Chapter 8 - Software Testing

MI Qing (Lecturer)

Telephone: 15210503242

Email: miqing@bjut.edu.cn

Office: 410, Information Building

Topics Covered

✧ Overview of Software Testing

✧ Static Testing VS Dynamic testing

✧ Black Box Testing VS White Box Testing

✧ Manual Testing VS Automated Testing

✧ Development Testing VS User Testing

✧ Unit Testing VS Integration Testing VS System Testing

Overview of Software Testing

Do I need to do testing?

✧ Everyone who develops software does testing.

✧ The only question is whether testing is conducted haphazardly by random trial-and-error or systematically with a plan.

Skills Required to Become a Software Tester

✧ Non-Technical Skills

- Analytical skills
- Communication skills
- Time management skills
- GREAT attitude

✧ Technical Skills

- Knowledge and hands-on experience of any defect tracking tool
- Knowledge and hands-on experience of automation tool
- Basic knowledge of Database
- Basic knowledge of Linux commands

Terms

错误（Error）： 在软件生存期内的不希望或不可接受的人为错误，其结果是导致软件缺陷的产生。

缺陷（Defect）： 软件缺陷是存在于软件产品之中的那些不希望或不可接受的偏差，其结果是软件运行于某一特定条件时出现故障。

故障（Fault）： 软件运行过程中出现的一种不希望或不可接受的内部状态，若无适当措施（容错）加以及时处理，便产生软件失效。

失效（Failure）： 软件运行时产生的一种不希望或不可接受的外部行为结果。

Terms

举例
爱国者导弹

软件
系统

计时用系统时钟
并以整数表达

缺少防
范措施

1991年2月，一次拦截失
利造成28名美国士兵丧生

缺陷 → 激活 → 故障 → 演变 → 失效

寄存器存储导致误差
（0.000000095）$_{10}$

$0.000000095 \times 100h \times 60 \times 60 \times 10 = 0.34s$

Software Testing Goals

✧ Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.

- To demonstrate to the developer and the customer that the software meets its requirements.
- To discover faults or defects in the software where its behavior is incorrect or not in conformance with its specification.
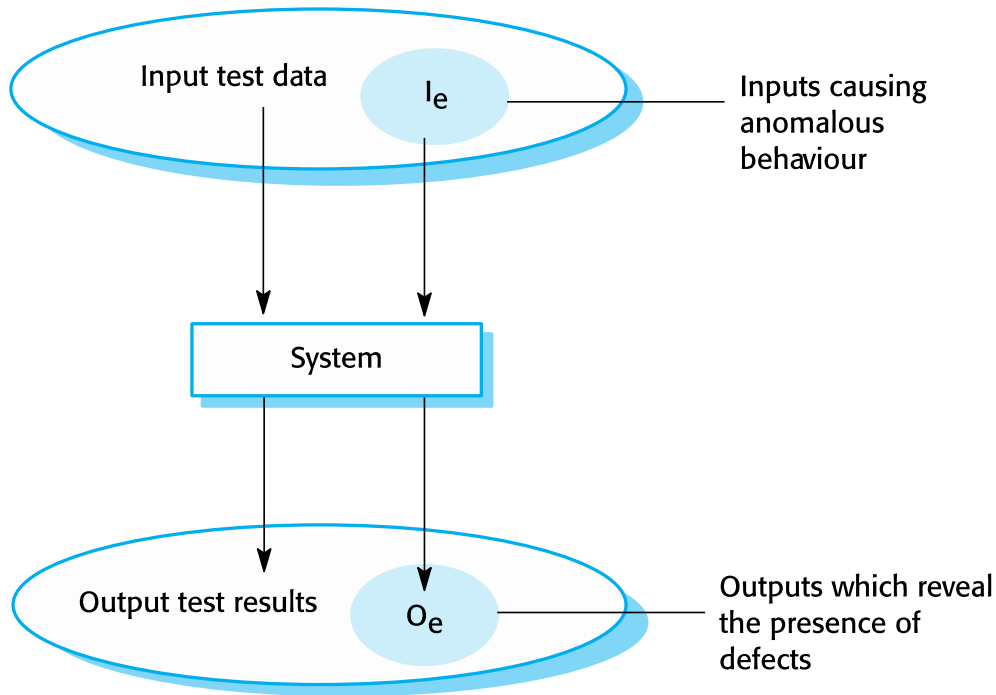
Validation testing

A successful test shows that the system operates as intended.

Defect testing

A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

Overview of Software Testing

Input test data

I_e

Inputs causing anomalous behaviour

System

Output test results

O_e

Outputs which reveal the presence of defects

✧ A test case is a particular choice of input data to be used in testing a program and the expected output or behavior.

✧ A test is a finite collection of test cases.

The goal is to try as many "critical" input combinations as possible, given the time and resource constraints.

Psychology of Testing

✧ Human beings tend to be highly goal-oriented, and establishing the proper goal has an important psychological effect on them.

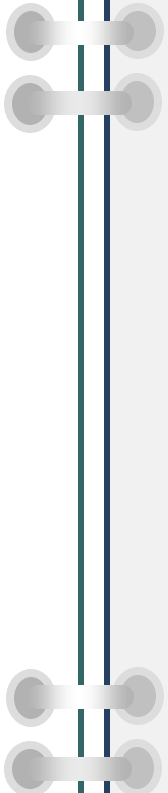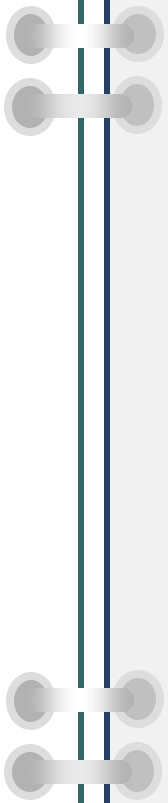|		
If our goal is to demonstrate that a program has no errors, then we will tend to select test data that have a low probability of causing the program to fail.	**VS**	If our goal is to demonstrate that a program has errors, our test data will have a higher probability of finding errors.

Testing shows the presence, not the absence of bugs.

Edsger W. Dijkstra, 1972

Software Testing Principles

1. A necessary part of a test case is a definition of the expected output or result.

2. A programmer should avoid attempting to test his or her own program.

3. A programming organization should not test its own programs.

4. Any testing process should include a thorough inspection of the results of each test.

5. Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.

Software Testing Principles

6. Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do.

7. Avoid throwaway test cases unless the program is truly a throwaway program.

8. Do not plan a testing effort under the tacit assumption that no errors will be found.

9. The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.

10. Testing is an extremely creative and intellectually challenging task.

Defect Clustering 理解和PP的区别

✧ Defect clustering means a small number of modules containing most of the defects.

- Basically, the defects are not distributed uniformly across the entire application.

✧ Defect clustering is based on "Pareto Principle". It means that 80% of the defects found are due to 20% of the modules in the application.

Number of errors found already
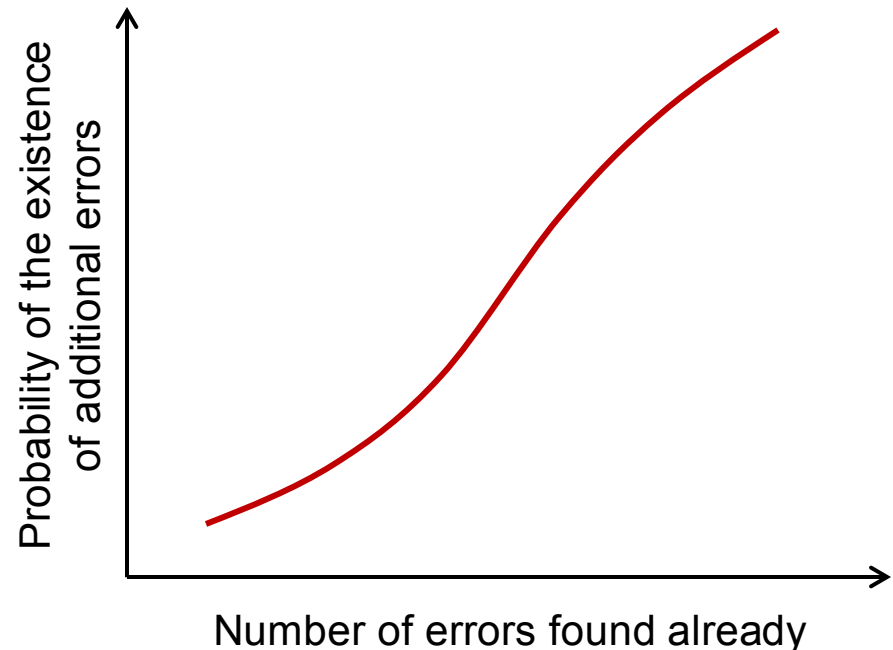
# Pesticide Paradox

✧ Pesticide paradox means that if the same set of test cases are executed again and again over the period of time then these set of tests are not capable enough to identify new defects in the system.

✧ The set of test cases needs to be regularly reviewed and revised. If required, a new set of test cases can be added and the existing test cases can be deleted if they are not able to find any more defects from the system.

Software Tester   Software Bug
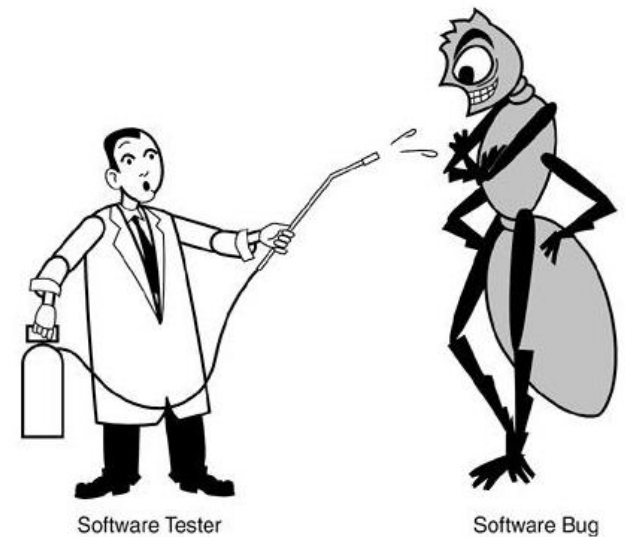
Process of Software Testing

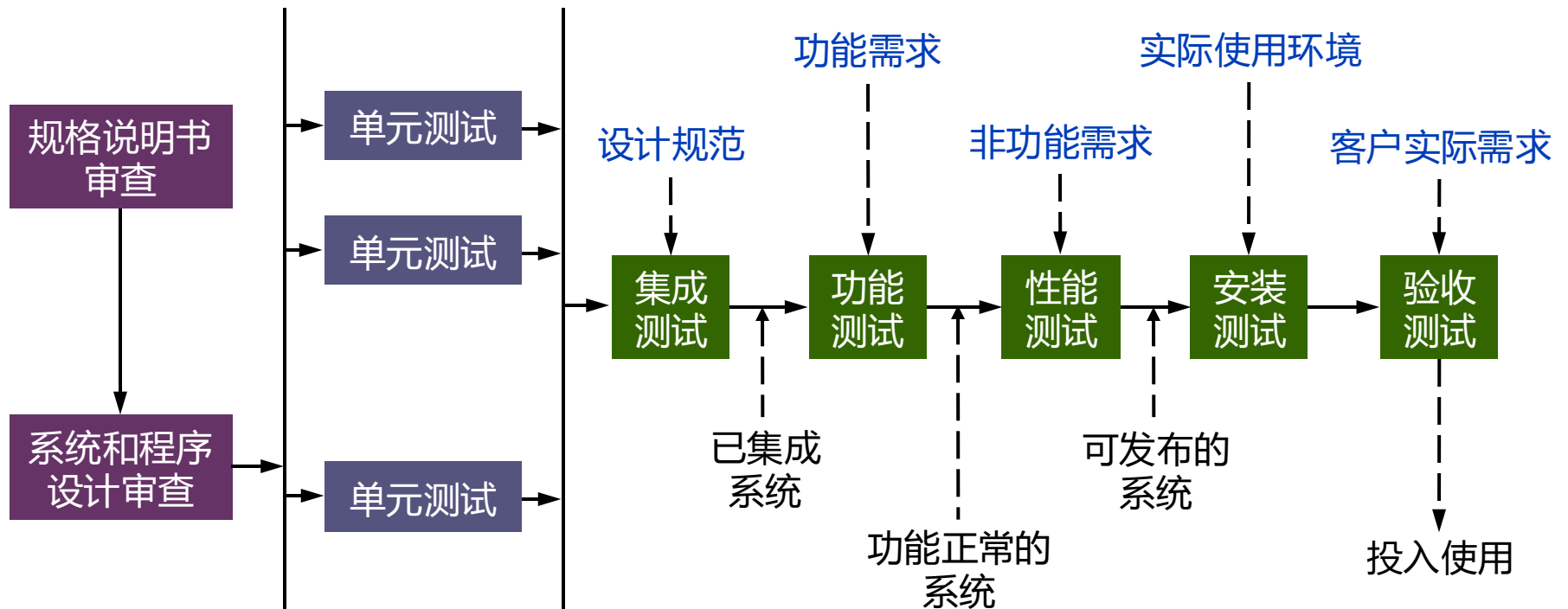| 计划		准备		执行		报告
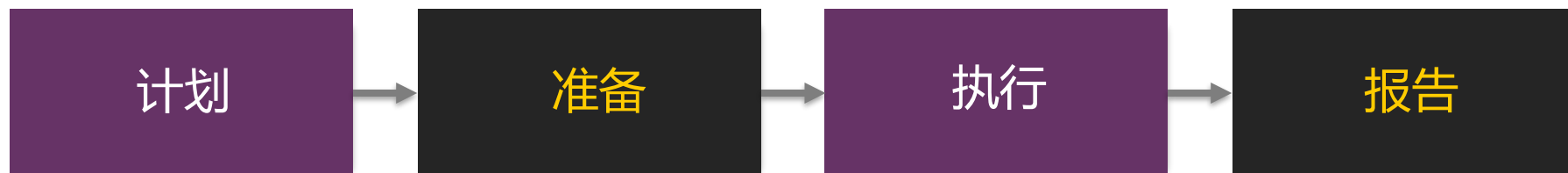
- 识别测试需求
- 分析质量风险
- 拟定测试方案
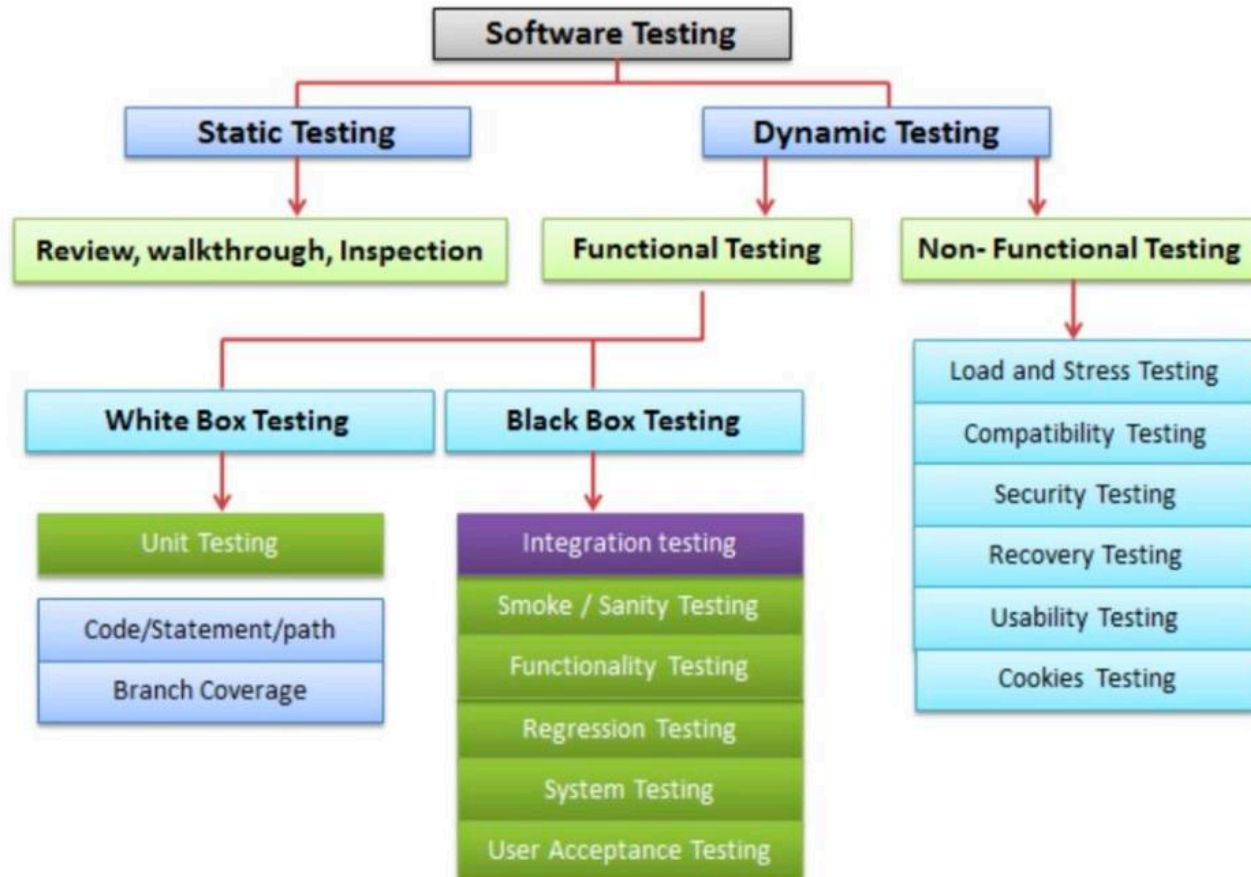- 制定测试计划

- 组织测试团队
- 设计测试用例
- 开发工具和脚本
- 准备测试数据

- 获得测试版本
- 执行和实施测试
- 记录测试结果
- 跟踪和管理缺陷

- 分析测试结果
- 评价测试工作
- 提交测试报告

Phases of Software Testing

Types of Software Testing

Static Testing VS Dynamic testing

理解区别

Static Testing

✧ **Walkthroughs**

- Walkthrough is a method of conducting informal group/individual review. A small set of paper test cases is prepared. Each test case is mentally executed; that is, the test data are ''walked through'' the logic of the program.

✧ **Inspections**

- A code inspection is a set of procedures and error-detection techniques for group code reading. These involve people examining the source representation with the aim of discovering anomalies and defects.

- Inspections not require execution of a system so may be used before implementation. They may be applied to any representation of the system (requirements, design, configuration data, test data).

Inspections and Testing
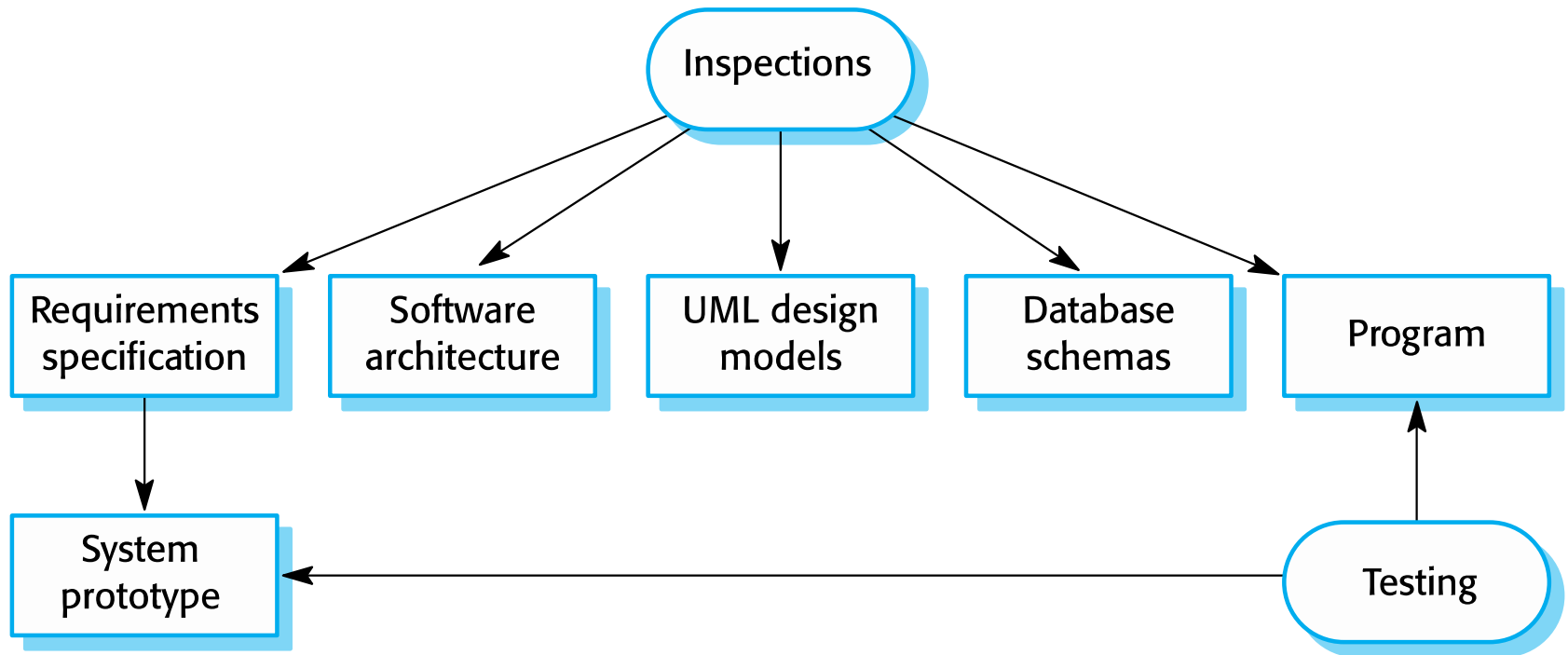
Advantages of Inspections

✧ During testing, errors can mask (hide) other errors. Because inspection is a static process, you don't have to be concerned with interactions between errors.

✧ Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.

✧ As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

Dynamic Testing

✧ Dynamic Testing is a software testing method used to test the dynamic behavior of software code.

  ▪ White box testing

  ▪ Black box testing

✧ The main purpose of dynamic testing is to test software behavior with dynamic variables and finding weak areas in software runtime environment.
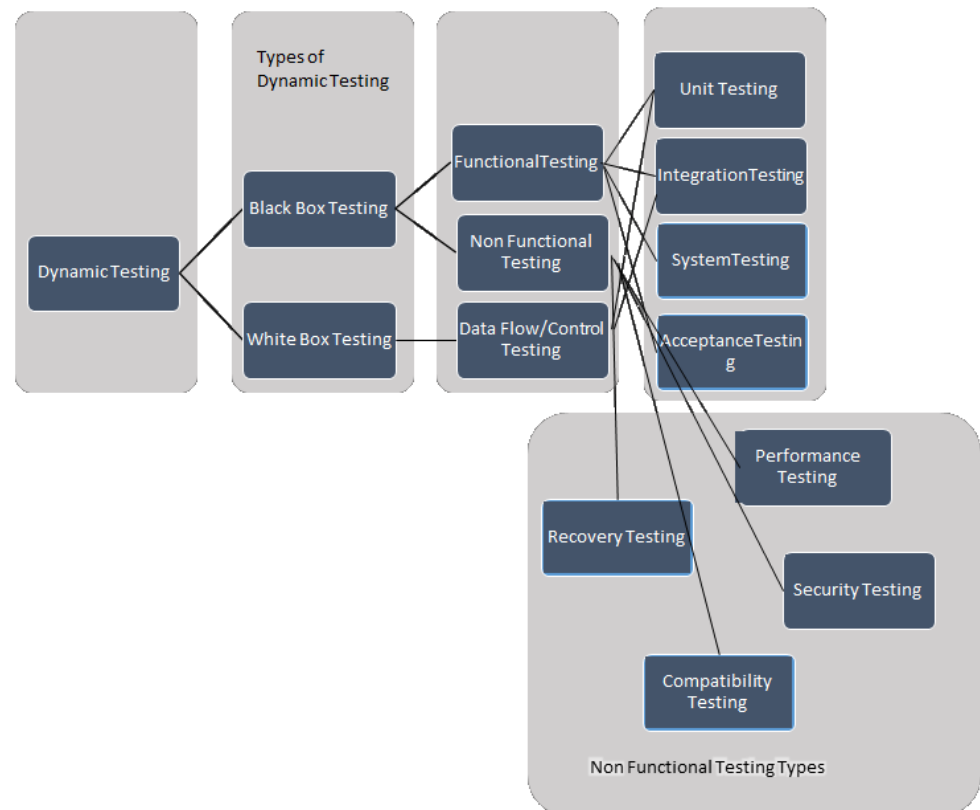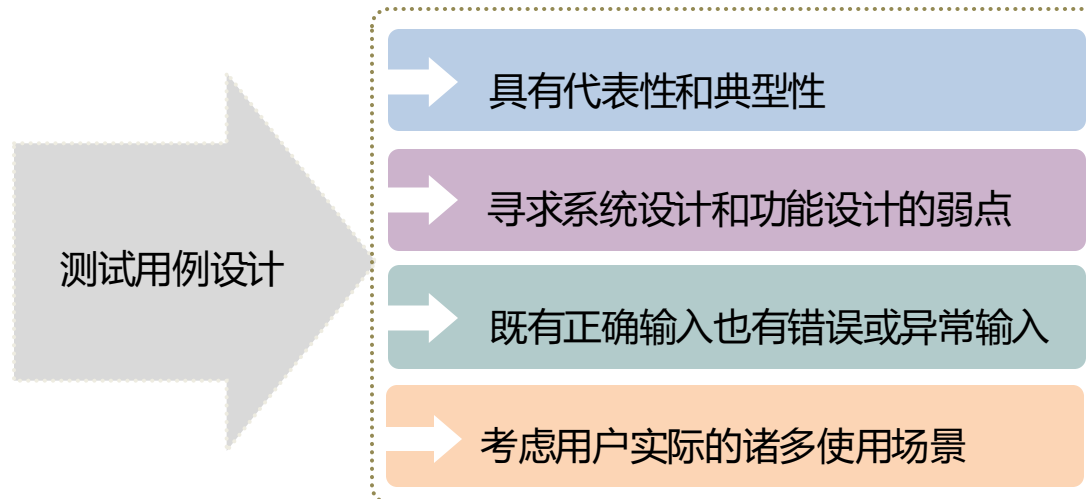
Black Box Testing VS White Box Testing

1.白盒-控制流图
2.理解黑白区别

Test-Case Design

✧ Test-case design is so important because complete testing is impossible. The obvious strategy, then, is to try to make tests as complete as possible.

✧ Given constraints on time and cost, the key issue of testing becomes: what subset of all possible test cases has the highest probability of detecting the most errors?

测试用例设计 →

具有代表性和典型性

寻求系统设计和功能设计的弱点

既有正确输入也有错误或异常输入

考虑用户实际的诸多使用场景

Test-Case Design

✧ The recommended procedure is to develop test cases using the black box methods and then develop supplementary test cases with white box methods.

White box testing

Statement coverage
Decision coverage
Condition coverage
Decision/condition coverage
Multiple-condition coverage

+

Black box testing

Equivalence partitioning
Boundary value analysis
Error guessing

✧ It is recommended to use a combination of most, if not all, of them to design a rigorous test of a program, since each method has distinct strengths and weaknesses.

White Box Testing

✧ White box testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security.

▪ White box testing is concerned with the degree to which test cases exercise or cover the logic of the program.

✧ Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today.
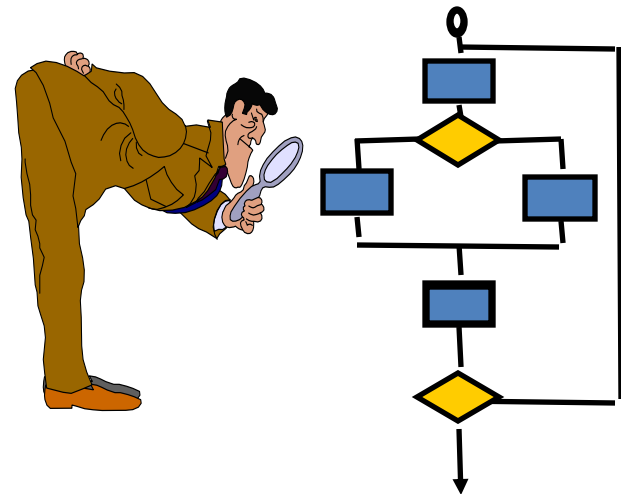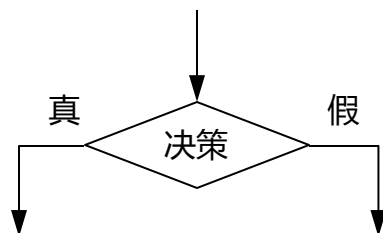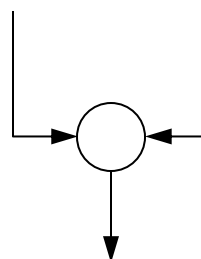
Control Flow Graph

```
public void foo(int A, int B, int X) {
    if(A>1 && B==0) {
        X=X/A;
    }
    if(A==2 || X>1) {
        X=X+1;
    }
}
```
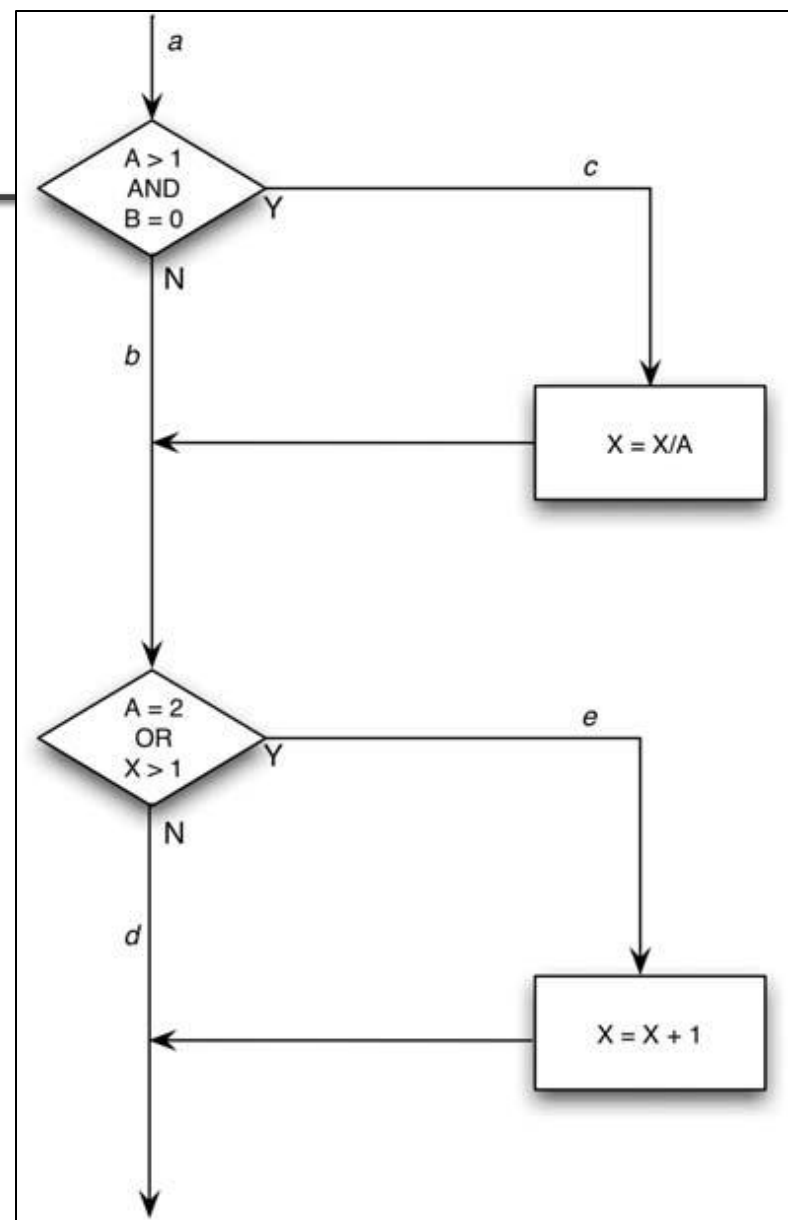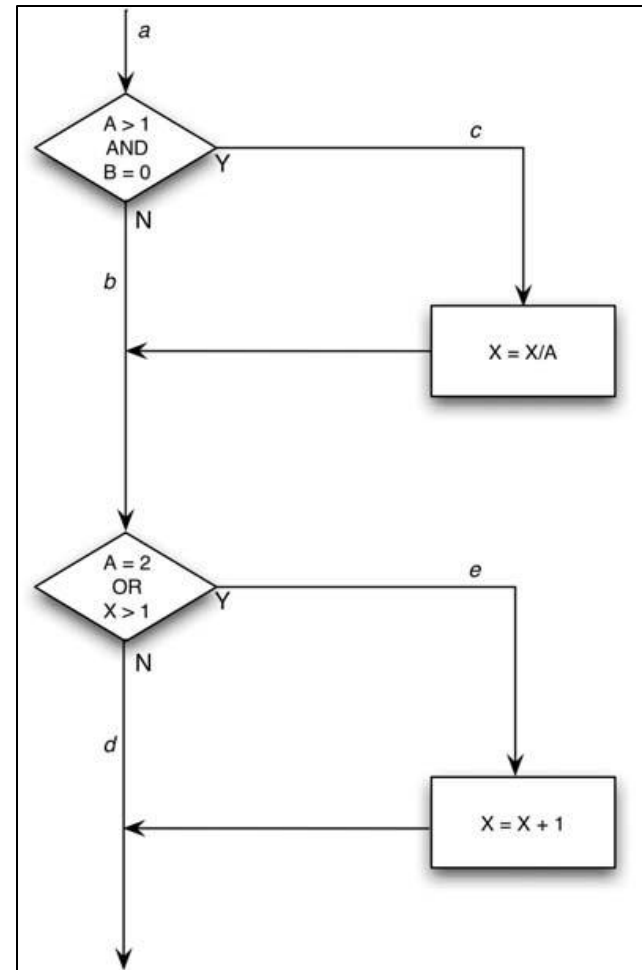

顺序计算

判断节点

合并节点

真   决策   假

计算
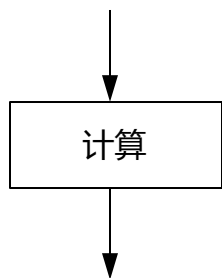
Statement Coverage

✧ This technique requires every possible statement in the code to be tested at least once during the testing process.

✧ You could execute every statement by writing a single test case that traverses path *ace*.

- A=2, B=0, X=3

The statement coverage criterion is so weak that it generally is useless.

Decision Coverage


♦ This technique checks every possible path (if-else and other conditional loops) of a software application.

- In other words, each branch direction must be traversed at least once.

♦ Decision coverage can be met by two test cases covering paths *ace* and *abd* or, *acd* and *abe*.

  - A=3, B=0, X=3
  - A=2, B=1, X=1
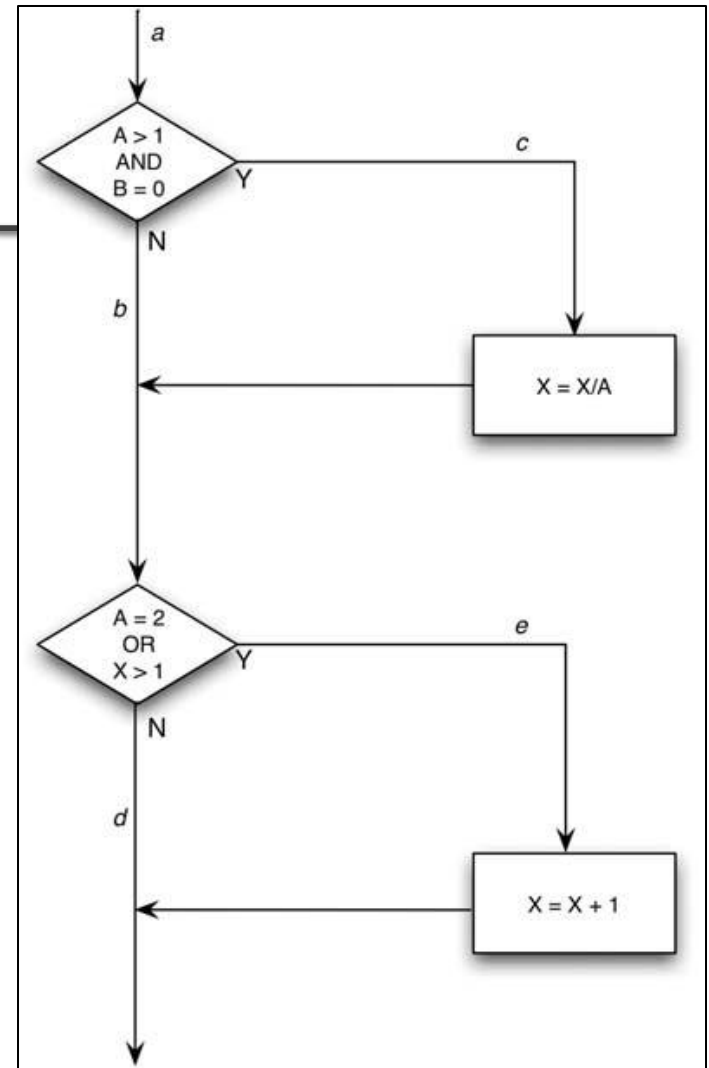
Decision coverage usually can satisfy statement coverage.

Condition Coverage

✧ In this case, you write enough test cases to ensure that each condition in a decision takes on all possible outcomes at least once.

✧ There are four conditions: A>1, B=0, A=2, and X>1. Hence, enough test cases are needed to force the situations where A>1, A<=1, B=0, and B<>0 are present at point a and where A=2, A<>2, X>1, and X<=1 are present at point b.

- ▪ A=1, B=0, X=3
- ▪ A=2, B=1, X=1
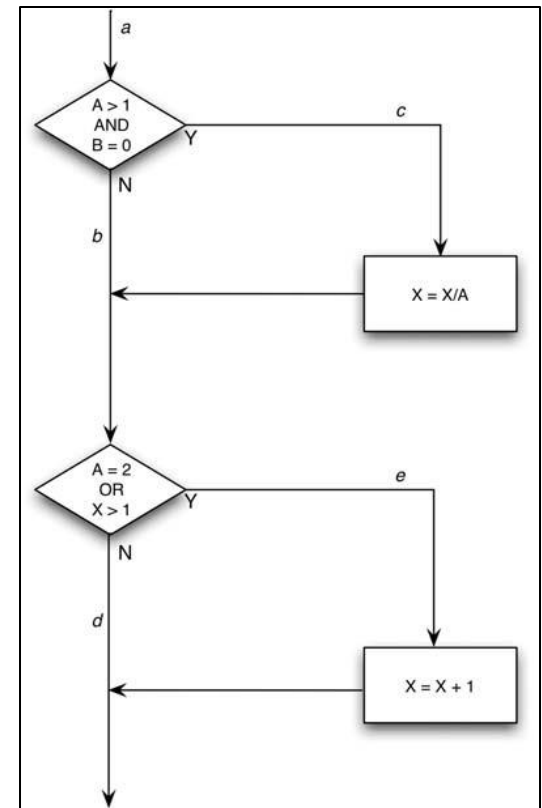
Decision/Condition Coverage

✧ Decision/condition coverage requires sufficient test cases such that each condition in a decision takes on all possible outcomes at least once, and each decision takes on all possible outcomes at least once.

- A=2, B=0, X=4
- A=1, B=1, X=1
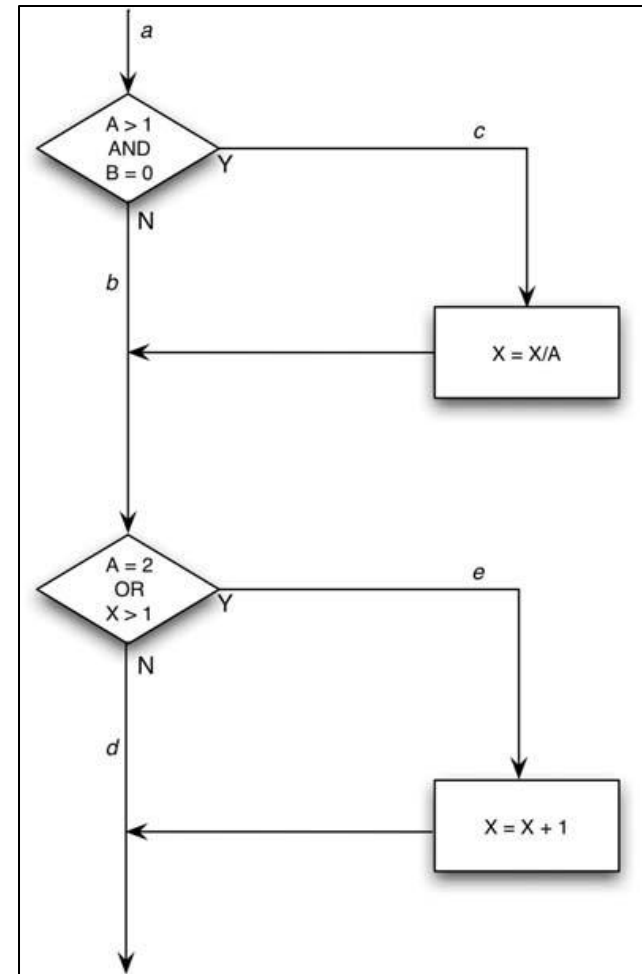
Multiple-Condition Coverage

♦ This criterion requires that you write sufficient test cases such that all possible combinations of condition outcomes in each decision are invoked at least once.

1. A>1, B=0	5. A=2, X>1
2. A>1, B<>0	6. A=2, X<=1
3. A<=1, B=0	7. A<>2, X>1
4. A<=1, B<>0	8. A<>2, X<=1

A=2, B=0, X=4	Covers 1, 5
A=2, B=1, X=1	Covers 2, 6
A=1, B=0, X=2	Covers 3, 7
A=1, B=1, X=1	Covers 4, 8

♦ A set of test cases satisfying the multiple-condition criterion also satisfies the decision coverage, condition coverage, and decision/condition coverage criteria.

Advantages and Disadvantages

✧ Advantages of white box testing:

- Forces test developer to reason carefully about implementation.

- Reveals errors in "hidden" code.

- Spots the Dead Code or other issues with respect to best programming practices.

✧ Disadvantages of white box testing:

- Expensive as one has to spend both time and money to perform white box testing.

- In-depth knowledge about the programming language is necessary to perform white box testing.

Black Box Testing

✧ Black box testing is a high level of testing that focuses on the behavior of the software. It involves testing from an external or end-user perspective.

✧ A well-selected test case should have two properties:

- It reduces, by more than a count of one, the number of other test cases that must be developed to achieve some predefined goal of "reasonable" testing.

- It covers a large set of other possible test cases. That is, it tells us something about the presence or absence of errors over and above this specific set of input values.

Equivalence Partitioning

✧ Identifying the equivalence classes: identify groups of inputs that have common characteristics and should be processed in the same way.

Input equivalence partitions

Output partitions

System

Possible inputs

Correct outputs

Possible outputs
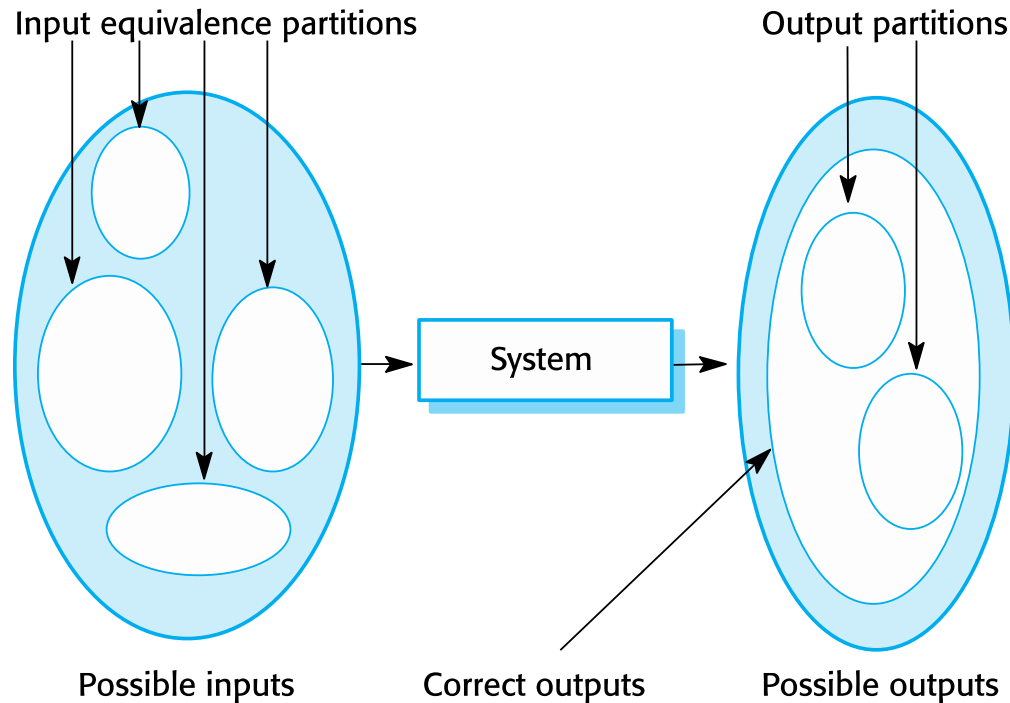
Types of Equivalence Class

✧ **有效等价类**是对规格说明有意义、合理的输入数据构成的集合。利用有效等价类，能够检验程序是否实现了规格说明中预先规定的功能和性能。

✧ **无效等价类**是对规格说明无意义、不合理的输入数据构成的集合。利用无效等价类，可以发现程序异常处理的情况，检查被测对象的功能和性能的实现是否有不符合规格说明要求的地方。
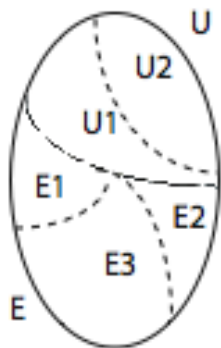
- E 表示所有正常和合法的输入
- U 表示所有异常和非法的输入

Equivalence Partitioning

⬦ Given an input or external condition, identifying the equivalence classes is largely a heuristic process.

- If an input condition specifies a range of values, identify one valid equivalence class and two invalid equivalence classes.

- If an input condition specifies a set of input values, and there is reason to believe that the program handles each differently, identify a valid equivalence class for each and one invalid equivalence class.

- If an input condition specifies a "must-be" situation, identify one valid equivalence class and one invalid equivalence class.

⬦ If there is any reason to believe that the program does not handle elements in an equivalence class identically, split the equivalence class into smaller ones.

Examples

1) 程序的输入参数 x 是小于100大于10的整数。

1个有效等价类： 10 < x < 100

2个无效等价类： x≤10 和 x≥100

2) 姓名是长度不超过20的非空字符串，且只由字母组成，数字和其他字符都是非法的。

1个有效等价类： 满足了上述所有条件的字符串

3个无效等价类：

- 空字符串
- 长度超过20的字符串
- 包含了数字或其它字符的字符串

Examples

3) 某程序根据不同的学历分别计算岗位工资，其中学历可以是专科、本科、硕士、博士等四种类型。

    4个有效等价类：专科、本科、硕士、博士

    1个无效等价类：其他学历

4)
```
struct student {
        string name;
        string course[100];
        int grade[100];
    }
```
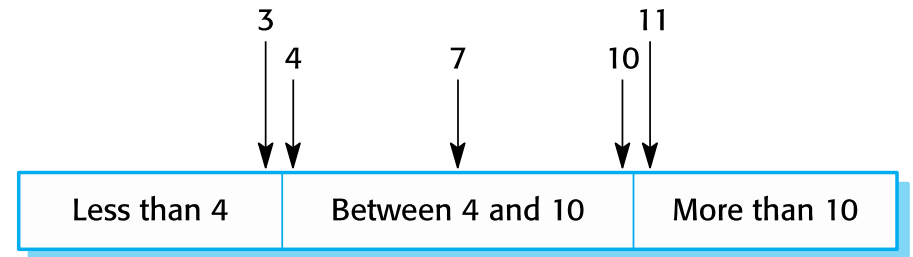
对复合数据类型中的每个元素进行等价类划分，再将这些等价类进行组合，最终形成对软件整个输入域的划分。

Identifying the Test Cases

✧ We then make use equivalence classes to identify the test cases. The process is as follows:

- ▪ Assign a unique number to each equivalence class.

- ▪ Until all valid equivalence classes have been covered by (incorporated into) test cases, write a new test case covering as many of the uncovered valid equivalence classes as possible.

- ▪ Until your test cases have covered all invalid equivalence classes, write a test case that covers one, and only one, of the uncovered invalid equivalence classes.

Boundary Value Analysis

✧ Experience shows that test cases that explore boundary conditions have a higher payoff than test cases that do not.

✧ Boundary conditions are those situations directly on, above, and beneath the edges of input equivalence classes and output equivalence classes.

3
4   7   11
                      10

| Less than 4 | Between 4 and 10 | More than 10 |

Number of input values

9999   100000
    10000   50000   99999

| Less than 10000 | Between 10000 and 99999 | More than 99999 |
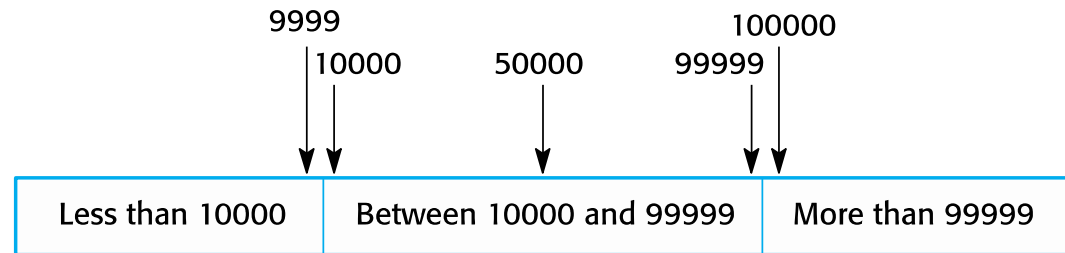
Input values

Error Guessing

✧ Given a particular program, some people surmise, both by intuition and experience, certain probable types of errors and then write test cases to expose those errors.

- The basic idea is to enumerate a list of possible errors or error-prone situations and then write test cases based on the list.

- Another idea is to identify test cases associated with assumptions that the programmer might have made when reading the specification.

Black Box Testing VS White Box Testing

Parameter	Black Box testing	White Box testing
Base of Testing	Testing is based on external expectations; internal behavior of the application is unknown.	Internal working is known, and the tester can test accordingly.
Usage	This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing.	Testing is best suited for a lower level of testing like Unit Testing, Integration testing.
Programming knowledge	Programming knowledge is not needed to perform Black Box testing.	Programming knowledge is required to perform White Box testing.
Tested by	Performed by the end user, developer, and tester.	Usually done by tester and developer.
Time	It is less exhaustive and time-consuming.	Exhaustive and time-consuming method.