Software Testing (I)

Jianyong Wang(王建勇)
Department of Computer Science and Technology
Tsinghua University, Beijing, China

Outline

- Software Quality Assurance
- Basic Testing Concepts
- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
- Other types of Testing

Software Quality Assurance

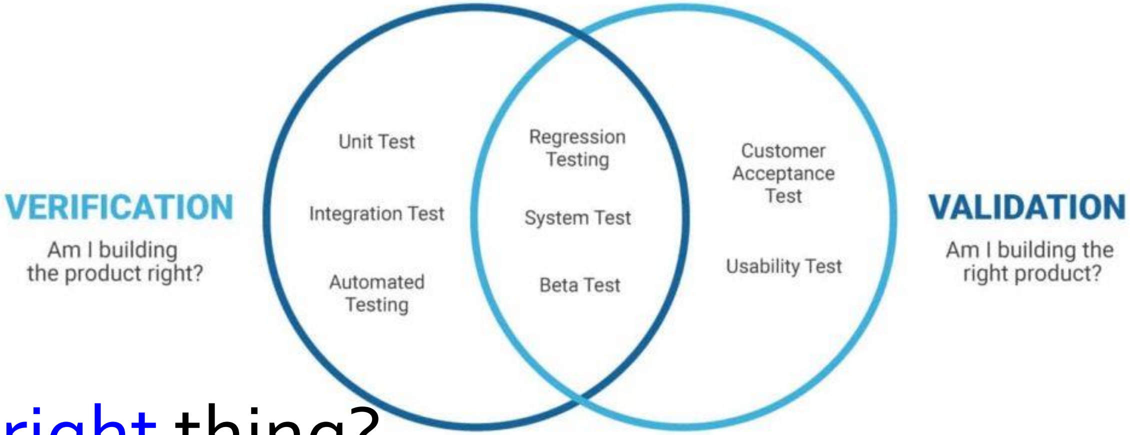
Software Quality

- Quality == "fitness for use"
  - Business value for customer *and* manufacturer
  - *Quality Assurance* : processes/standards to *produce* product and to *improve* quality over time

- Software quality:
  1. Satisfies customers' needs—easy to use, gets correct answers, does not crash, …
  2. Easy for developer to debug and enhance

SW Quality Assurance: Verification and Validation

- Verification（验证）: Did you build the thing right?
  - Did you meet the specification?
    - A test of a system to prove that it meets all its specified requirements at a particular stage of its development[IEEE-STD-610]



- Validation（确认）: Did you build the right thing?
  - Is this what the customer wants?
  - Is the specification correct?
    - An activity that ensures that an end product stakeholder's true needs and expectations are met[IEEE-STD-610]

3 Options for Verification and Validation

- 3 options for Verification and Valdiation

Software testing
Dynamic V&V
Execute the software with test data and examine the outputs

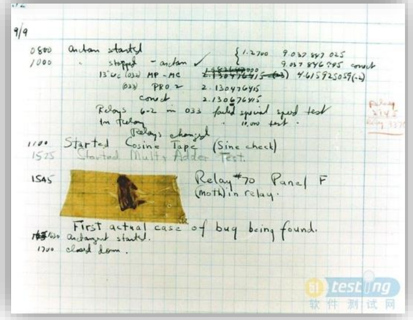
Peer Review
Code Review by others
Pair Programming: driver and observer
Walkthrough: author leads discussion
Technical Review: by a team

Software inspection (formal method)
Static V&V
Analyze and check system representations, such as specification, model, code

Software Testing

Grace Hopper

BUG



9/9

0800 Arctan started

1000 " stopped - arctan ✓ {1.2700 9.037 847 025

9.037 846 995 correct

13"uc (032) MP - MC ~~2.130476415~~ (-3) 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10,000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

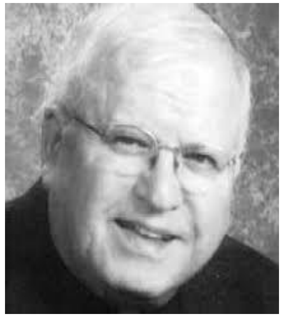
1630 Arctangent started.

1700 closed down.

1945年

Bill Hetzel

Prove it works



The Complete Guide to SOFTWARE TESTING

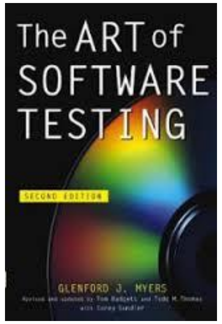
Second Edition

Bill Hetzel

1972年

G. J. Myers

Prove it Does NOT work



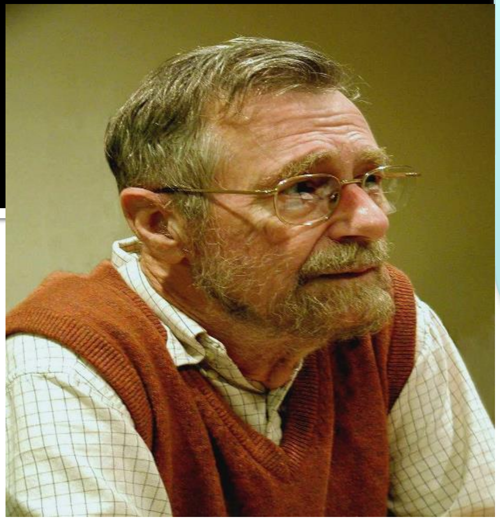
The ART of SOFTWARE TESTING

SECOND EDITION

GLENFORD J. MYERS

Revised and updated by Tom Badgett and Todd M. Thomas with Corey Sandler

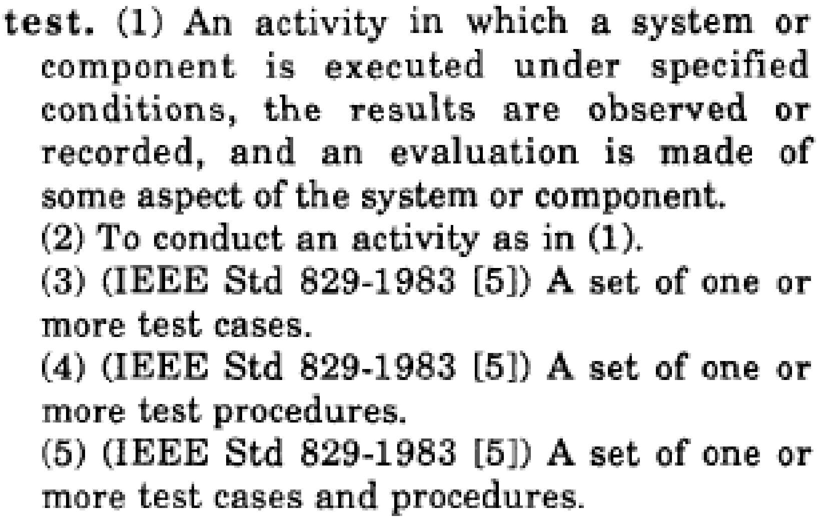
1979年



Testing can never demonstrate the absence of errors in software, only their presence

Edsger W. Dijkstra

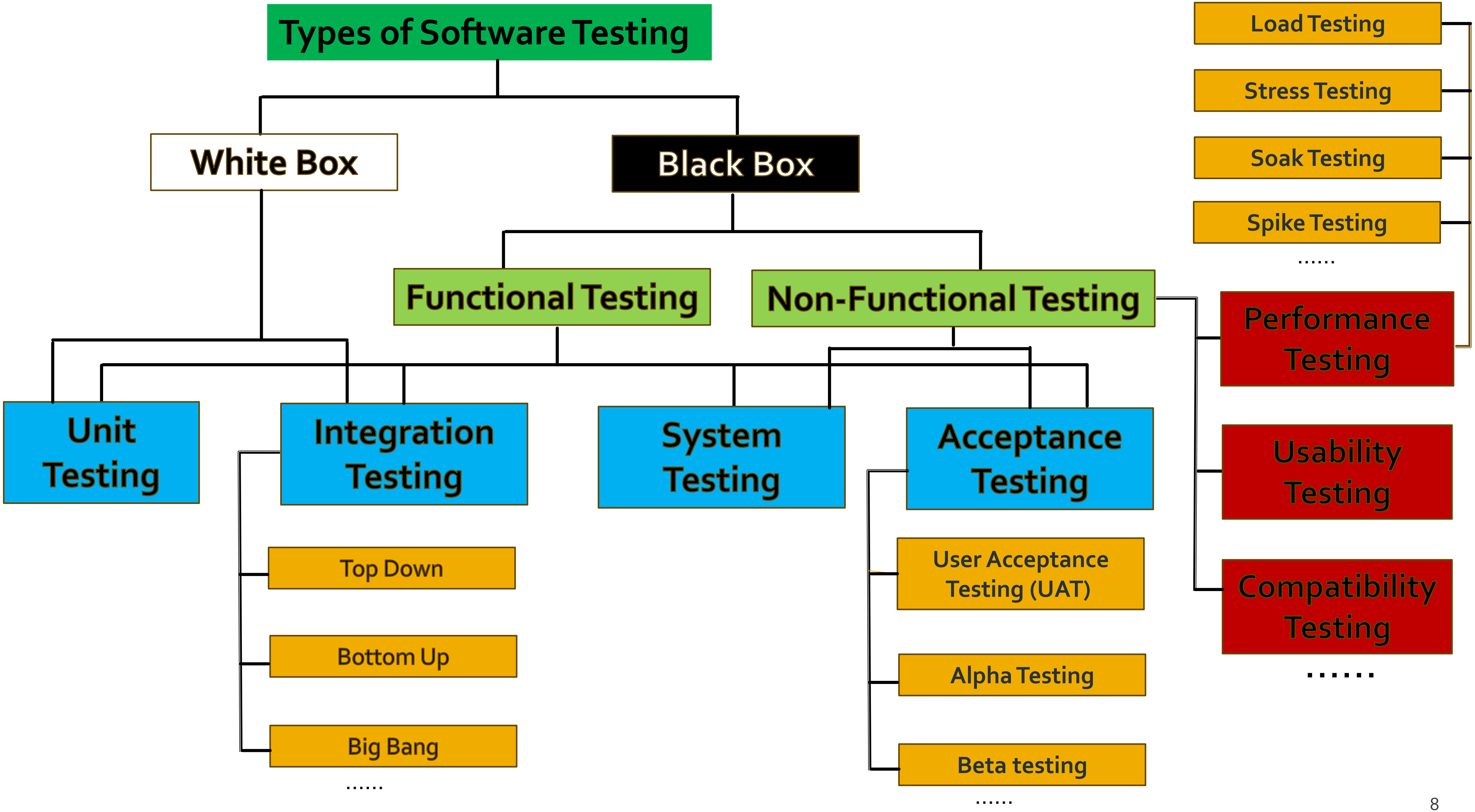
IEEE std.610.12



test. (1) An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
(2) To conduct an activity as in (1).
(3) (IEEE Std 829-1983 [5]) A set of one or more test cases.
(4) (IEEE Std 829-1983 [5]) A set of one or more test procedures.
(5) (IEEE Std 829-1983 [5]) A set of one or more test cases and procedures.

1990年

Types of Software Testing



White Box

Black Box

Functional Testing

Non-Functional Testing

Unit Testing

Integration Testing

System Testing

Acceptance Testing

Top Down

Bottom Up

Big Bang

......

User Acceptance Testing (UAT)

Alpha Testing

Beta testing

......

Performance Testing

Usability Testing

Compatibility Testing

......

Load Testing

Stress Testing

Soak Testing

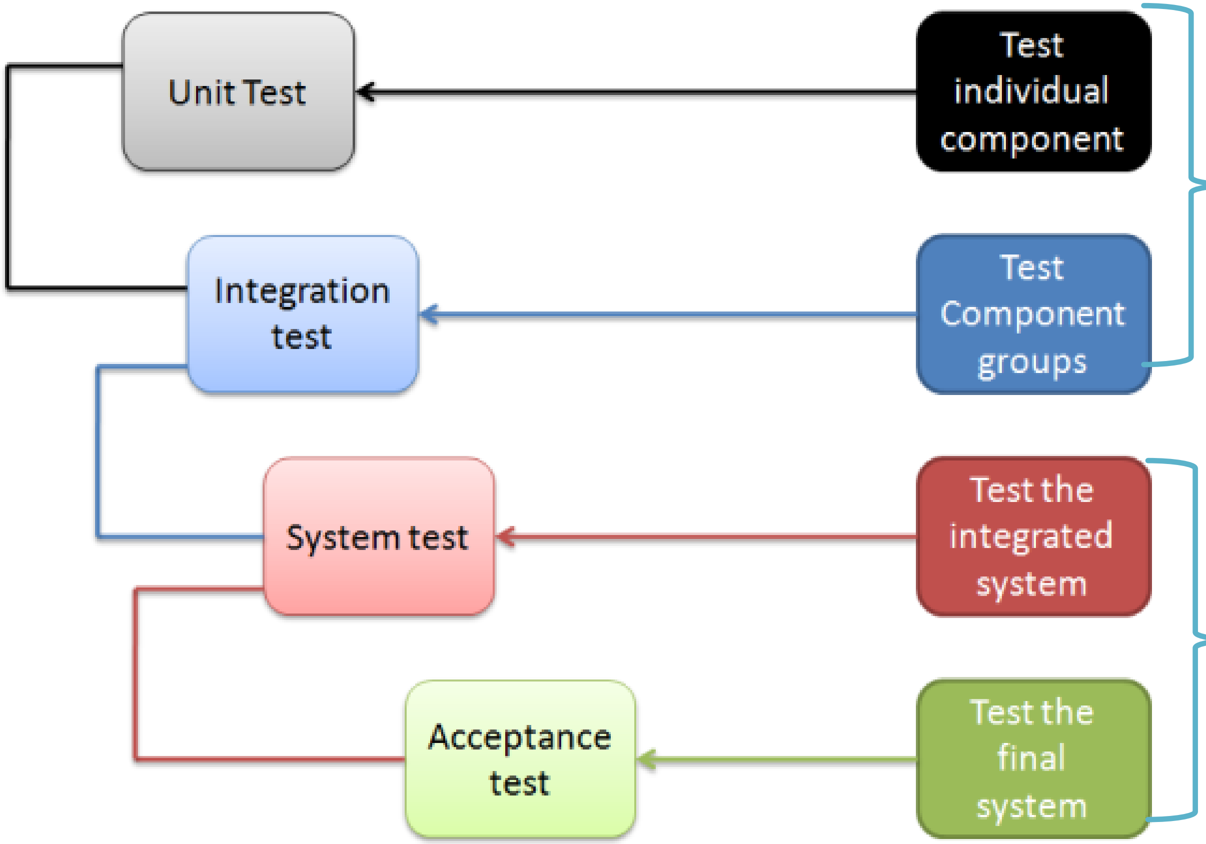
Spike Testing

......

Levels of Software Testing

(Each Test)
shorter run time, fewer cost,
less dependency

More Tests



White-Box Testing （code-based)
or Black-Box Testing (interface/specification-based)

Black-Box Testing (interface/specification-based)

Less Tests

(Each Test)
longer run time, more cost,
more dependency

Outline

- Software Quality Assurance
- Basic Testing Concepts
- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
- Other types of Testing

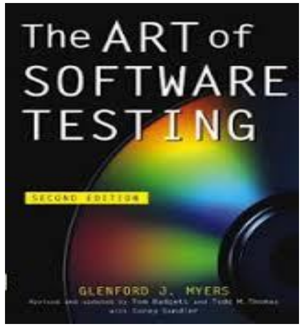
Basic Testing Concepts

Testing Objectives

- Testing is the process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet undiscovered error.
- A successful test is one that uncovers an as-yet undiscovered error.

G. J. Myers

Prove it Does NOT work



1979年

Testing Principles

- All tests should be traceable to customer requirements
- Tests should be planned long before testing begins
- The Pareto (20-80) principles applies to software testing
  - Defect clustering : approximately 80% of the problems are found in 20% of the modules
  - About 80 percent of errors and crashes come from 20 percent of the most frequent bugs
- Testing should begin "in the small" and progress toward testing "in the large"
- Exhaustive testing is not possible

201 Principles of Software Development,
A. Davis, McGraw-Hill, 1995.

Tests should be FIRST

- **F**ast: run (subset of) tests quickly (since you'll be running them *all the time*)
- **I**ndependent: no tests depend on others, so can run *any subset* in *any order*
- **R**epeatable: consistent results every run (to isolate bugs and enable automation)
- **S**elf-checking: test can *automatically* detect if passed (*no human checking* of output)
- **T**imely: written about the same time as code under test (with TDD, written *first!*)

The Challenges

How to design test cases?

How do we know when we have tested enough?

A Test Exercise

- The NextDate Function

  The NextDate function accepts three parameters as inputs: the year, month and day. It returns the next date to the input date.

  Please design your test cases.

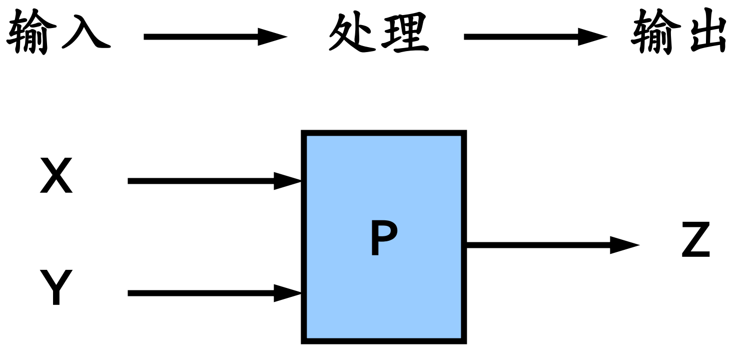
The Basic Concepts

- Software Testing
  - Testing is the process of executing a program with the intent of finding an error.
- Test Case
  - Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.

Example nextDate test cases

Test Case	**Input**			**Expected Output**
	Month	**Day**	**Year**	
1	**1**	**31**	**1812**	**1812-2-1**
2	**2**	**28**	**2000**	**2000-2-29**
3	**2**	**20**	**2001**	**2001-2-21**

Exhaustive Testing is Impossible



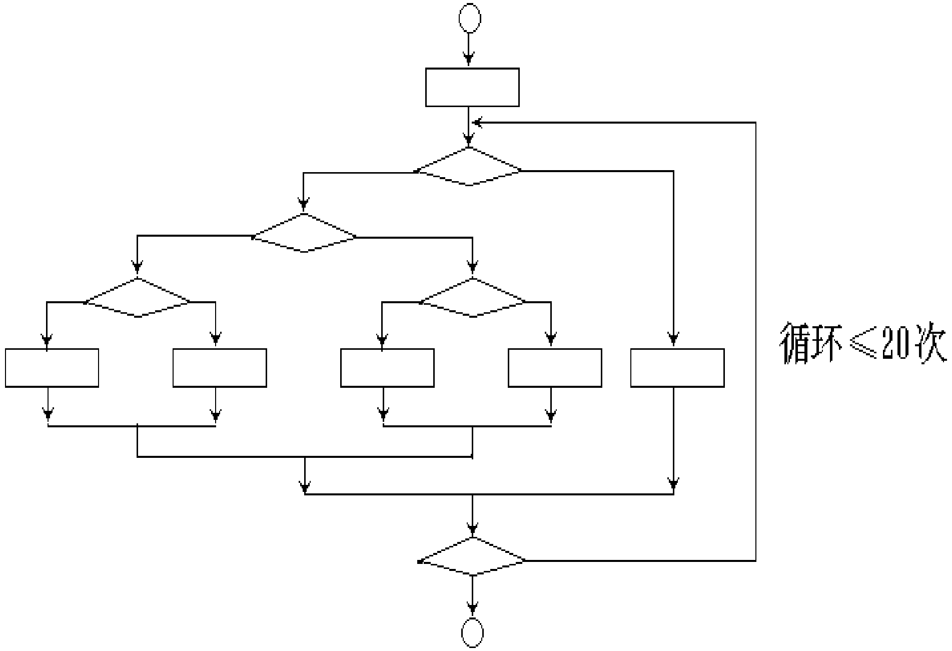
X、Y：32位整数

可能采用的测试数据组：

2^{32}X2^{32}=2^{64}

测试时间（假设每个测试用例1毫秒，一年工作365x24小时）：

约5亿年



不同执行路径：5^{20}

假设每条路径测试时间为1毫秒，
每年的工作时间为24*365小时，
则测试时间约为3170年。

Coverage-Based Testing

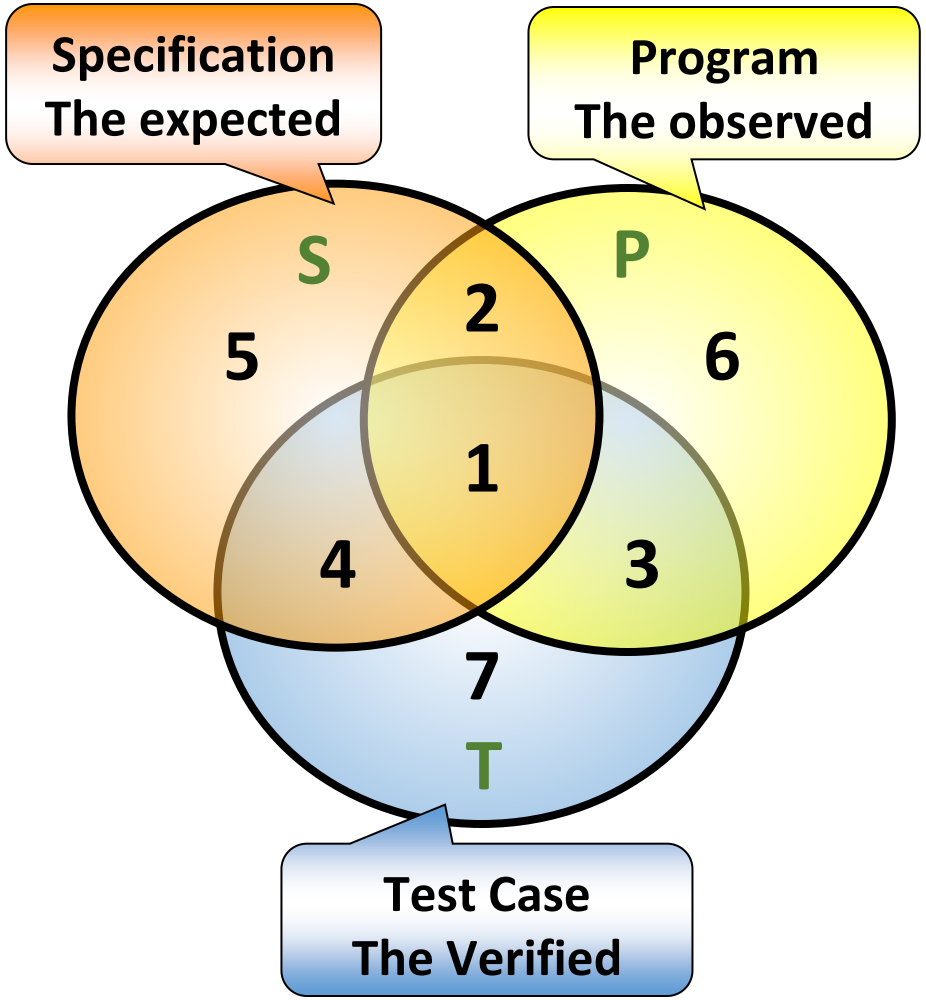
- Goodness is determined by the coverage of the product by the test set
  - E.g. % of statements or requirements are tested

- Typical metrics
  - Control-flow coverage
  - Data-flow coverage

    } **White-Box (Structure-Based) Testing**
  - Requirement coverage → **Black-Box (Requirement-Based) Testing**

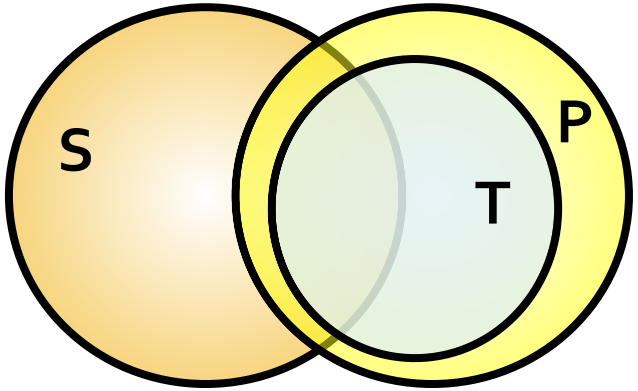
The Testing Venn Diagram

- **S**: the expected behavior of the system defined by the specification
- **P**: the behavior exposed by the system implementation
- **T**: The behavior that detected by test cases

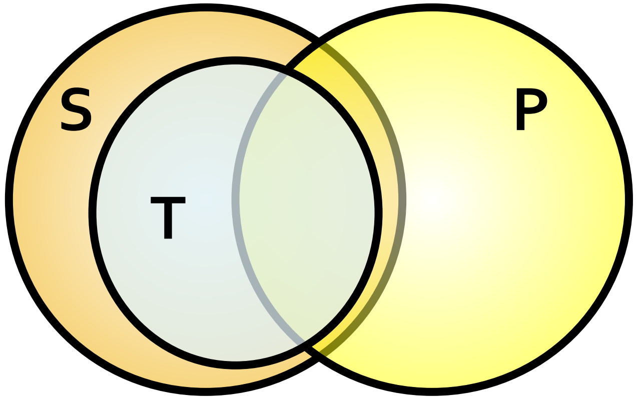


White Box and Black-Box Testing

- **White-box testing**
  - Knowing the internal workings of a product
  - Testing based on the code
  - Code-based coverage



- **Black-box testing**
  - Knowing the specified function that a product has been designed to perform
  - Testing based on the interface
  - Specification-based coverage



White-Box Testing: Control Flow Testing

An Example

```
If (A>B)  and (C==5)
      then do P1;
D = 5;
```

1. (A > B) and (C == 5)
2. (A > B) and (C <> 5)
3. (A <= B) and (C == 5)
4. (A <= B) and (C <> 5)

```
If (A>B)  and (C==5)
      then do P1;
D = 5;
```

Statement Coverage
语句覆盖

```
If (A>B)  and (C==5)
      then do P1;
D = 5;
```

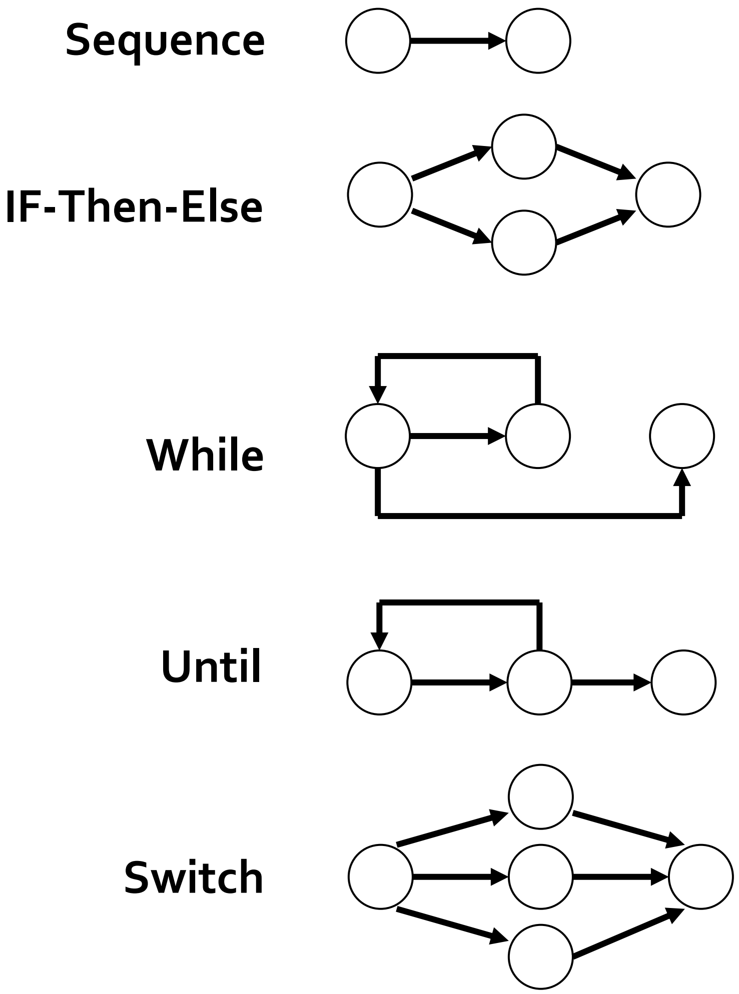
Decision Coverage
判定覆盖

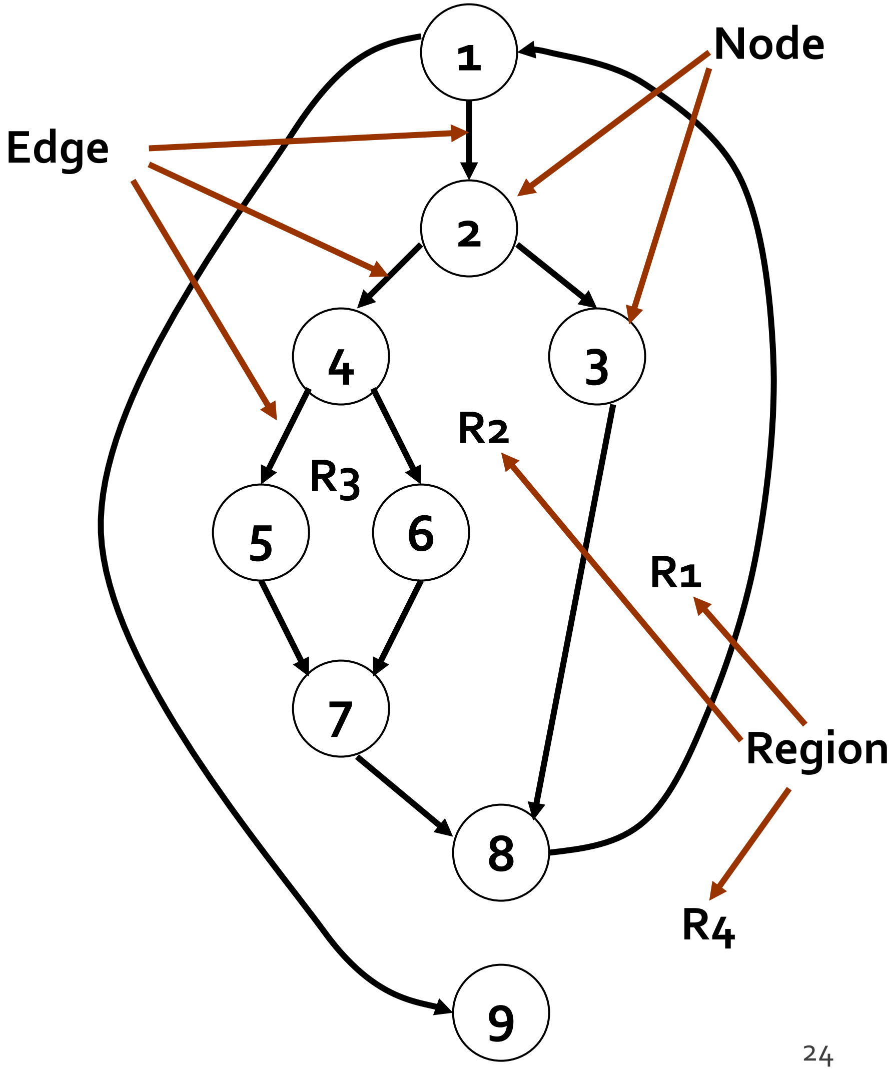
```
If (A>B)  and (C==5)
      then do P1;
D = 5;
```

Predicate (condition) Coverage
条件覆盖

Flow graph notation

- Circle: called node, one or more procedure statements
- Links (edges): flow of control
-
- Region: Area bounded by edges and nodes
- Degree of node
  - Outdegree: the number of links initiated from the node
  - Indegree: the number of links terminated at the node



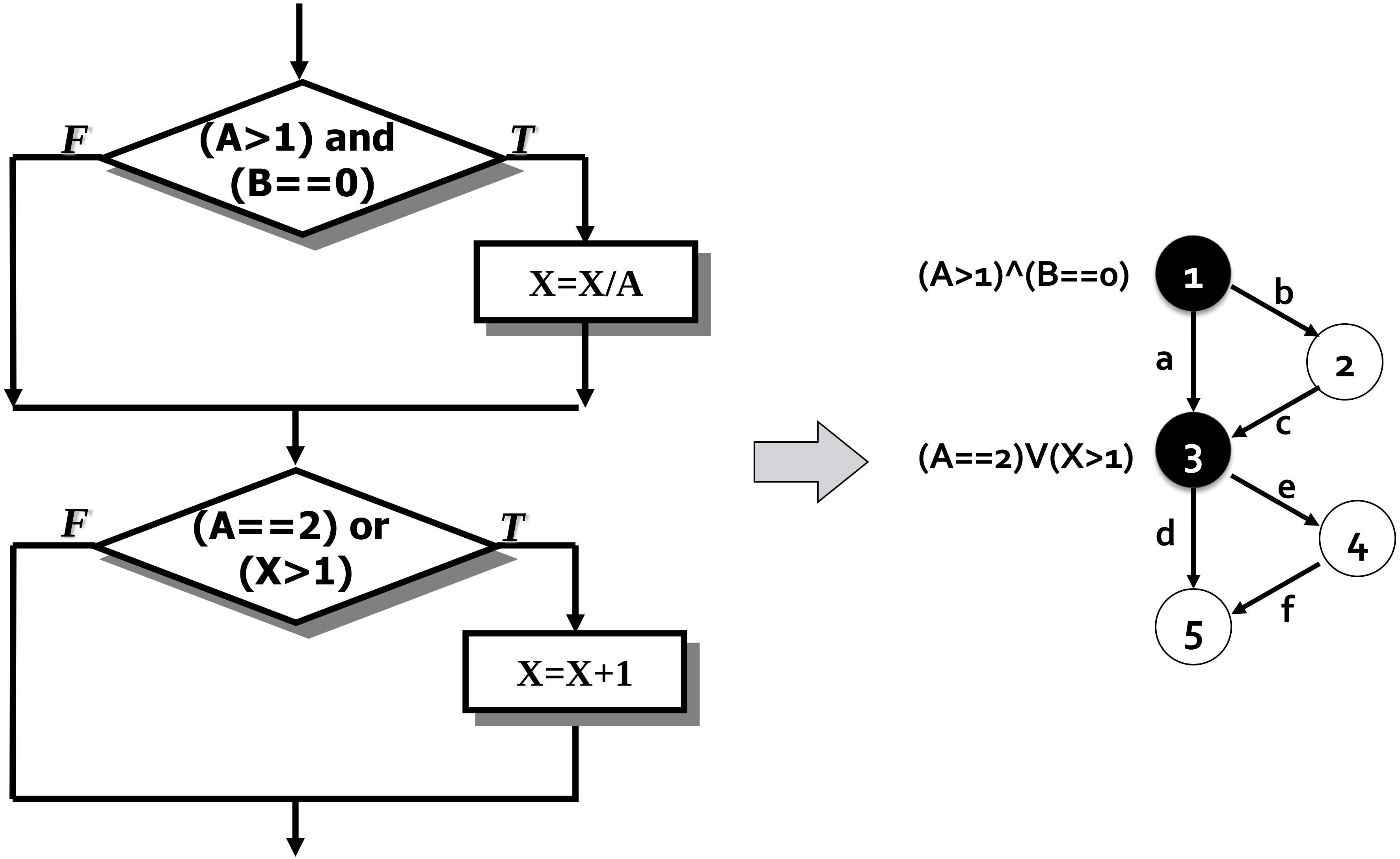


Path Coverage

Metric	Coverage Description
C_0	Every statement (语句覆盖)
C_1	Every DD-path (Decision-to-Decision path) (判定覆盖)
$C_{1,P}$	Every predicate of each decision (条件覆盖)
C_2	C_1 + loop coverage
C_d	C_1 + every dependent pair of DD-path
C_{MCC}	Multiple condition coverage(多重条件覆盖)
$C_{i,K}$	Every program path that contains to K repetitions of a loop
C_{stat}	“Statistically significant” fraction of paths
C_{∞}	All possible execution paths(全路径覆盖)

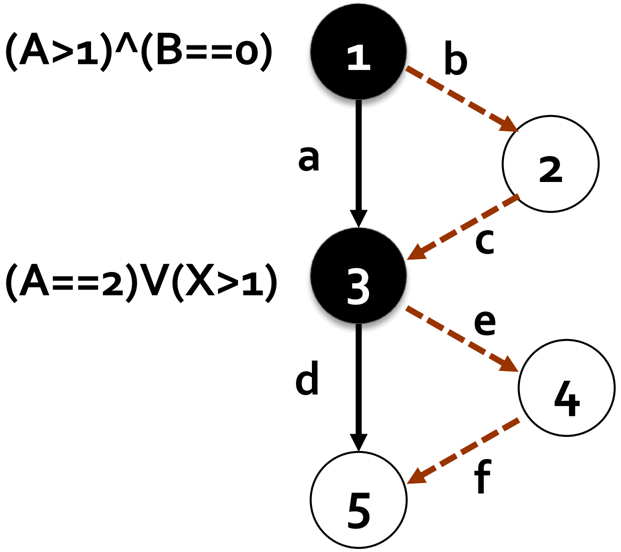
E. F. Miller, 1977

An Example



Statement Coverage

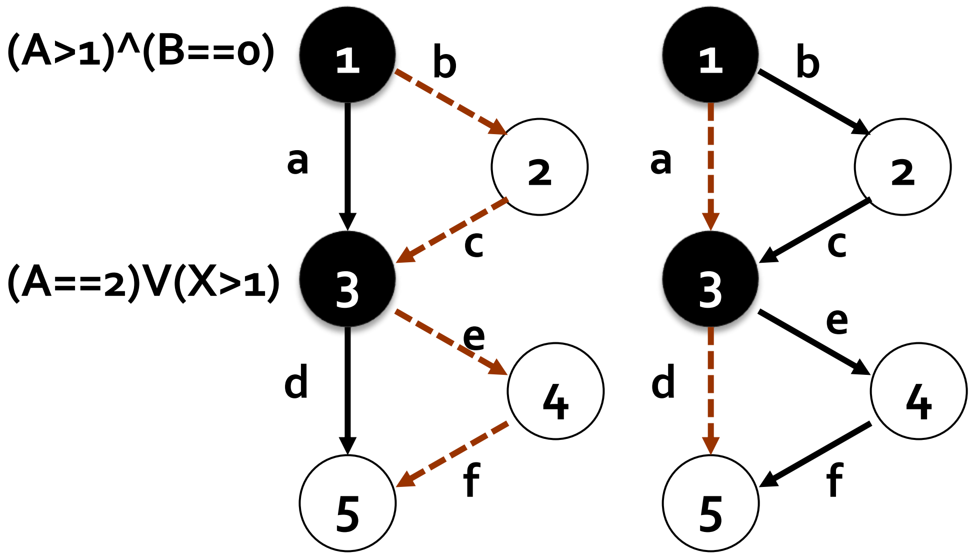
- Criterion
  - All statements must be covered during test execution
- Procedure
  - Find paths that cover all statements
  - Choose input data that will result in the selected paths



(A>1)^(B==0)	(A==2)V(X>1)	Path	Test Case
1	1	b->c->e->f	A=2,B=0,X=4

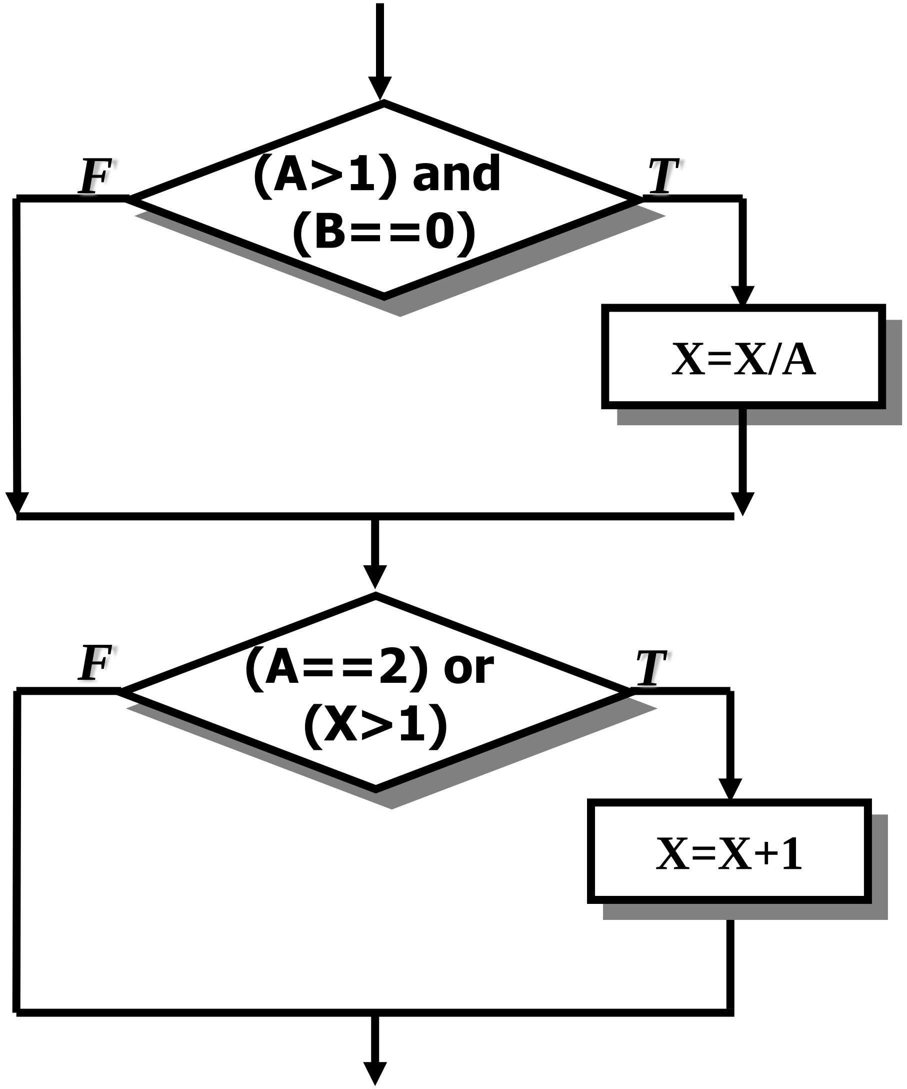
Decision Coverage (Branch Coverage)

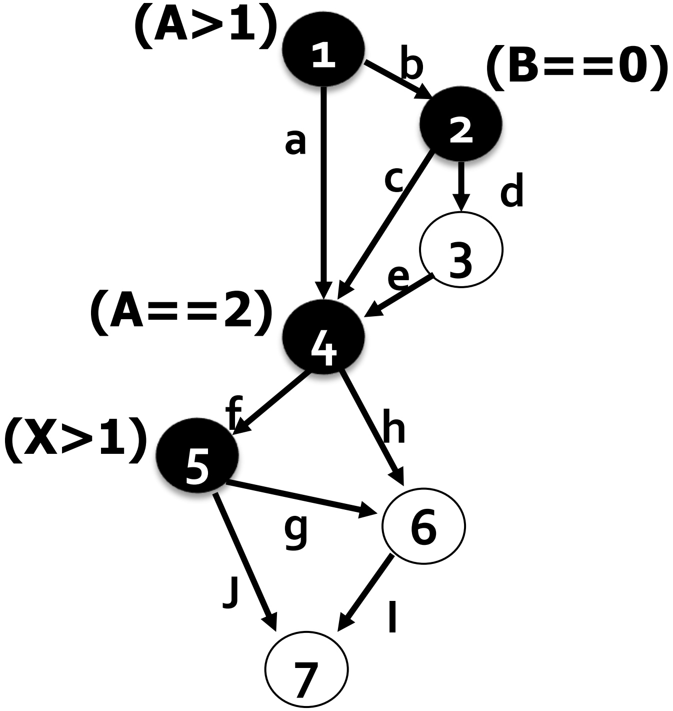
- Criterion
  - At any branch point, each branch must be covered during test execution.
    - The *true* and *false* branch of a 2-way IF statement
    - Each *case* in a SWITCH statement
- Procedure
  - Find paths that cover all branches
  - Choose input data that will result in the selected paths.



(A>1)^(B==0)	(A==2)V(X>1)	Path	Test Case
1	1	b->c->e->f	A=2,B=0,X=4
0	0	a->d	A=1,B=1,X=1

Predicate Coverage (Condition Coverage)



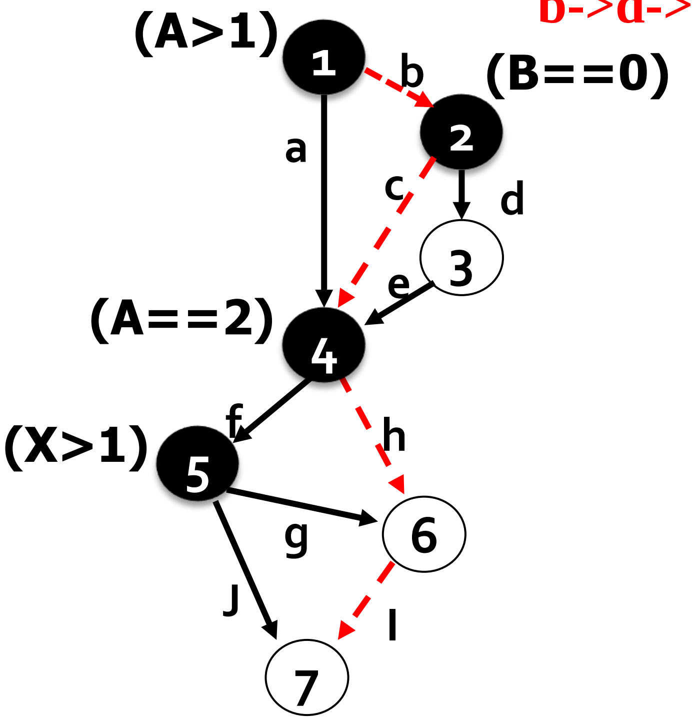


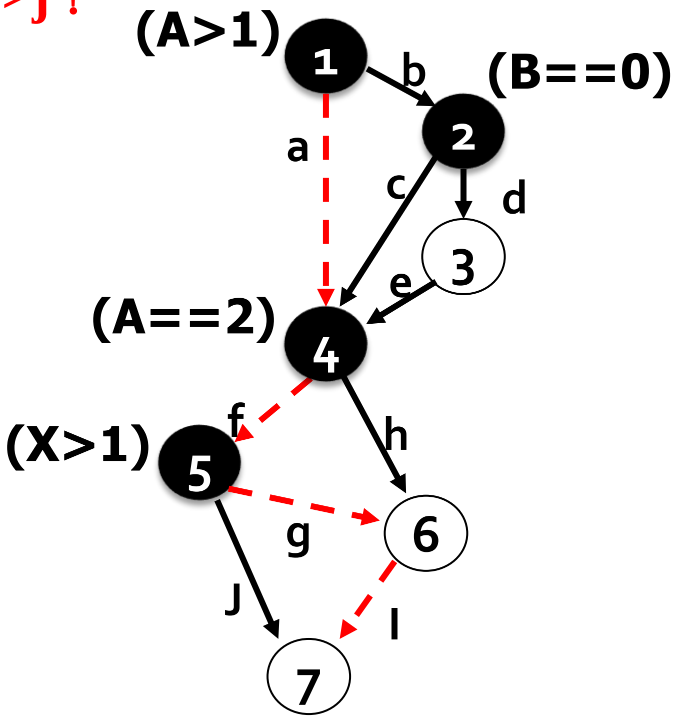
- Criterion
  - The branches of every atomic condition (i.e. does not include AND or OR) must be covered during test execution.
- Procedure
  - Decompose each decision into atomic conditions
  - Draw the flow graph with nodes of conditions
  - Find paths that cover all branches
  - Choose input data that will result in the selected paths.

Predicate Coverage (Condition Coverage)

A>1	B==0	A==2	X>1	Path	Test Case
1	0	1	0	b->c->h->I	A=2,B=1,X=1
0	1	0	1	a->f->g->I	A=1,B=0,X=3

b->d->e->f->j ?

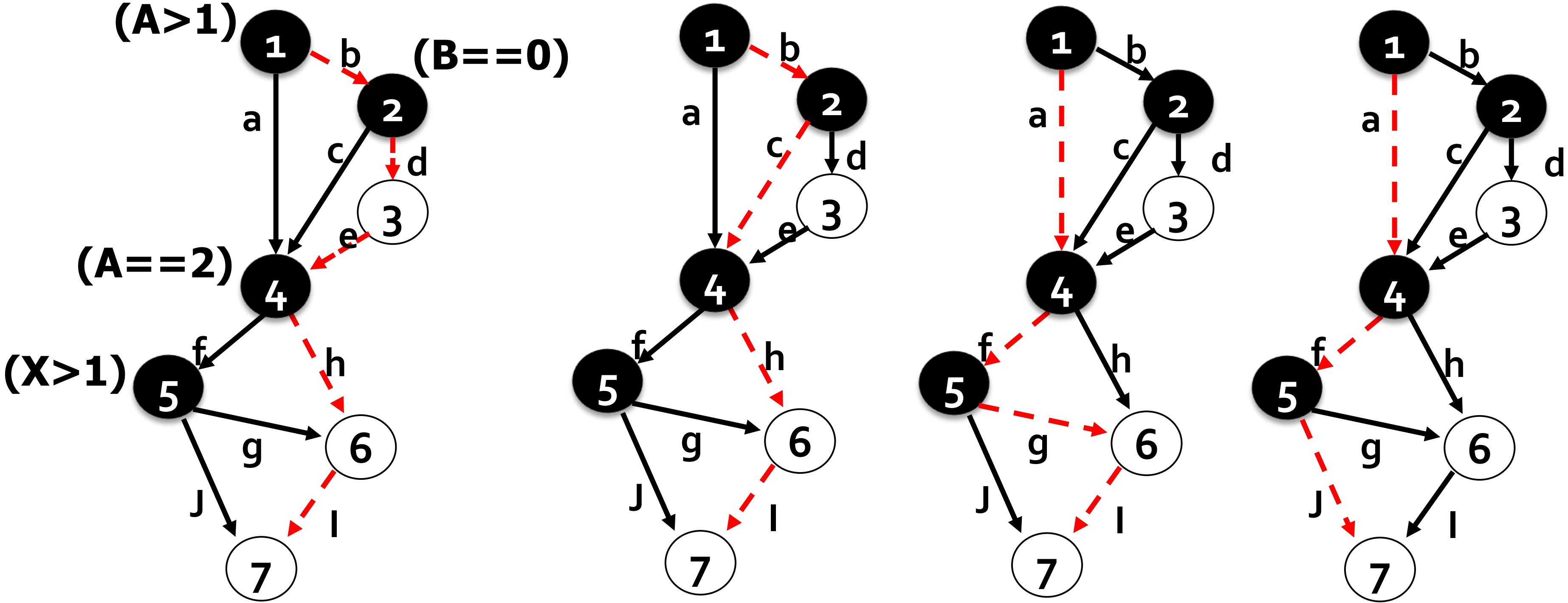




- Criterion
  - The branches of every atomic condition (i.e. does not include AND or OR) must be covered during test execution.
- Procedure
  - Decompose each decision into atomic conditions
  - Draw the flow graph with nodes of conditions
  - Find paths that cover all branches
  - Choose input data that will result in the selected paths.

Multiple Condition Coverage

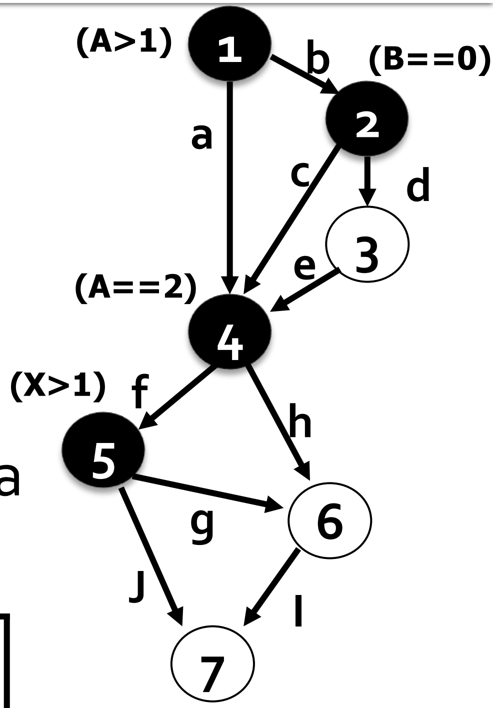
A>1	B==0	A==2	X>1	Path	Test Case
1	1	1	1	b->d->e->h->I	A=2,B=0,X=4
1	0	1	0	b->c->h->I	A=2,B=1,X=1
0	1	0	1	a->f->g->I	A=1,B=0,X=3
0	0	0	0	a->f->J	A=1,B=1,X=1



- Criterion
  - In a compound decision, every combination of atomic conditions must be covered during test execution.
- Procedure
  - Decompose each decision into atomic conditions
  - Find all the combinations of conditions
  - Find paths that cover all combinations of conditions
  - Choose input data that will result in the selected paths.

All Path Coverage

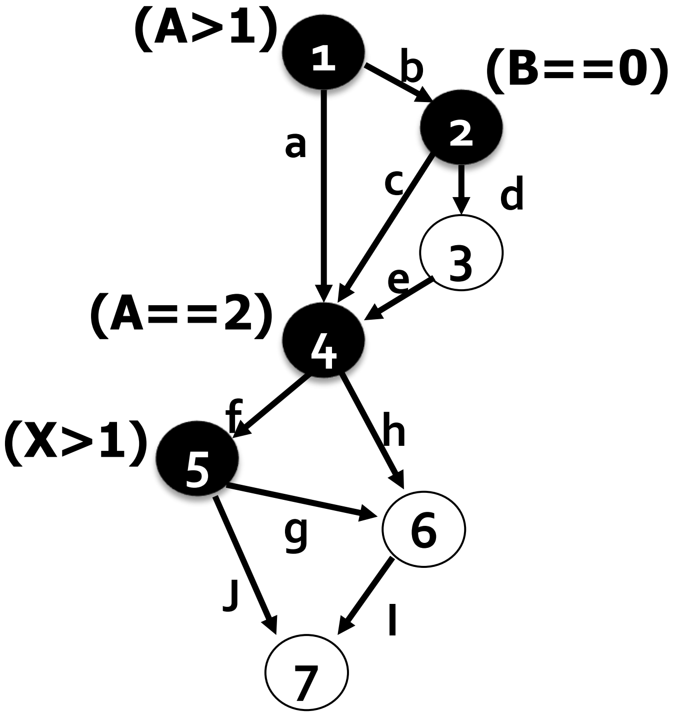
- Criterion
  - All paths through the code must be covered.
- This is typically infeasible when loops present
  - A version of this coverage with loops is to treat loops as having two paths:
    - The loop is executed (normally, once)
    - The loop is skipped
- Some paths may also be infeasible because there is no combination of data conditions that permit a path to be taken.

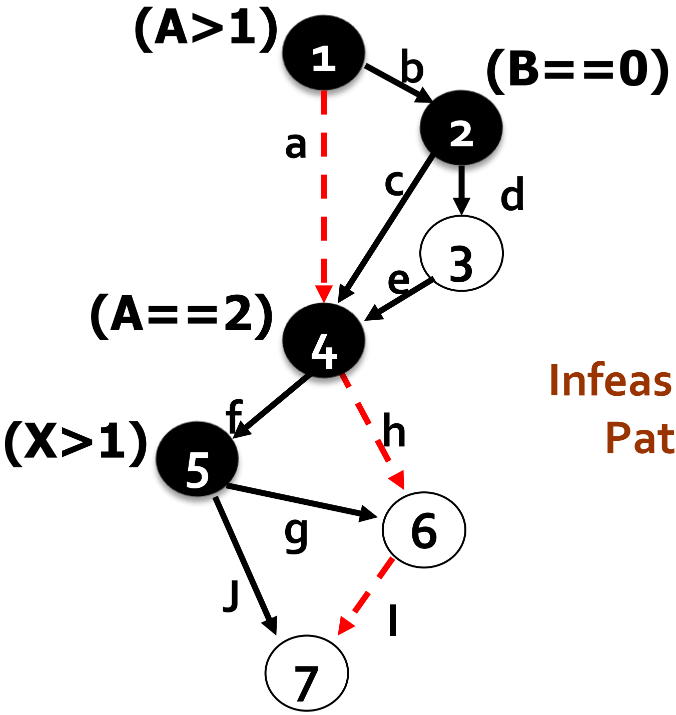


A>1	B==0	A==2	X>1	路径	测试用例
1	1	1	1	b->d->e->h->l	A=2,B=0,X=4
0	0	0	0	a->f->J	A=1,B=1,X=1
0	0	0	1	a->f->g->l	A=1,B=1,X=2
1	1	0	0	b->d->e->f->J	A=3,B=0,X=3
……				……	……

All Path Coverage

A>1	B==0	A==2	X>1	路径	测试用例
......				......	......
0	-	1	-	a->h->l	?
......				......	......





Infeasible Path

Black-Box Testing: Equivalent Partition & Boundary Value Analysis

Equivalence Partitioning

- To divide the input domain of a program into classes of data called equivalent classes
  - Equivalent classes are mutual exclusive
  - Data in the same class are “equivalent”, i.e., the program behaves in an equivalent way from defect revealing perspective
- Test cases are derived from each equivalent class
- **Key:** equivalent class identification

EP Testing Strategy

- Weak vs. Strong Testing
  - Weak testing
    - The single defect hypothesis: Only one of the input can cause invalid output
    - Cover all but no combinations of input
  - Strong testing
    - Multiple defects hypothesis: There may exist multiple inputs concurrently to cause invalid output
    - Cover all combinations of input
- General vs. Robust Testing
  - General: Only the valid inputs are covered
  - Robust: To cover valid as well as invalid inputs

Weak General EP Testing

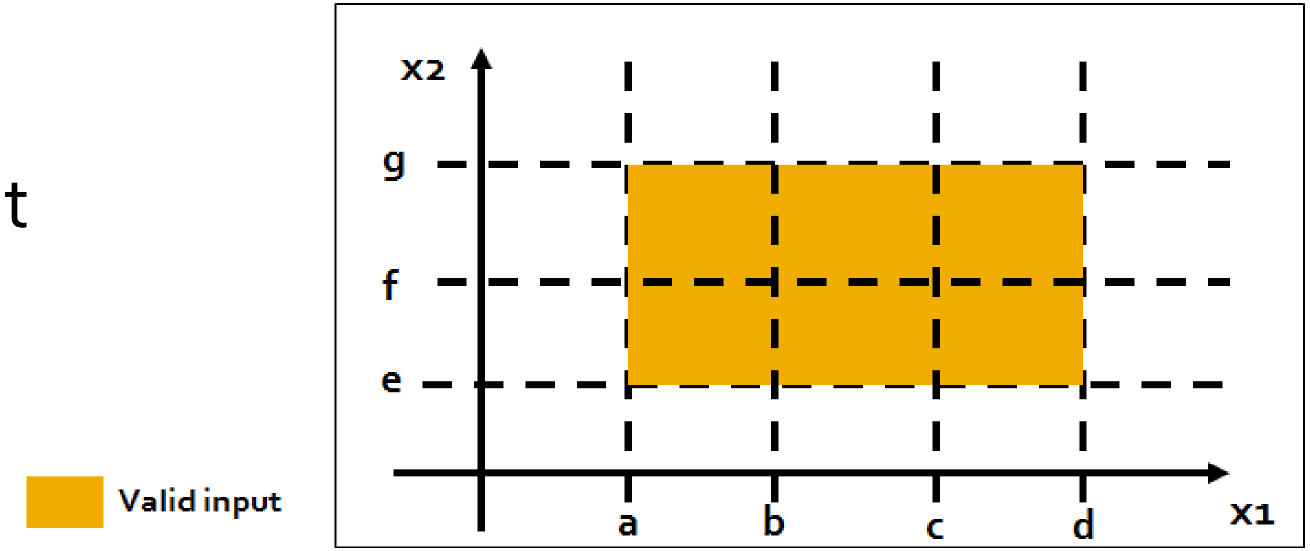
Strong General EP Testing

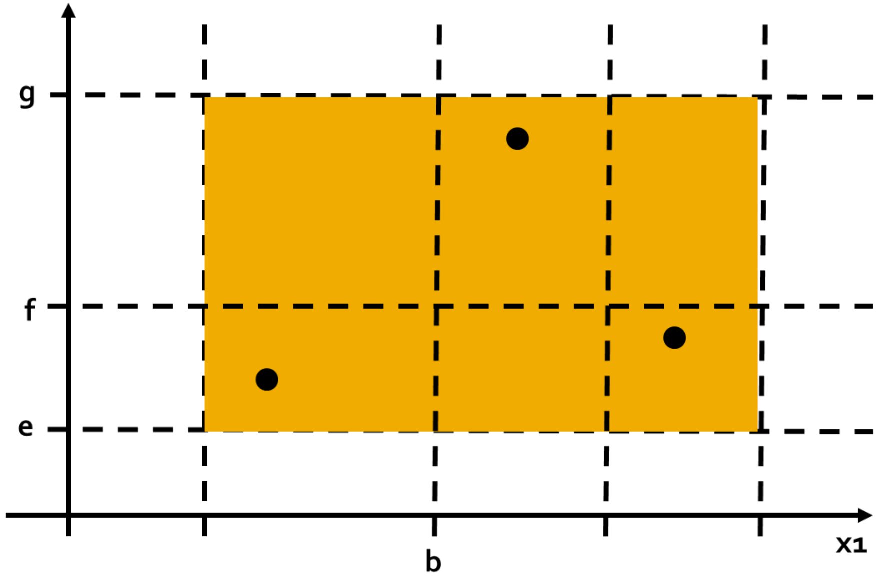
Weak Robust EP Testing

Strong Robust EP Testing

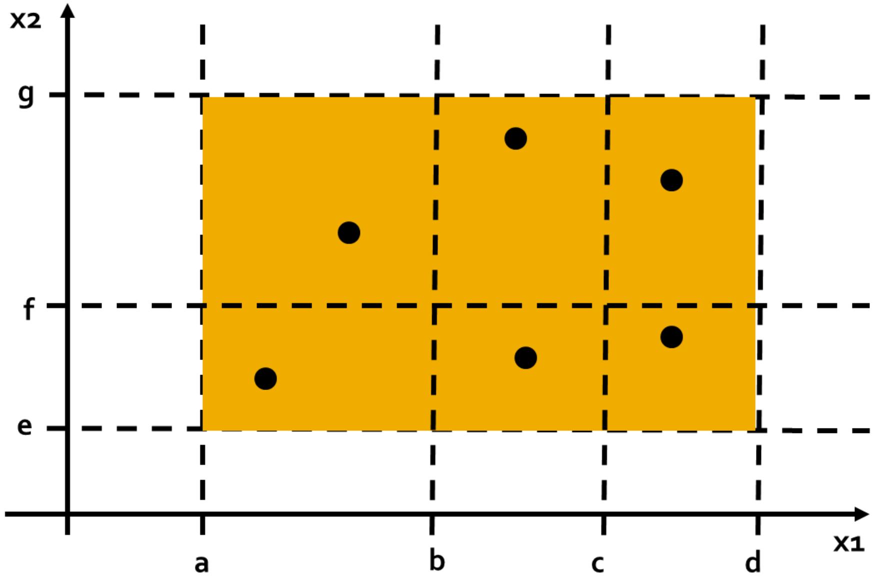
EP Testing Example

- Suppose a program F has two inputs x1 and x2 with the following constraints
  - a<=x1<=d
  - e<=x2<=g
- Suppose the following equivalent classes are identified for valid inputs
  - x1: [a, b), [b,c), [c,d]
  - x2: [e, f), [f,g]

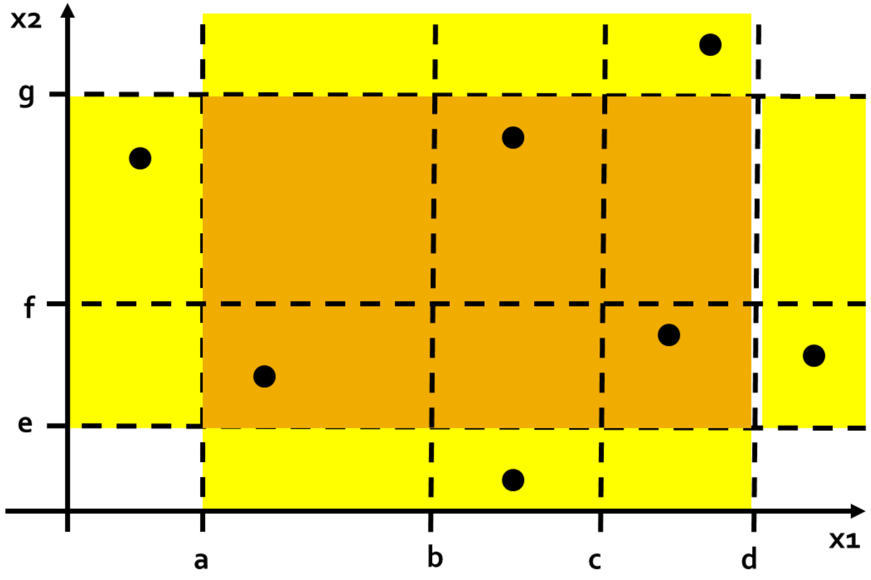




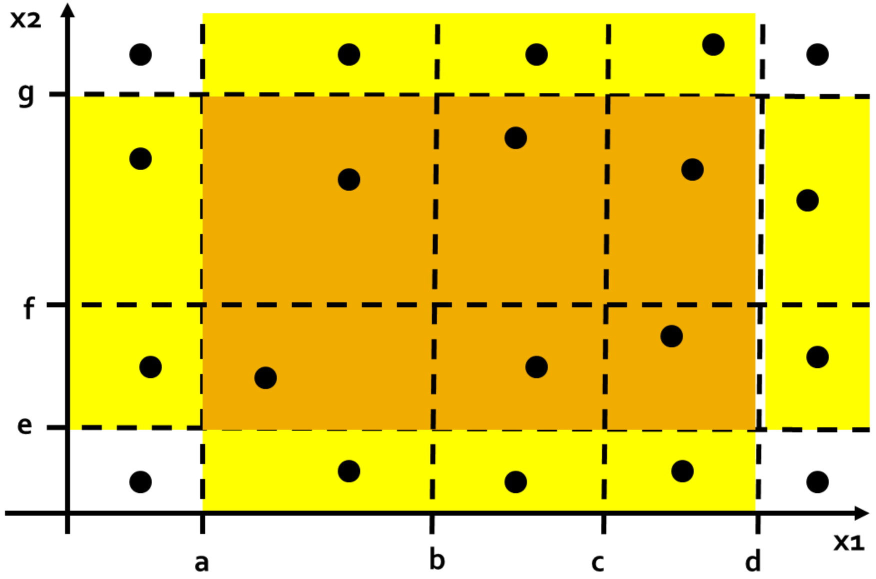
Weak General EP Testing



Strong General EP Testing



Weak Robust EP Testing



Strong Robust EP Testing

Example

- The NextDate Function

  The NextDate function accepts three parameters as inputs: the year, month and date. It returns the next date to the input date. All the input parameters should be integers which satisfy the following rules

  c1. 1 <= month <= 12

  c2. 1 <= date <= 31

  c3. 1812 <= year <= 2020

- Equivalent classes
  - Month:
    - M1={month | month<1}
    - M2={month | 1<=month<=12}
    - M3={month | month>12}
  - Date
    - D1={date | date<1}
    - D2={date | 1<=date<=31}
    - D3={date | date>31}
  - Year
    - Y1={year | year<1812}
    - Y2={year | 1812<=year<=2020}
    - Y3={year | year>2020}

Test Case Design for the NextDate

Equivalence classes			Inputs			Expected Output
Month	Date	Year	Month	Date	Year	
M1	D1	Y1	0	0	1800	Invalid
M1	D1	Y2	-2	-1	2000	Invalid
M1	D1	Y3	-1	-1	3000	Invalid
…	…	…	…	…	…	…
M2	D2	Y2	1	30	1985	1985-1-31
M2	D2	Y3	4	12	2045	Invalid
….	….	….	….	….	….	…
M3	D3	Y3	15	35	2014	Invalid

Boundary Value Analysis

- A great number of errors occur at the boundaries of the input domain rather than in the “center”
- Complementary to equivalence partitioning
  - Rather than randomly select data in the class, select the class boundary data
  - Also derive test cases from **output domain**

Example BVA of NextDate

Input			Expected Output
Month	Date	Year	
2	28	2000	2000-2-29
2	28	2007	2007-3-1
2	29	2000	2000-3-1
2	29	2007	Invalid
12	31	1999	2000-1-1
12	31	2020	Out of bound
12	31	1825	1826-1-1
1	1	1983	1983-1-2
1	1	2004	2004-1-2

Thank you!