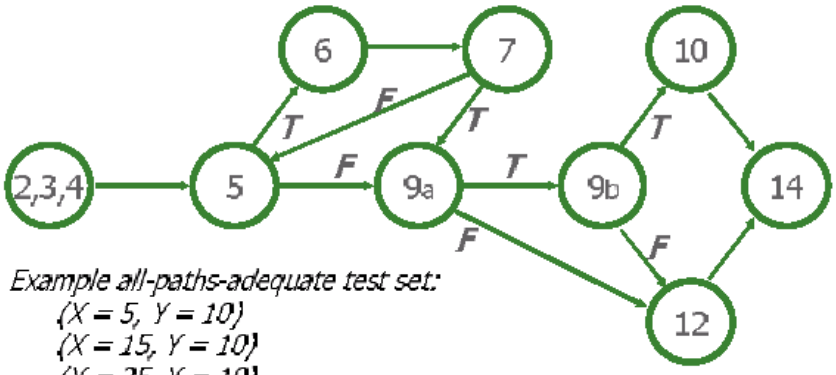



*Example all-edges-adequate test set:*

*(X = 20, Y = 10)*
*<2,3,4,5,6,7,9a,9b,10,14>*
*(X = 5, Y = 30)*
*<2,3,4,5,9a,12,14>*
*(X = 21, Y = 10)*
*<2,3,4,5,6,7,5,6,7,5,9a,9b,12,14>*

# All-Paths Coverage of P

*Infinitely many test cases needed*

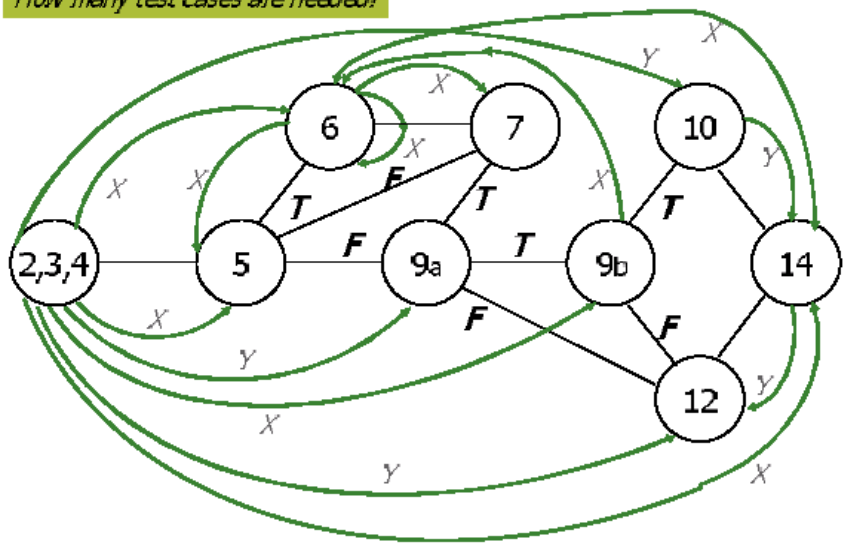



*Example all-paths-adequate test set:*

*(X = 5, Y = 10)*
*(X = 15, Y = 10)*
*(X = 25, Y = 10)*
*(X = 35, Y = 10)*
*...*

## All-Uses Coverage of P

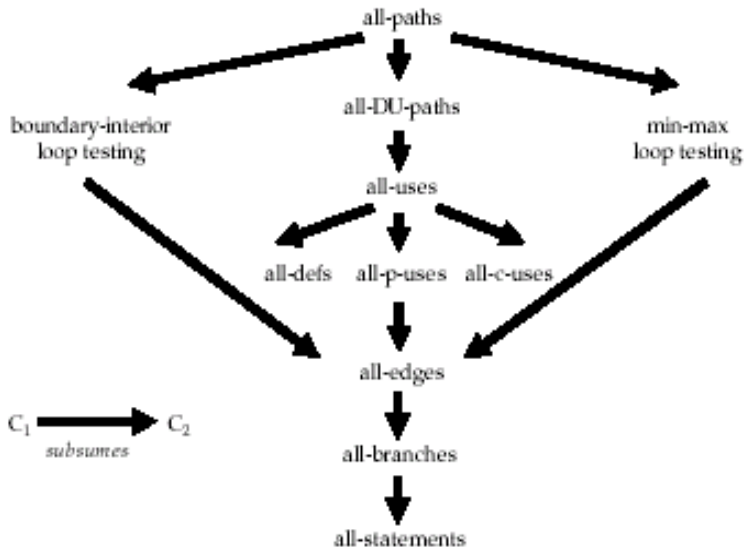
*How many test cases are needed?*




## Subsumption of Criteria

- Criterion C1 subsumes criterion C2 if any C1-adequate test set T is also C2-adequate
  - But some T1 satisfying C1 may detect fewer faults than some T2 satisfying C2

## Structural Subsumption Hierarchy




## The test generation problem

How to generate test data

Partition testing: divide program in (quasi-) equivalence classes

- random
- functional (black box)
  - based on specifications
- structural (white box)
  - based on code
- fault based
  - based on classes of faults

## The termination problem

How to decide when to stop testing

- The main problems for managers
- When resources (time and budget) are over
  - no information about the efficacy of the test
  - BUT... resource constraints must be taken into account
- When some coverage is reached
  - no assurance of software quality
  - it can be a reasonable and objective criterion
  - It can be (partially) automated



## Key points

- Verification and validation are not the same thing
- Testing is used to establish the rpesence of defects and to show fitness for purpose
- Testing activities include unit testing, module testing, sub-system testing, integration testing and acceptance testing
- Object classes should be testing in O-O systems

## Key points

- Testing should be scheduled as part of the planning process. Adequate resources must be made available
- Test plans should be drawn up to guide the testing process
- Testing strategies include top-down testing, bottom-up testing, stress testing, thread testing and back-to-back testing