目录

1.概述 3

1.1 测试步骤 3

1.2 测试层次 3

1.3Test 概念 5

1.4Test 目的 5

1.5 Fault, Error and Failure 5

1.6Testing technology 6

2.Black Box Testing 9

2.1Domain testing 9

2.2Boudary Value Analysis 15

2.2.1On Point and Off Point 15

3 White Box Testing 16

3.1Graph Coverage 16

3.2 Source Code Coverage 19

3.3 Basis Path Testing 24

3.4 Data Flow Criteria 24

3.5Logical Coverage 29

3.5.1Condition and Decision 29

3.5.2Clause and Predicate 31

3.5.3Fault-based Testing 35

4 专项测试.....36

4.1Configuration Testing.....36

4.2Compatibility Testing.....39

4.3Language Testing.....39

4.4Usability Testing.....40

4.5Testing the Documentation.....41

4.6Testing for Security.....42

4.7Web Site Testing.....43

1.概述

1.1 测试步骤

1 a.	Design	Design test values to satisfy engineering goals
	Criteria	Requires knowledge of discrete math, programming and testing
1 b.	Design	Design test values from domain knowledge and intuition
	Human	Requires knowledge of domain, UI, testing
2.	Automation	Embed test values into executable scripts
		Requires knowledge of scripting
3.	Execution	Run tests on the software and record the results
		Requires very little knowledge
4.	Evaluation	Evaluate results of testing, report to developers
		Requires domain knowledge

此外，还有 Test management， Test maintenance and Test documentation。

1.2 测试层次

Acceptance testing:

Is the software acceptable to the user?

检验所开发的软件是否按软件需求规格说明中确定的软件功能、性能、约束及限制等技术要求进行工作。

System testing:

Test the overall functionality of the system

指软件与该软件所属的系统对接并测试其接口的过程,目的是在真实工作环境下检验软件是否能与系统正确连接，并满足软件研制的系统目标。

Integration testing:

Test how modules interact with each other

将经过单元测试的模块逐步进行组装和测试，验证程序和概要设计说明的一致性。

Module testing:Test each class, file, module or component

Unit testing:Test each unit(method) individually

是对软件设计的最小单位：模块的测试，目的是检验每个软件单元能否正确地实现其功能满足其性能和接口要求。

Validation : The process of evaluating software at the end of software development to ensure compliance with intended usage.确认系统满足规格说明要求

Verification : The process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase.是否能够满足用户需求

1.3Test 概念

Finding inputs that cause the software to fail

（ and Debugging is the processing of finding a fault given a failure.)

1.4Test 目的

The goal of testing is to find bugs, find them as early as possible, and make sure they are fixed.

The goal of a QA person is to create and enforce standards and methods to improve the development process so that bugs are prevented.

1.5 Fault, Error and Failure

Software Fault : A static defect in the software

Software Error : An incorrect internal state that is the manifestation of some fault

Software Failure : External, incorrect behavior with respect to the requirements or other description of the expected behavior

1.6Testing technology

静态测试技术

定义：不执行程序代码而寻找程序代码中可能存在的缺陷或评估程序代码的过程

- 人工进行。
    -
        - 代码审查。
        - 代码走查。
        - 桌面检查。
- 软件工具自动进行的静态分析。

动态测试技术

定义：通过在抽样测试数据上运行程序来检验程序的动态行为和运行结果以发现缺陷

- 核心内容
    -
        - 生成测试用例
        - 运行程序
        - 验证程序的运行结果

- 辅助工作
    -
        - 文档编制
        - 数据管理
        - 操作规程
        - 工具应用

白盒测试与黑盒测试

- Black-box testing : Deriving tests from external descriptions of the software, including specifications, requirements, and design.
- White-box testing : Deriving tests from the source code internals of the software, specifically including branches, individual conditions, and statements.
- Model-based testing : Deriving tests from a model of the software (such as a UML diagram).

黑盒测试（功能测试、基于规约的测试）

- 测试人员无须了解程序的内部结构，直接根据程序输入和输出之间的关系或程序的需求规范确定测试数据，推断测试结果的正确性
- 方法
    - Domain Testing
    - 边界值分析

    - 等价类划分
    - 基于决策表的测试
    - 因果图
    - 正交实验设计
    - 状态测试

白盒测试（结构测试、基于程序的测试）

- 测试人员根据程序的内部结构特性和与程序路径相关的数据特性设计测试数据。
- 方法
    - 控制流测试
        - 语句覆盖
        - 分支覆盖
        - 条件覆盖
        - 判定-条件覆盖
        - 路径覆盖
    - 数据流测试
        - 定义覆盖
        - 引用覆盖
        - 定义-引用覆盖
    - 变异测试

灰盒测试（程序与规约相结合的测试）

- 黑盒测试与白盒测试相结合的方法
- 先用黑盒测试方法设计测试用例，然后尽可能多地用白盒测试方法完成测试

2.Black Box Testing

2.1Domain testing

Definition : partition the input domain into subdomains and then select some representative data from each subdomain for testing. These subdomians are called equivalence classes.

Applied Levels of testing

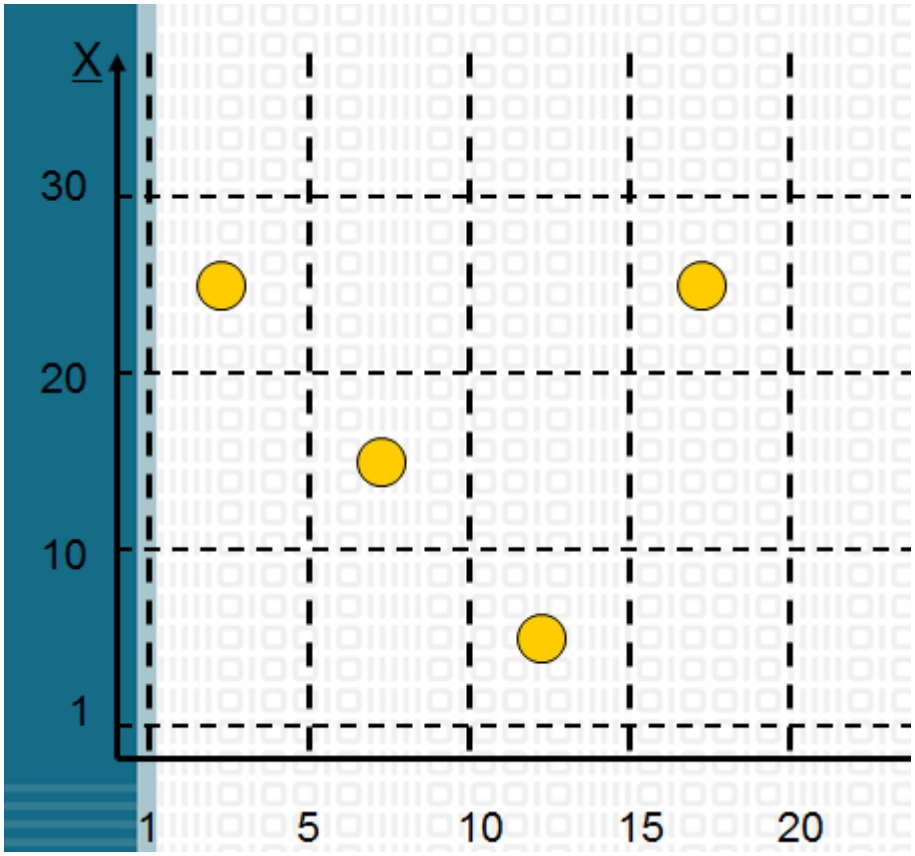
- Unit
- Integration
- System

Input parameters define the scope of the input domain

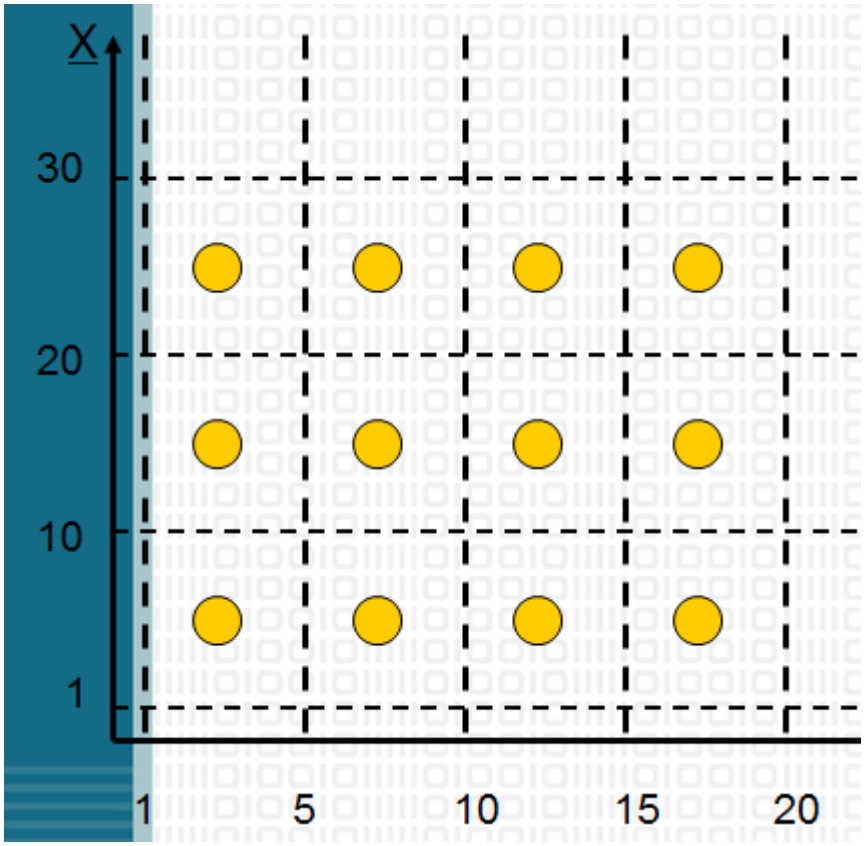
- Parameters to a method
- Data read from a file
- Global variables
- User level inputs

Type of Equivalence testing

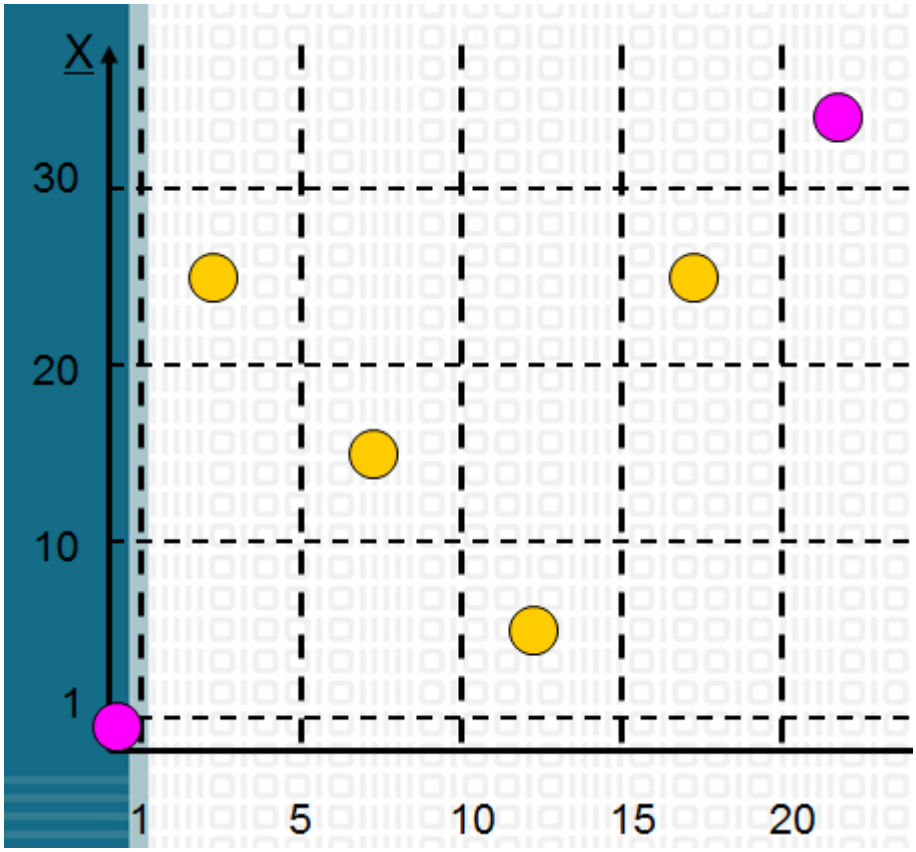
- Weak Equivalence testing(Assumes the independence of input variables):一般来说，测试用例数等于等价类最多的输入变量的等价类数目



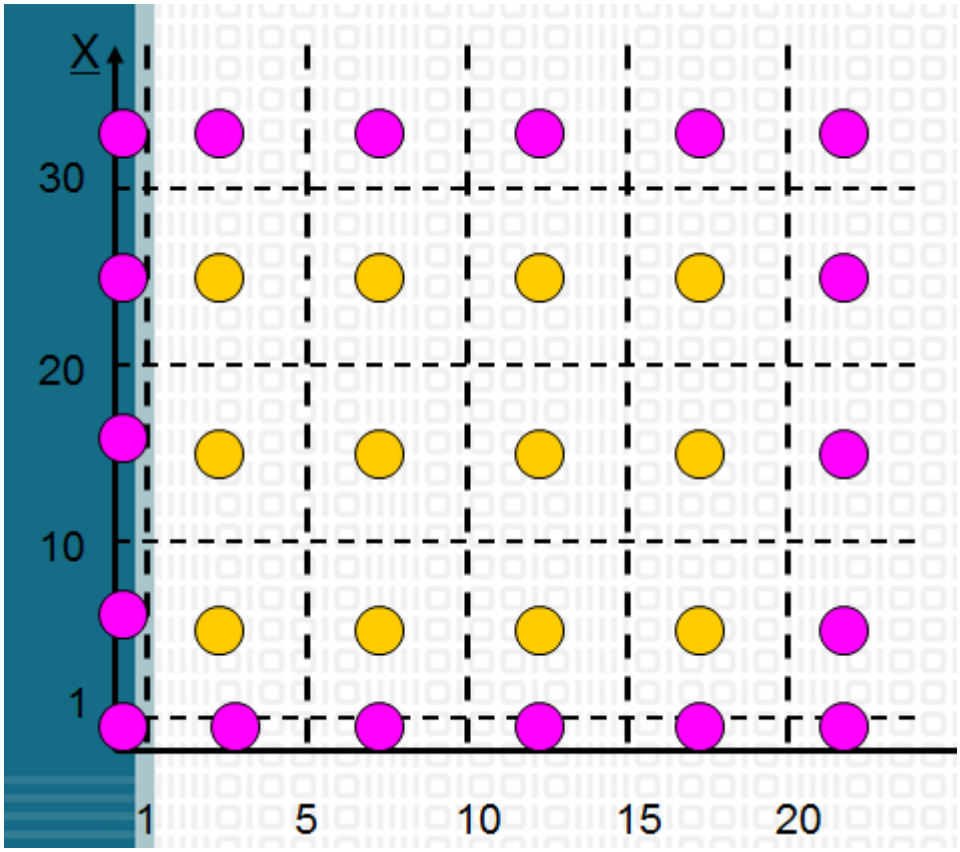
- Strong Equivalence testing (multiple fault assumption or dependence among the inputs):All the combinations of equivalence classes of variables must be included.



- Weak Robust Equivalence testing(invalid input variables are considered compared with weak equivalence testing):



- Strong Robust Equivalence testing(invalid input variables are now considered compared with strong equivalence testing)



Modeling the Input Domain

1. Identify testable functions
2. Find all the paremeters
    - Methods : Parameters and state (non-local) variables used
    - Components : Parameters to methods and state variables
    - System : All inputs, including files and databases
3. Model the input domain
    - scoped by the parameters
    - partition characterisitics into sets of blocks, whick represents a set of values

    - strategies for identifying values
        -
            - include valid, invalid and special values
            - sub-partition some blocks
            - explore boundaries of domains
            - include values that represent "normal user", such as 0, null
            - check for completeness and disjointness
4. Apply a test criterion to choose combinations of values
5. Refine combinations of blocks into test inputs
    - choose appropriate values from each block (boundary values)

Two Approaches to Input Domain Modeling

- Interface-based approach
    -
    - consider each parameter in isolation
    - rely on syntax
    - ignore relationships among parameters
- Functionality-based approach
    -
    - identify characteristics that correspond to the intended functionality
    - use relationship among parameters
    - modeling can be based on requirements, not implementataion

- hard to translate values to test cases for the same parameter may appear in multiple characteristics.

2.2Boudary Value Analysis

- choose one (n) arbitrary value(s) in each eq. class
- choose values exactly on lower/upper boundaries of eq. class
- choose values immediately below/above each boundary(if applicable)

2.2.1On Point and Off Point

On Point :

- Closed border lies on the border
- Open border lies close to the border and satisfies the inequality relation

Off Point :

- lies close to the border and on the open side
- does not satisfy the path condition associated with this border

3 White Box Testing

3.1Graph Coverage

Original Source

- Control flow graphs
- Design structure
- FSMs and statecharts
- Use cases

Each test executes one and only one test path

Syntactic reach : A subpath exists in the graph

Semantic reach : A test exists that can execute that subpath

(Semantic reach includes syntactic reach, that is semantic reach implicit syntactic reach, but not syntactic reach.)

Node and Edge Coverage

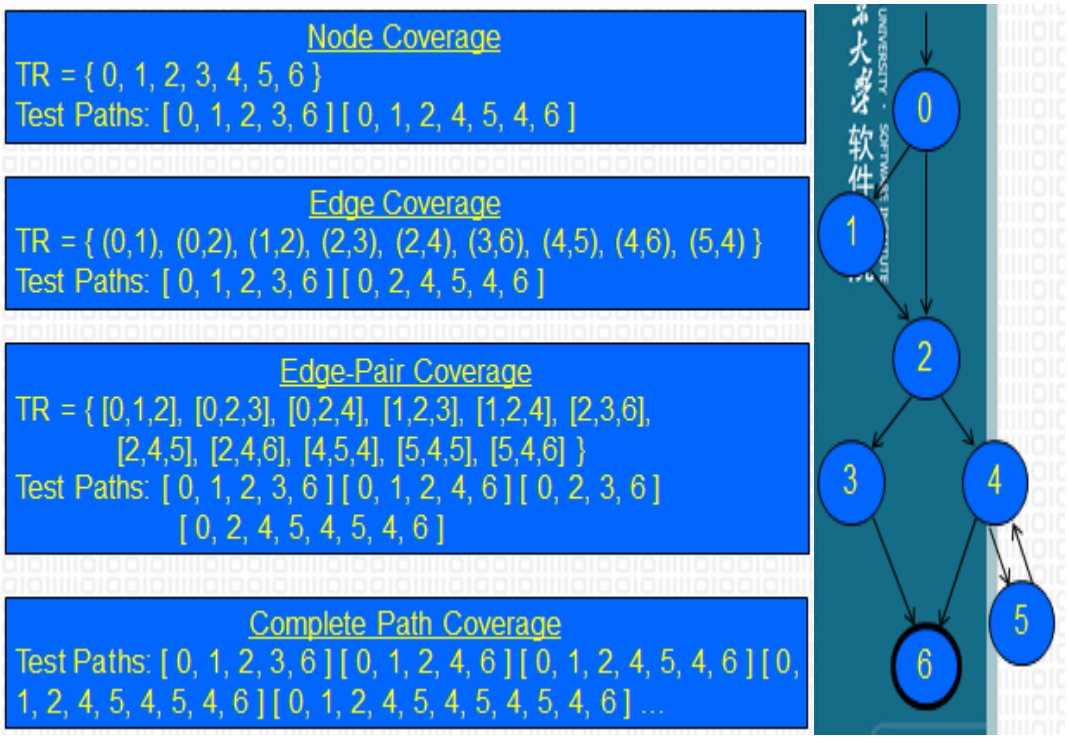
Node Coverage: TR contains each reachable node in graph G

Edge Coverage: TR contains each reachable path of length up to 1, inclusive, in graph G.

Edge coverage is slightly stronger than node coverage(只有当出现环时，两者才会有所区别)

Edge-Pair Coverage(EPC): TR contains each reachable path of length up to 2, inclusive, in G.

Complete Path Coverage(CRC): TR contains all paths in G.(If a graph contains a loop, CPC is not feasible)



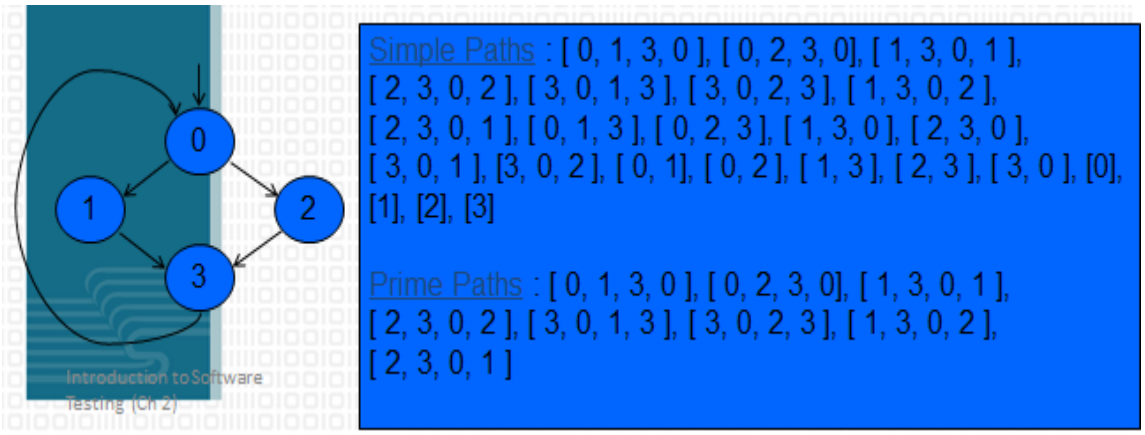
Simple Paths and Prime Paths

Simple Path: A path from node ni to nj is simple if no node appears more than once, except possibly the first and last nodes are the same.

- No internal loops

- Includes all other subpaths
- A loop is a simple path

Prime Path: A simple path that does not appear as a proper subpath of any other simple path(notes: it does not means a prime path is longest path in G and only one prime path in a G. There may be many prime paths.)



Prime Path Coverage(PPC): TR contains each prime path in G. (It subsumes node and edge coverage)

Round-Trip Path: A prime path that starts and ends at the same node.

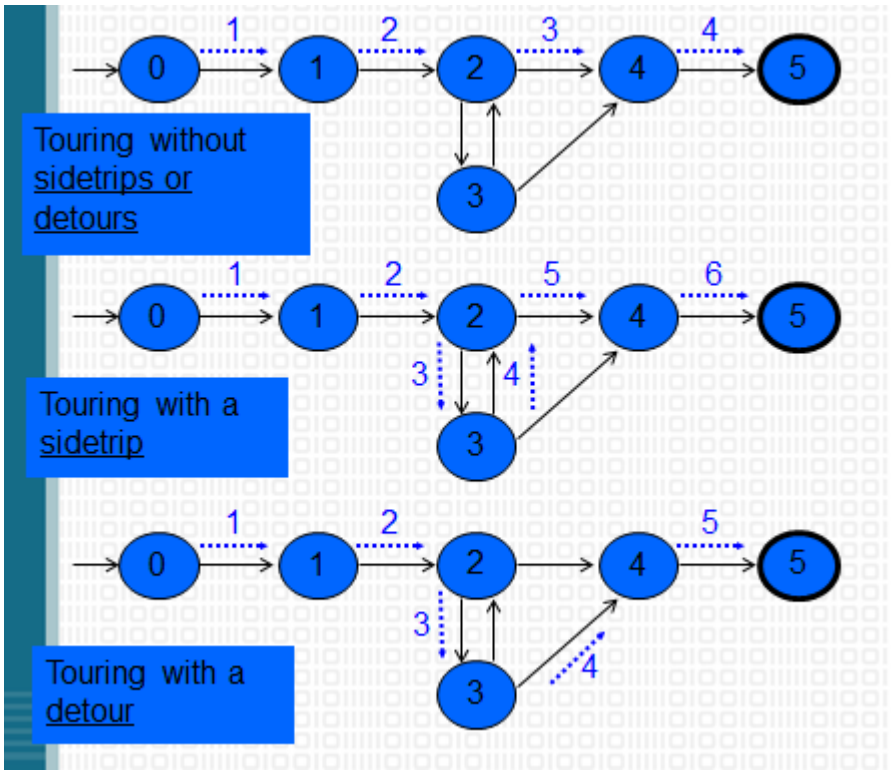
Simple Round Trip Coverage(SRTC):TR contains at least one round-trip path for each reachable node in G that begins and ends a round-trip path.

Complete Round Trip Coverage(CRTC): TR contains all round-trip paths for each reachable node in G.

(These criteria omit nodes and edges that are not in round trips)

Touring, Sidetrips and Detours(All of these refer to test paths, but not prime paths)

- Tour: A test path p tours subpath q if q is a subpath of p
- Tour With Sidetrips : A test path p tours subpath q with sidetrips iff every edge in q is also in p in the same order(也就是说兜了一圈，回到同一个节点)
- Tour With Detours : A test path p tours subpath q with detours iff every node in q is also in p in the same order（也就是说兜了一圈，回到紧接的下一个节点）

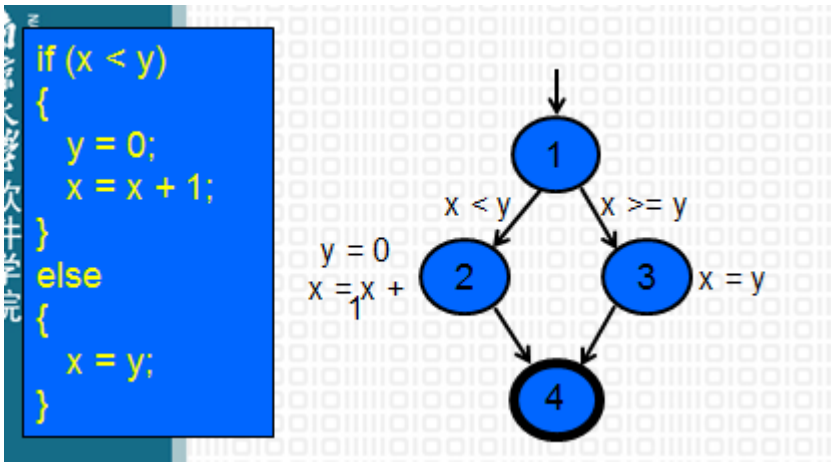


3.2 Source Code Coverage

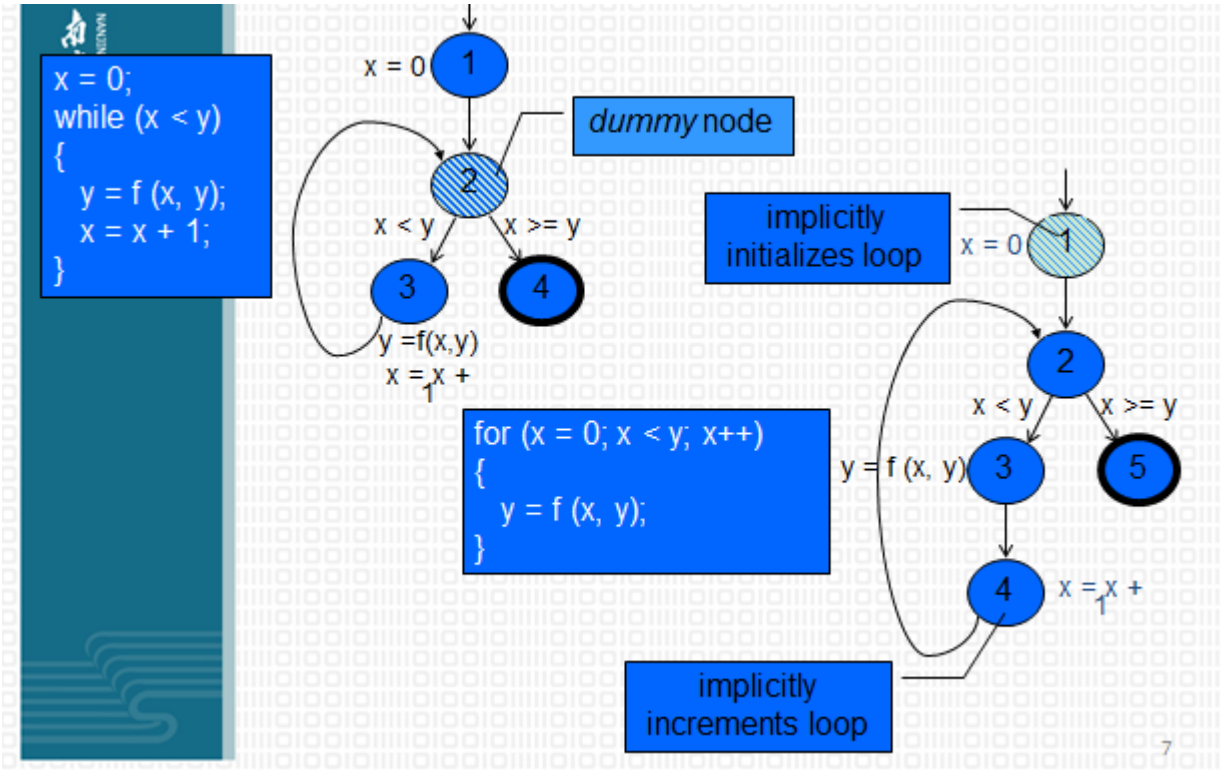
Control Flow Graphs: models all executions of a method by describing control structures

- Nodes : Statements or sequences of statements (basic blocks)
- Edges : Transfers of control
- Basic Block : A sequence of statements such that if the first statement is executed, all statements will be (no branches)

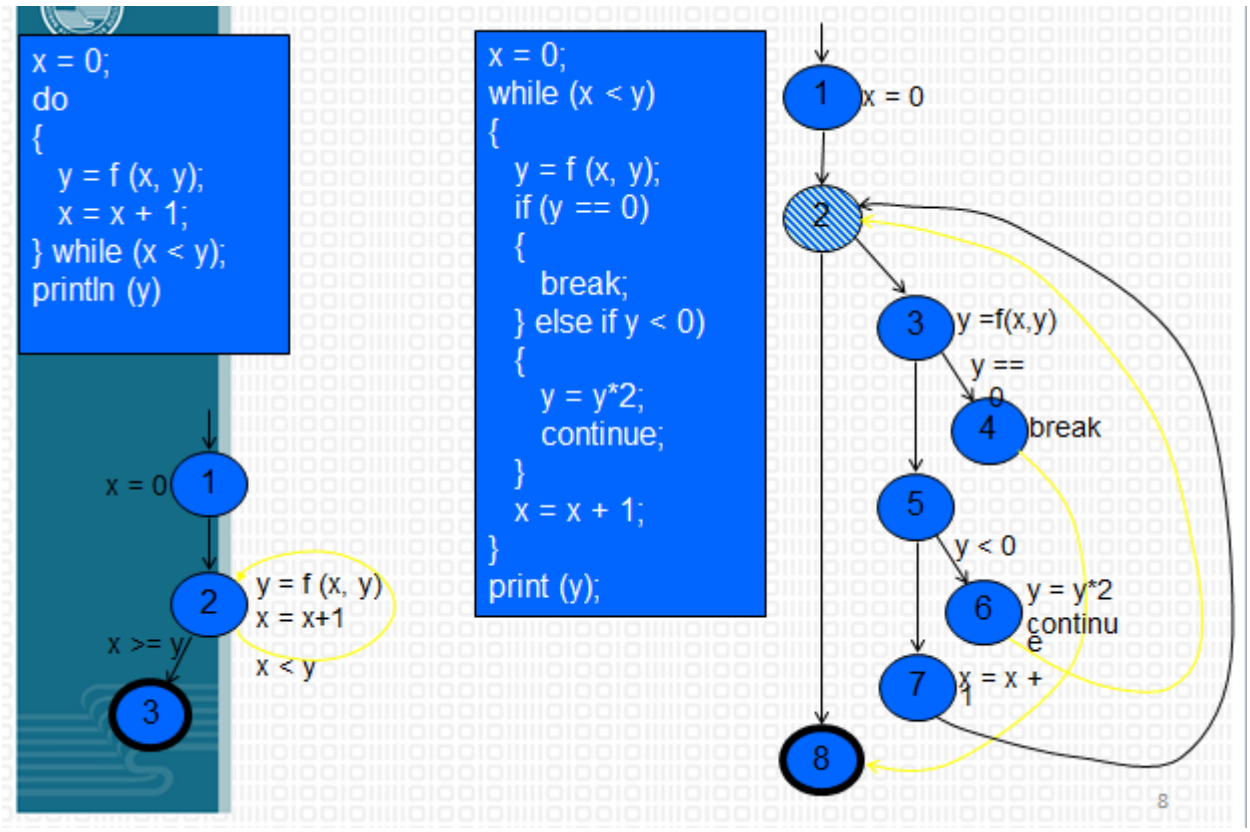
1. If Statement



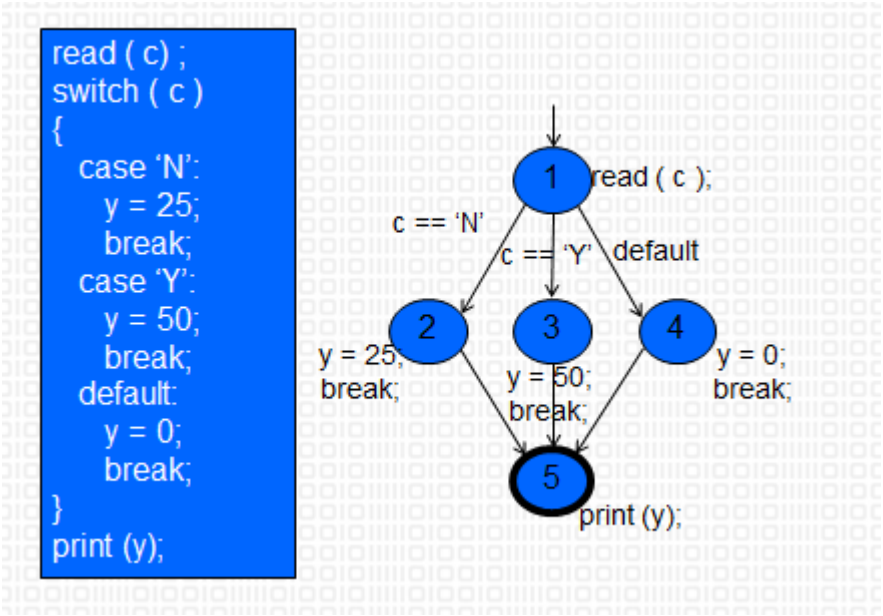
1. Loops (need "extra" nodes to be added that do not represent statements or basic blocks)
    - while and for loops

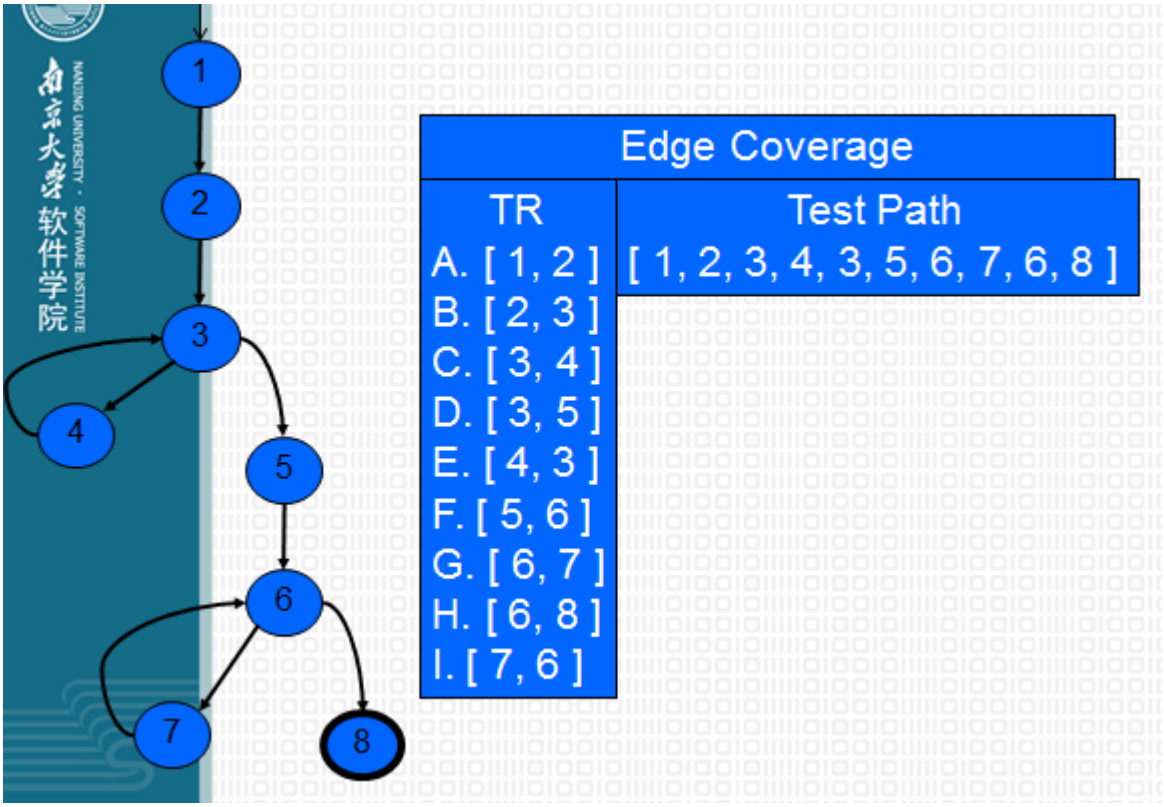


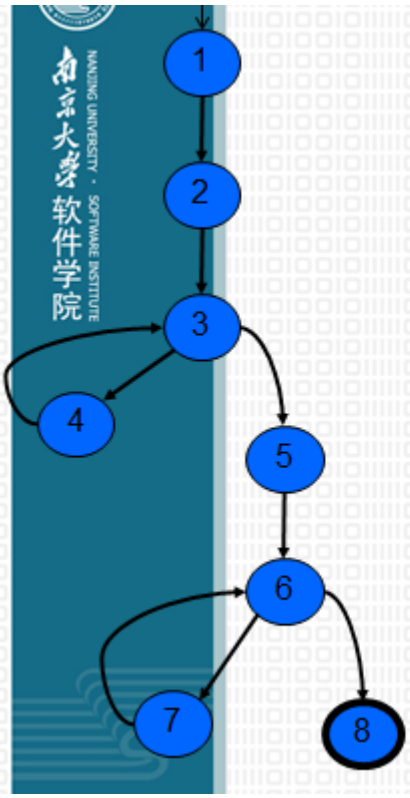
- do loop, break and continue



- case(switch) structure

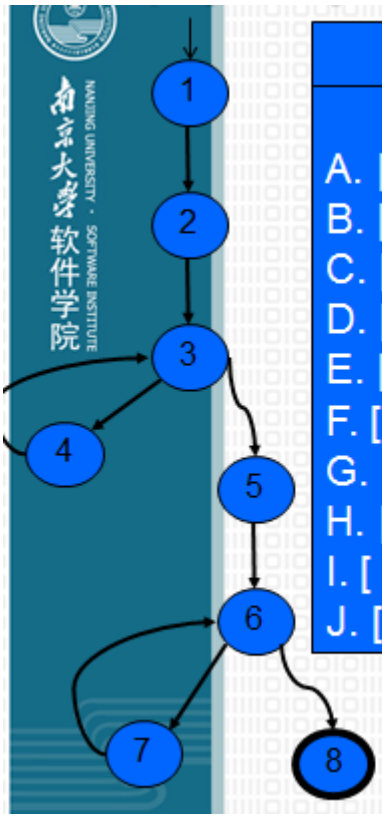






Edge-Pair Coverage

TR	Test Paths
A. [1, 2, 3] B. [2, 3, 4] C. [2, 3, 5] D. [3, 4, 3] E. [3, 5, 6] F. [4, 3, 5] G. [5, 6, 7] H. [5, 6, 8] I. [6, 7, 6] J. [7, 6, 8] K. [4, 3, 4] L. [7, 6, 7]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 5, 6, 8] iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]



Prime Path Coverage

TR	Test Paths
A. [3, 4, 3] B. [4, 3, 4] C. [7, 6, 7] D. [7, 6, 8] E. [6, 7, 6] F. [1, 2, 3, 4] G. [4, 3, 5, 6, 7] H. [4, 3, 5, 6, 8] I. [1, 2, 3, 5, 6, 7] J. [1, 2, 3, 5, 6, 8]	i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8] ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8] iii. [1, 2, 3, 4, 3, 5, 6, 8] iv. [1, 2, 3, 5, 6, 7, 6, 8] v. [1, 2, 3, 5, 6, 8]

3.3 Basis Path Testing

- a bybrid between path testing and branch testing

Basis Path: a unique path through the software where no iterations are allowed - all possible paths through the system are linear combinations of them.

Process of Basis Path Testing

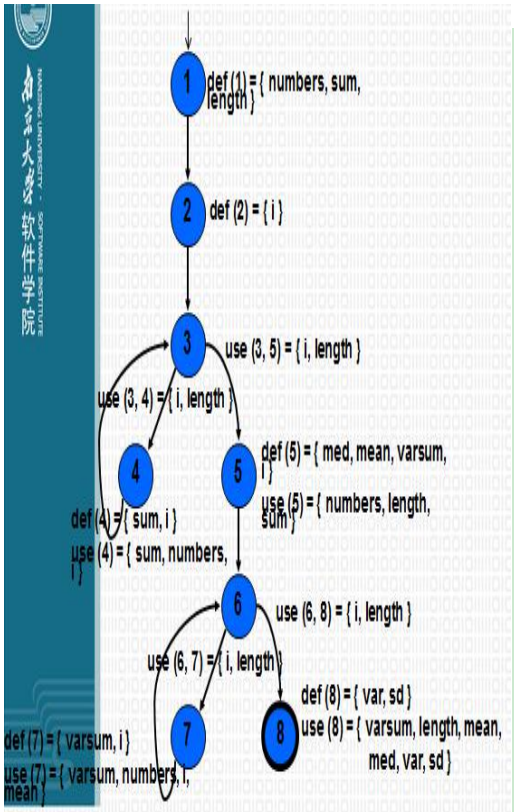
1. Draw a control flow graph
2. Calculate Cyclomatic complexity
    - e - n + 2
    - 图中封闭区域的数目
    - 控制节点的数目：有两个分支的控制节点，就+1，有三个分支的控制节点则+2；依次类推；
3. Choose a “basis set” of paths（数目为圈复杂度）
4. Generate test cases to exercise each path

3.4 Data Flow Criteria

Goal: Try to ensure that values are computed and used correctly.

- Definition (def) : A location where a value for a variable is stored into memory(定义的地点)

- Use : A location where a variable’ s value is accessed（使用的地点）
- def (n) or def (e) : The set of variables that are defined by node n or edge e
- use (n) or use (e) : The set of variables that are used by node n or edge e




```
public static void computeStats (int [ ] numbers)
{
   int length = numbers.length;
   double med, var, sd, mean, sum, varsum;

   sum = 0.0;
   for (int i = 0; i < length; i++)
   {
      sum += numbers [ i ];
   }
   med   = numbers [ length / 2 ];
   mean = sum / (double) length;

   varsum = 0.0;
   for (int i = 0; i < length; i++)
   {
      varsum = varsum  + ((numbers [ i ] - mean) * (numbers [ i ] - mean));
   }
   var = varsum / ( length - 1 );
   sd  = Math.sqrt ( var );

   System.out.println ("length:              " + length);
   System.out.println ("mean:                " + mean);
   System.out.println ("median:              " + med);
   System.out.println ("variance:            " + var);
   System.out.println ("standard deviation: " + sd);
}
```

DU Pairs and DU Paths

- DU pair : A pair of locations (li, lj) such that a variable v is defined at li and used at lj
- Def-clear : A path from li to lj is def-clear with respect to variable v if v is not given another value on any of the nodes or edges in the path
- Reach : If there is a def-clear path from li to lj with respect to v, the def of v at li reaches the use at lj

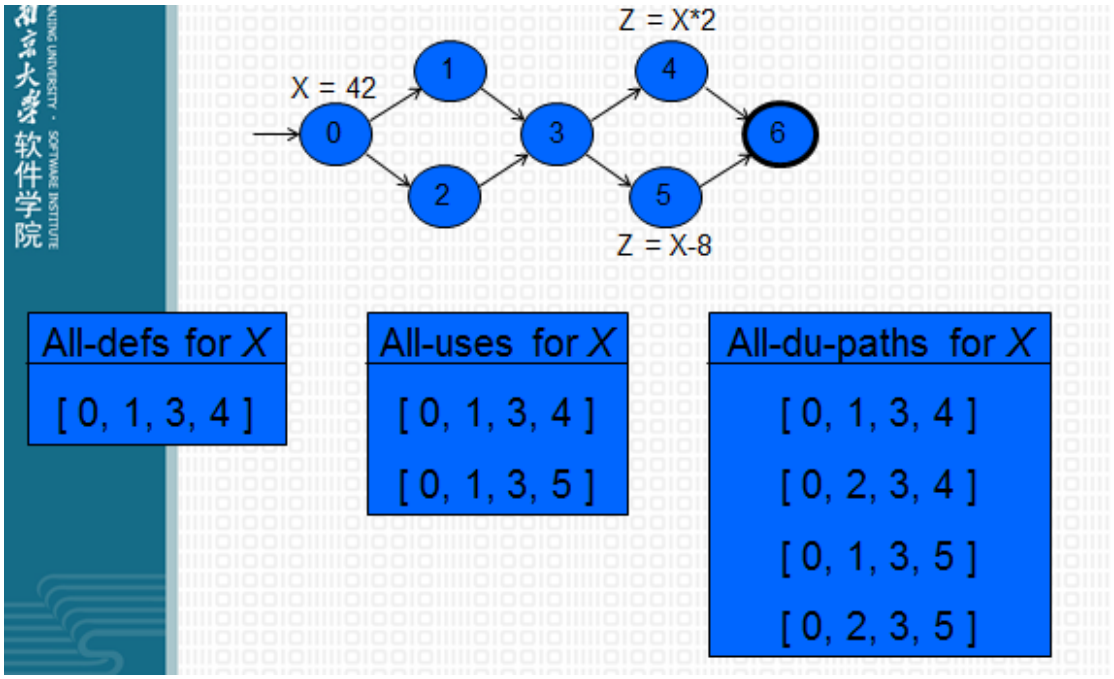
- du-path : A simple subpath that is def-clear with respect to v from a def of v to a use of v
- du (ni, nj, v) – the set of du-paths from ni to nj
- du (ni, v) – the set of du-paths that start at ni

Data Flow Test Criteria

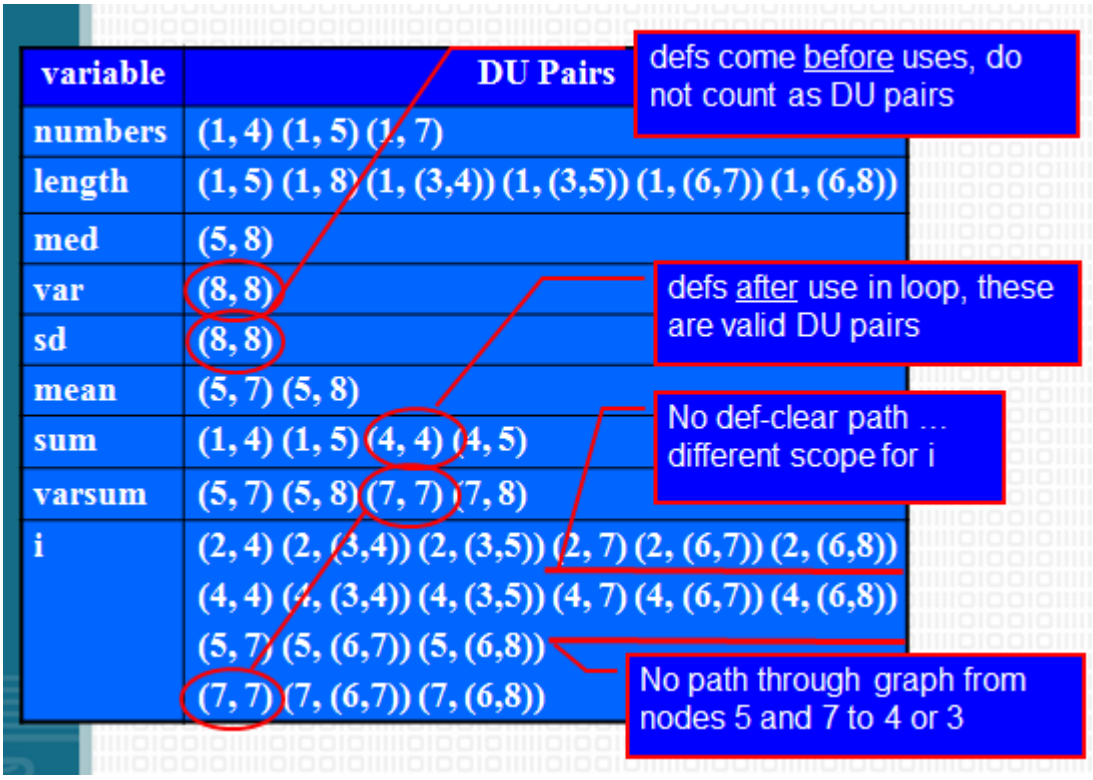
- First make sure every def reaches a use. That is, All-defs coverage (ADC): For each set of du-paths S = du (n, v), TR contains at least one path d in S.
- Then make sure that every def reaches all possible uses. That is, All-uses coverage (AUC) : For each set of du-paths to uses S = du (ni, nj, v), TR contains at least one path d in S.
- Finally, cover all the paths between defs and uses. That is, All-du-paths coverage (ADUPC) : For each set S = du (ni, nj, v), TR contains every path d in S.

Data Flow Testing Example

（下面的这幅图感觉有点怪怪，前面的 All-defs for X 和 All-uses for X 不知道为什么是这样）



1. 为每个变量找出 DU Paires，并去掉无效的 DU Paires

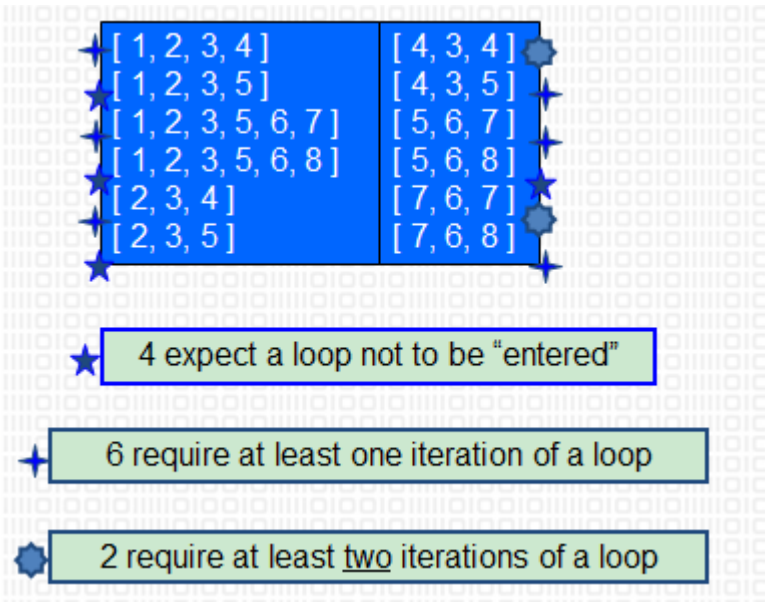


2. 根据 Du Pairs 找到 DU Paths

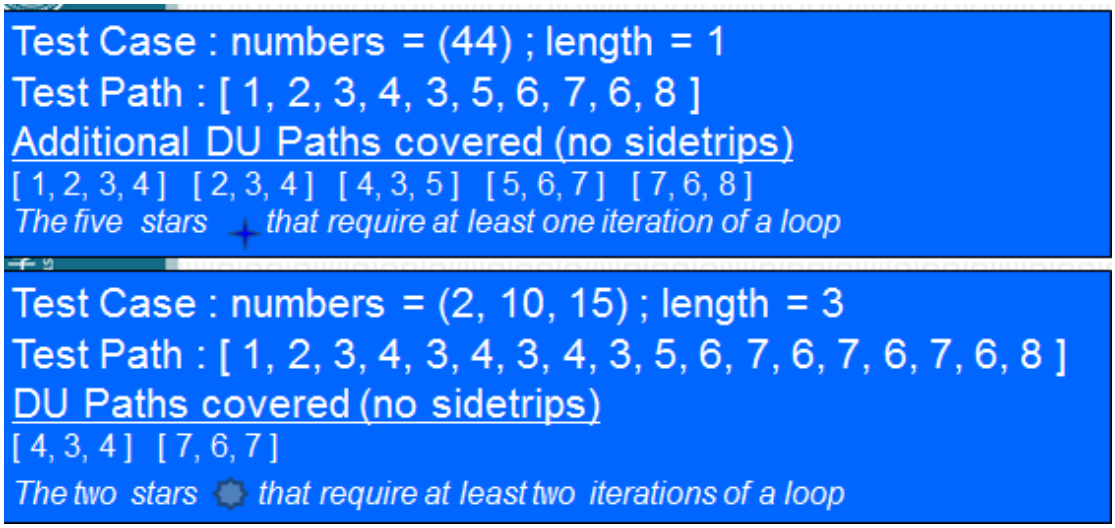
variable	DU Pairs	DU Paths
numbers	(1, 4) (1, 5) (1, 7)	[1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7]
length	(1, 5) (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))	[1, 2, 3, 5] [1, 2, 3, 5, 6, 8] [1, 2, 3, 4] [1, 2, 3, 5] [1, 2, 3, 5, 6, 7] [1, 2, 3, 5, 6, 8]
med	(5, 8)	[5, 6, 8]
var	(8, 8)	*No path needed*
sd	(8, 8)	*No path needed*
sum	(1, 4) (1, 5) (4, 4) (4, 5)	[1, 2, 3, 4] [1, 2, 3, 5] [4, 3, 4] [4, 3, 5]

variable	DU Pairs	DU Paths
mean	(5, 7) (5, 8)	[5, 6, 7] [5, 6, 8]
varsum	(5, 7) (5, 8) (7, 7) (7, 8)	[5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 8]
i	(2, 4) (2, (3,4)) (2, (3,5)) (4, 4) (4, (3,4)) (4, (3,5)) (5, 7) (5, (6,7)) (5, (6,8)) (7, 7) (7, (6,7)) (7, (6,8))	[2, 3, 4] [2, 3, 4] [2, 3, 5] [4, 3, 4] [4, 3, 4] [4, 3, 5] [5, 6, 7] [5, 6, 7] [5, 6, 8] [7, 6, 7] [7, 6, 7] [7, 6, 8]

3. 去掉重复的 DU Paths



4. 最后设计测试用例，覆盖所有的 DU Paths



3.5Logical Coverage

（建议参考 http://www.bullseye.com/coverage.htm ）

3.5.1Condition and Decision

Decision Coverage : T satisfies Decision Coverage (DC) of P, denoted by T∈DC(P), if each decision is evaluated to true and false at least once by T.

```
// {(6, 1), (1,1)}--DC
int foo(int x, int y) {
        int z = y*2; \\ z=y;
        if ((x>5) && (y>0)) {
                z = x; }
        return x*z;
}
```

Condition Coverage : T satisfies Decision Coverage (CC) of P, denoted by T∈CC(P), if each condition is evaluated to true and false at least once by T. (A condition is an

operand of a logical operator that does not contain logical operators.)

```
// {(6, 0), (0,1)}--CC
int foo(int x, int y) {
        int z = y;
        if ((x>5) && (y>0)) {
                z = x; }
        return x*z;
}
```

Condition / Decision Coverage : T satisfies Condition/ Decision Coverage of P if T satisfies both CC and DC.

```
// {(6, 1), (0,0)}--CC
int foo(int x, int y) {
        int z = y;
        if ((x>5) && (y>0)) {
                z = x; }
        return x*z;
}
```

Modified C / DC : Modified Condition/Decision Coverage (MC/DC) extends condition/decision coverage with requirements that each condition should affect the decision outcome independently.

Subsumption Relation

- DC ≥ SC
- CC not ≥ SC

- DC not ≥ CC , CC not ≥ DC
- CDC(*P*)=CC(*P*) ∩ DC(*P*)
- CDC ≥ CC
- CDC ≥ DC
- MCDC(*P*)=ACC(*P*) ∩ DC(*P*)

(DC : Decision Coverage, CC : Condition Coverage, SC : Source code Coverage, CDC : Condition / Decision Coverage, MCDC : Modified C/D Coverage)

3.5.2Clause and Predicate

Predicate : an expression that evaluates to a boolean value

Clause : a predicate with no logical operators

Predicate Coverage (PC) : For each p in P, TR contains two requirements: p evaluates to true, and p evaluates to false. (Predicate coverage views paths as possible combinations of logical conditions and a path is a unique sequence of branches from the function entry to the exit.)

Clause Coverage (CC) : For each c in C, TR contains two requirements: c evaluates to true, and c evaluates to false.

Subsumption Relation

- Predicate coverage includes decision coverage.
- Predicate coverage includes decision coverage and multiple condition

coverage that reports whether every possible combination of conditions occurs.

Combinatorial Coverage (CoC) : For each p in P, TR has test requirements for the clauses in Cp to evaluate to each possible combination of truth values.

Determination : A clause ci in predicate p, called the major clause, determines p if and only if the values of the remaining minor clauses cj are such that changing ci changes the value of p. This is considered to make the clause active.

Active Clause Coverage (ACC) : For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci determines p. TR has two requirements for each ci : ci evaluates to true and ci evaluates to false.

General Active Clause Coverage (GACC) : For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci determines p. TR has two requirements for each ci : ci evaluates to true and ci evaluates to false. The values chosen for the minor clauses cj do not need to be the same when ci is true as when ci is false, that is, cj(ci = true) = cj(ci = false) for all cj OR cj(ci = true) != cj(ci = false) for all cj. (It is possible to satisfy GACC without satisfying predicate coverage)

Restricted Active Clause Coverage (RACC) : For each p in P and each major clause

ci in Cp, choose minor clauses cj, j != i, so that ci determines p. TR has two requirements for each ci: ci evaluates to true and ci evaluates to false. The values chosen for the minor clauses cj must be the same when ci is true as when ci is false, that is, it is required that cj(ci = true) = cj(ci = false) for all cj.

Correlated Active Clause Coverage (CACC) : For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci determines p. TR has two requirements for each ci: ci evaluates to true and ci evaluates to false. The values chosen for the minor clauses cj must cause p to be true for one value of the major clause ci and false for the other, that is, it is required that p(ci = true) != p(ci = false).

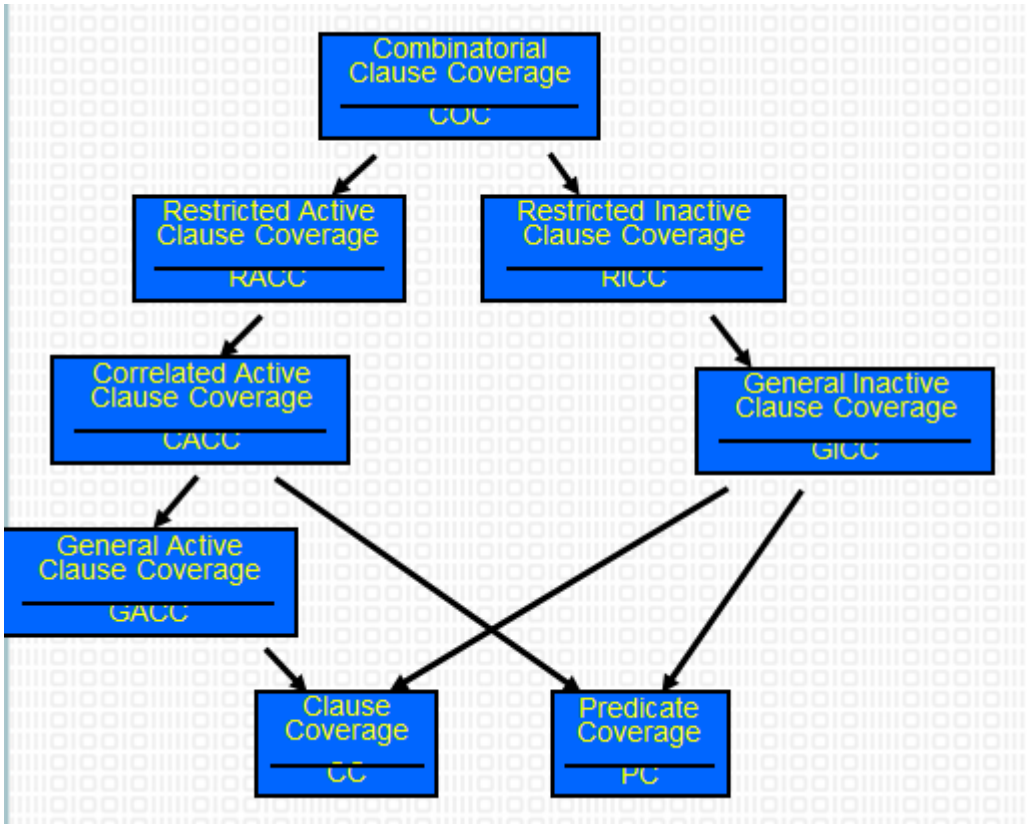
Inactive Clause Coverage (ICC) : For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci does not determine p. TR has four requirements for each ci: (1) ci evaluates to true with p true, (2) ci evaluates to false with p true, (3) ci evaluates to true with p false, and (4) ci evaluates to false with p false.

General Inactive Clause Coverage (GICC) : For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci does not determine p. The values chosen for the minor clauses cj do not need to be the same when ci is true as when ci is false, that is, cj(ci = true) = cj(ci = false) for all cj OR cj(ci = true) != cj(ci = false) for all cj.

Restricted Inactive Clause Coverage (RICC)：For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci does not determine p. The values chosen for the minor clauses cj must be the same when ci is true as when ci is false, that is, it is required that cj(ci = true) = cj(ci = false) for all cj.

(注意：这里没有 CICC，即 Correlative ICC，因为 major clause does not determine p)

Subsumption relation

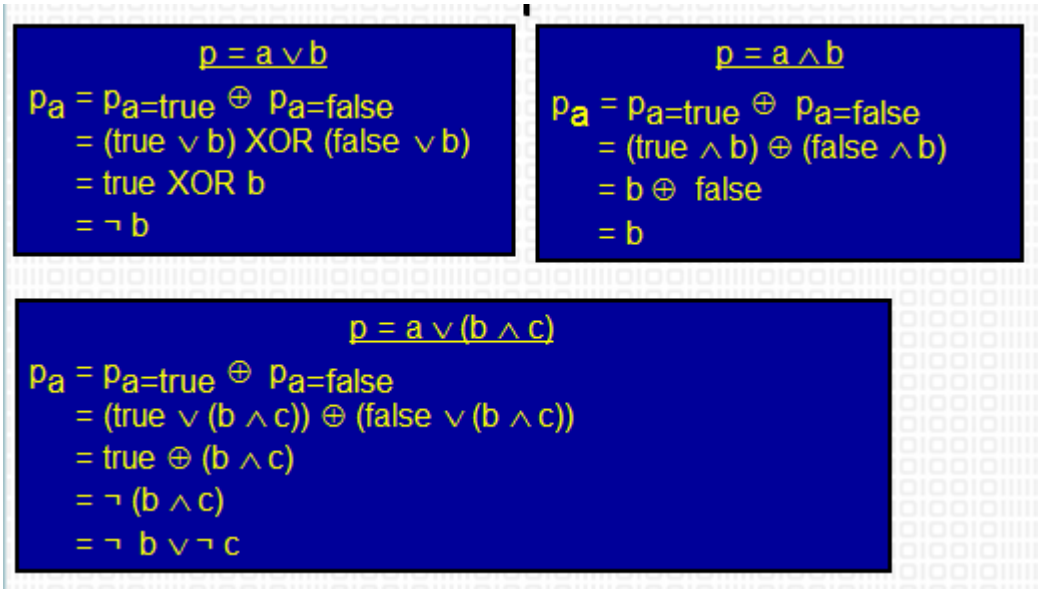


Method to find values for the minor clauses

- Connect Pc = true and Pc = false with exclusive OR when clause c is the major clause.

$$p_c = p_{c=true} \oplus p_{c=false}$$

- After solving, Pc describes exactly the values needed for c to determine p



(in case one, b need to be false; in case two, b need to be true; in case three, either b or c can be false.)

Summary

Predicates are often very simple—in practice, most have less than 3 clauses , so

- With only clause, PC is enough
- With 2 or 3 clauses, CoC is practical
- Advantages of ACC and ICC criteria significant for large predicates
- CoC is impractical for predicates with many clauses

3.5.3Fault-based Testing

Hypothesize certain types of faults that may be committed by programmers and then design test cases targeted at these faults.

Fault Classes :Operator Faults(这个感觉有所了解就好了)

- Operator Reference Fault (ORF)
- Expression Negation Fault (ENF)
- Variable Negation Fault (V NF)
- Associative Shift Fault (ASF)
- Missing Variable Fault (MVF)
- Variable Reference Fault (VRF)
- Clause Conjunction Fault (CCF)
- Clause Disjunction Fault (CDF)
- Stuck-At-0 Fault (SA0)
- Stuck-At-1 Fault (SA1)

（MC/DC=SA0 and SA1）

4 专项测试

4.1Configuration Testing

Performance Testing Type

- Load testing

    - conducted to understand the behavior of the system under a specific expected load.
- Stress testing
    - used to understand the upper limits of capacity within the system
    - determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum
- Endurance testing (soak testing)
    - determine if the system can sustain the continuous expected load
- Spike testing
    - The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load. (Compared with stress testing, this testing focuses more on the changes in load.)
    - done by suddenly increasing the number of, or load generated by, users by a very large amount and observing the behavior of the system
- Configuration testing
    - determine the effects of configuration changes to the system's components on the system's performance and behavior.
    - A common example would be experimenting with different methods of load-balancing
- Isolation testing
    - describe repeating a test execution that resulted in a system problem

- Often used to isolate and confirm the fault domain

Definition: the process of checking the operation of the software you are testing with all these various types of hardware.

Method to identify Configuration bugs

The sure way to tell if a bug is a configuration problem and not just an ordinary bug is to perform the exact same operation that caused the problem, step by step, on another computer with a completely different configuration. If the bug does not occur, it is very likely a configuration problem. If the bug happens on more than one configuration, it is probably just a regular bug

Approaching the Task

1. Decide the types of hardware you will need.
2. Decide what hardware brands, models, and device drivers are available.
3. Decide which hardware features, modes, and options are possible.
4. Pare down the identified hardware configurations to a manageable set
5. Identify your software unique features that work with the hardware configurations.
6. Design the test cases to run on each configuration.
7. Execute the tests on each configuration.
8. Rerun the tests until the results satisfy your team.

4.2Compatibility Testing

Definition: Checking that your software interacts with and shares information correctly with other software.

Type :

- System compatibility
- Backward and Forward Compatibility
- Data compatibility
  - supports and adheres to published standards and allows users to easily transfer data to and from other software

4.3Language Testing

I18N(Internationalization) : the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Internationalization is the task of software developers.

L10N(localization) : the process of translating and adapting software to a particular language and culture for an already internationalized software. Localization need implementation translation of text, the change of UI, sounds and images, product testing.

G11N(Globalization) : a general term which is used to cover two different processes, internationalization and localization.

An internationalized program has the following *characteristics*:

- With the addition of localized data, the same executable can run worldwide
- Textual elements, such as status messages and the GUI component labels, are not hardcoded in the program. Instead they are stored outside the source code and retrieved dynamically
- Support for new languages does not require recompilation
- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language (data format)
- It can be localized quickly

I18N Testing: determine how well internationalization has been done.

L10N Testing contains

- linguistic Testing
- Translation verification Testing
- Cosmetic (UI) Testing
- In-country Testing
- Functionality Testing

4.4Usability Testing

Usability – Easy to discover, Easy to learn, Easy to use, Availability

Usability is how appropriate , functional, and effective that interaction is.

The main content of usability testing is UI testing.

What makes a Good UI

- Follows Standards or Guidelines
- Intuitive
- Correct
- Consistent
- Flexible
- Comfortable
- Useful
- Simple

Accessibility Testing – Testing for the disabled

- Visual Impairments
- Hearing Impairments
- Motion Impairments
- Cognitive and language

4.5Testing the Documentation

System Documentation - Detailed information about a system’ s design specs, its

internal workings, and its functionality

User Documentation - Written or visual information about an application system, how it works, and how to use it

Documentation Types

- Considering the Audience
- User’ s Manuals
  - System Summary
  - Manual functional description
- Operator’ s Manuals
- General System Guide
- Tutorials and Automated System Overviews
- Failure Message Reference Guide
- Other system Documentation

4.6Testing for Security

5 Principles Needing to Test

- Authentication: Identity - Validity
- Integrity: protection from tampering/spoofing
- Privacy: protection from eavesdropping
- Non-Repudiation: accountability

- Availability: RAID, clusters, cold standbys

4.7Web Site Testing

Web Site Testing including:

- Web page content
- Functionality
- Usability
- Security
- Performance

Testing method used for Web Site Testing:

- Black-box Testing(静态页面)
  - Text, ATL text
  - Hyperlinks
  - Graphics
  - Forms
  - Objects and other Simple Miscellaneous Functionality(此 Object 非彼 Object)
- Gray-box Testing – Test the software as a black-box, but you supplement the work by taking a peek(not a full look, as in white-box testing) at what makes the software work

- White-Box Testing(动态生成页面、服务器、安全方面)
  - Dynamic Content
  - Database Driven Web pages
  - Programmatically Created Web Pages
  - Server performance and Loading
  - Security
- Configuration and Compatibility Testing
  - To ensure that a site performs as intended across multiple operating system and browser configurations
  - Forward/Backward Compatibility
  - OS、Browser、Connection Speed、Monitor Resolution
- Network Testing
  - Ensure a product is networkable
  - Make sure it can be run on several network OS environments
  - Stress tests is also needed.
- Usability Testing

Automation Tools

- Load Test
  - **Webload** (PPT 里详细介绍了，留意点)
    - Indicates where problems are occurring
    - Indicates how many concurrent users your site can handle before

response times become unacceptable

- WAS（Web Application Stress）
- JMeter

	Advantages	Disadvantages
WebLoad	1. easy to learn 2. can generate proper report 3. can record the test script automatically	1. License is not free 2. could not capture program error effectively
Robot VU	1. can generate sophisticated report 2. can use C or Java to write the test script	1. license is not free 2. very hard to analysis the test result 3. the longer learning curve
JMeter	1. License free/open source 2. Can use third-party software to record the test script. 3. can check the every request and response pair more in detail	1. Could not generate sophisticated report 2. some bugs on itself

- Uptime Monitoring Tool – check your site continue by requesting a page from your server every 15 minutes.