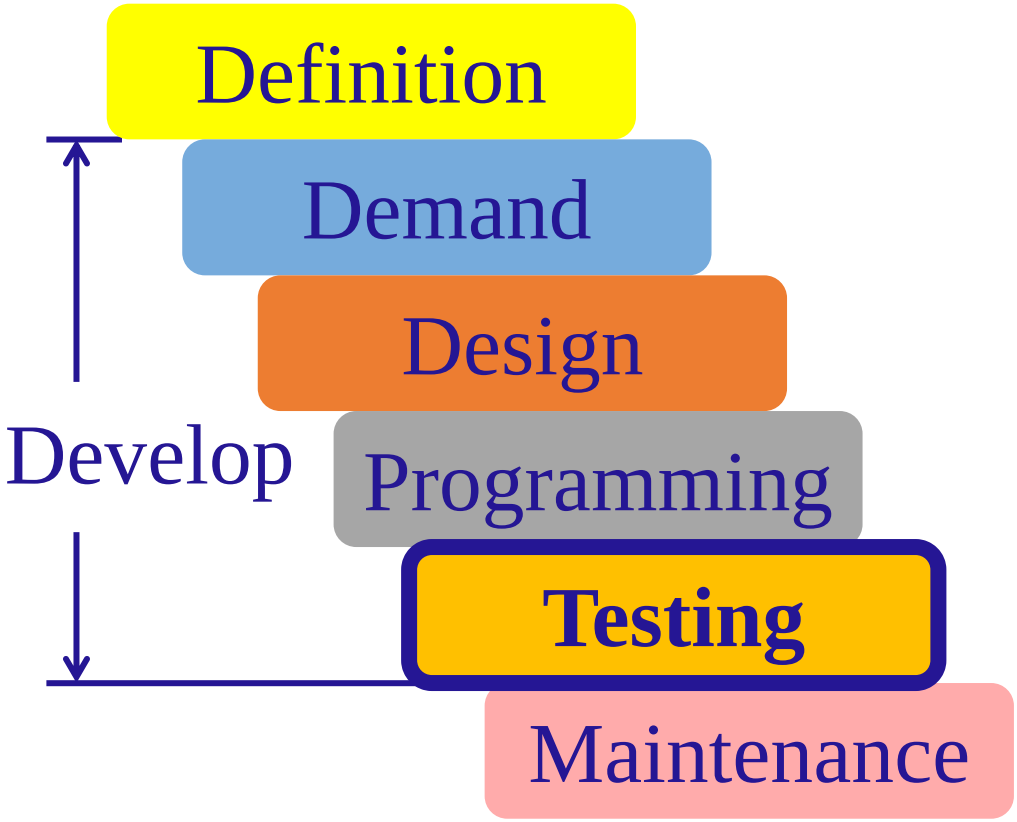
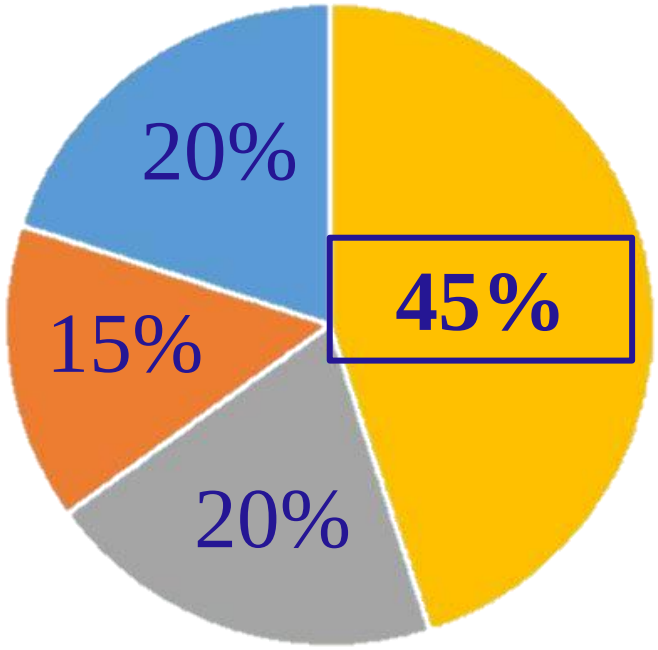
SOFTWARE TESTING

苏临之

sulinzhi029@nwu.edu.cn

Software Life Cycle





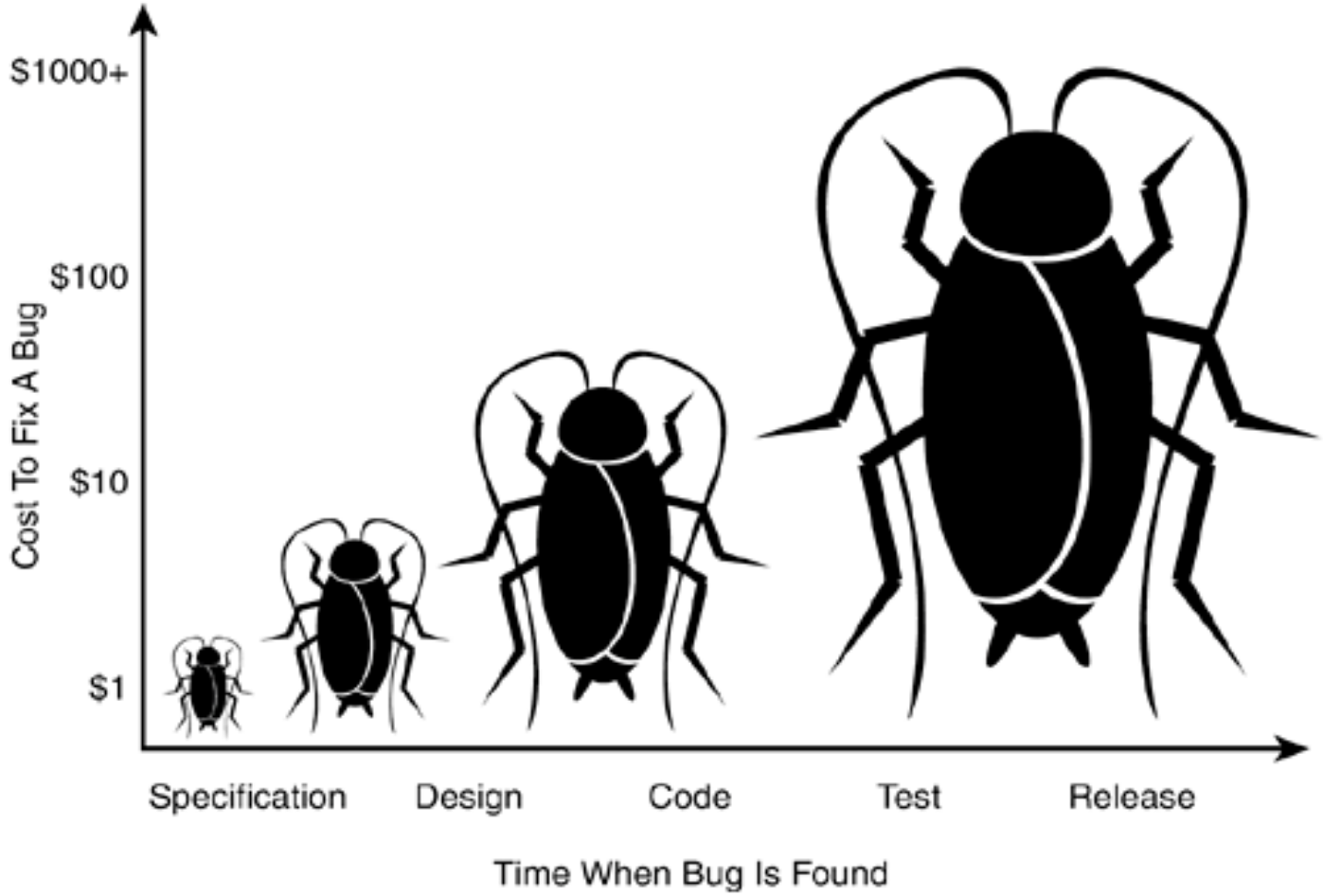
Error, Failure, Fault and Defect

- Error: The software can not operate as fully anticipated.
- Failure: The software engenders wrong results caused by errors.
- Fault: The existing physical problems that may or may not result in failures.
- Defect: The software does not work as required. A software defect, or bug as often referred to, may or may not result in an error.

Identification of a Software Defect

- A software defect occurs when one or more of the following five rules is true:
  1. The software doesn't do something that the product specification says it should do.
  2. The software does something that the product specification says it shouldn't do.
  3. The software does something that the product specification doesn't mention.
  4. The software doesn't do something that the product specification doesn't mention but should.
  5. The software is difficult to understand, hard to use, slow, or in the tester's eyes will be viewed not good by the end user.

Cost of Defects



Aim of Software Testing

- Software testing is actually a process or a series of processes, identifying that the computer has achieved the functions it should accomplish and has not achieved the ones it did not mention. So the aim of software testing is **to find the defects**.
- One can never think that the aim can be defined as the certification of software validity.

Software Development Process

- In a software development process, some basic elements are indispensable:

1. Customer requirements
2. Specifications
3. Schedules
4. Software design documents
5. Test documents

Software Development Lifecycle

- Four frequently used models
  1. Big-Bang
  2. Code-and-Fix
  3. Waterfall
  4. Spiral

Some Important Testing Axioms

- Not all the found bugs will be fixed. Generally four reasons:
  - There's not enough time.
  - It's really not a bug.
  - It's too risky to fix.
  - It's just not worth it.
- The more bugs one have found, the more bugs there are.
- The more one tests software, the more immune it becomes to his tests, i.e. the pesticide paradox (杀虫剂效应/悖论).

Test-to-Pass & Test-to-Fail

- Two fundamental approaches to testing software
  - Test-to-Pass: Designing and running test cases to assure only what the software can minimally work is called test-to-pass.
  - Test-to-Fail: Designing and running test cases with the sole purpose of breaking the software is called test-to-fail or error-forcing.

Four Basic Techniques

- Static Black-Box Testing
- Dynamic Black-Box Testing
- Static White-Box Testing
- Dynamic White-Box Testing

Four Basic Techniques

- **Static Black-Box Testing**
- Dynamic Black-Box Testing
- Static White-Box Testing
- Dynamic White-Box Testing

Static Black-Box Testing

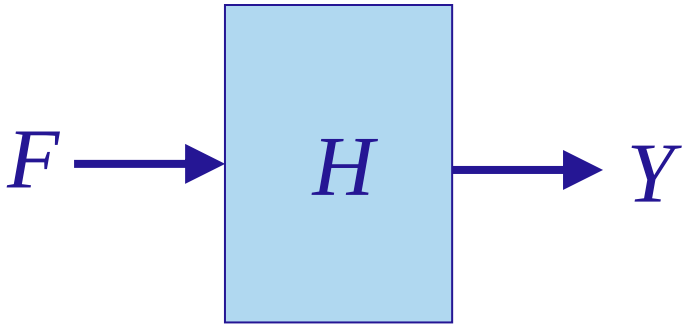
- High-Level Review of the Specification
  - Pretend to be the customer
  - Research existing standards and guidelines
  - Review and test similar software
- Low-Level Review of the Specification
  - Specification attributes
  - Specification terminology

Four Basic Techniques

- Static Black-Box Testing
- **Dynamic Black-Box Testing**
- Static White-Box Testing
- Dynamic White-Box Testing

Dynamic Black-Box Testing

- Suppose the software is a black box, and for each input *F*, the box processes it by using its inner function *H*, and finally we can get an output *Y*.
- So if there is an error in the box (*H*), then *Y* will also be wrong. Thus, we design proper test cases (as *F*) and can check *H* by examining *Y*.
- Reference: in an LTI system, we have $y = h*f$ or $Y = HF$.



Dynamic Black-Box Testing

- If the entire available inputs are viewed as a universal set (universe), then the test cases comprise one subset. So to design test cases entail finding a specific subset of the input universe.
- Given fixed time and space, which subset will lead to finding the most bugs? That serves as the key point to design test cases, and several techniques have been proposed.

An Example

- **Description**
  - 某企业的报表处理系统要求用户输入处理报表的日期，日期限制在2003年1月至2008年12月，即系统只能对该段期间内的报表进行处理，如日期不在此范围内，则显示输入错误信息。输入信息要求：系统日期规定由表征年、月的6位数字字符组成，前4位代表年，后2位代表月。
- **Please consider why it is suitable to use dynamic black-box testing and then design test cases to check the function for date input.**

Analysis

- In the example, the aim is to test the function of the system, i.e. it is not necessary to know the way it works. Actually, the system must consist of several units and somehow be combined together, but we do not care that. All we need to do is check whether the outputs to test cases are what they should be.
- Therefore, it is suitable to use dynamic black-box testing.

An Example

- **Description**
  - 某企业的报表处理系统要求用户输入处理报表的日期，日期限制在2003年1月至2008年12月，即系统只能对该段期间内的报表进行处理，如日期不在此范围内，则显示输入错误信息。输入信息要求：系统日期规定由表征年、月的6位数字字符组成，前4位代表年，后2位代表月。
- **Please design test cases to check the function for date input.**

Techniques

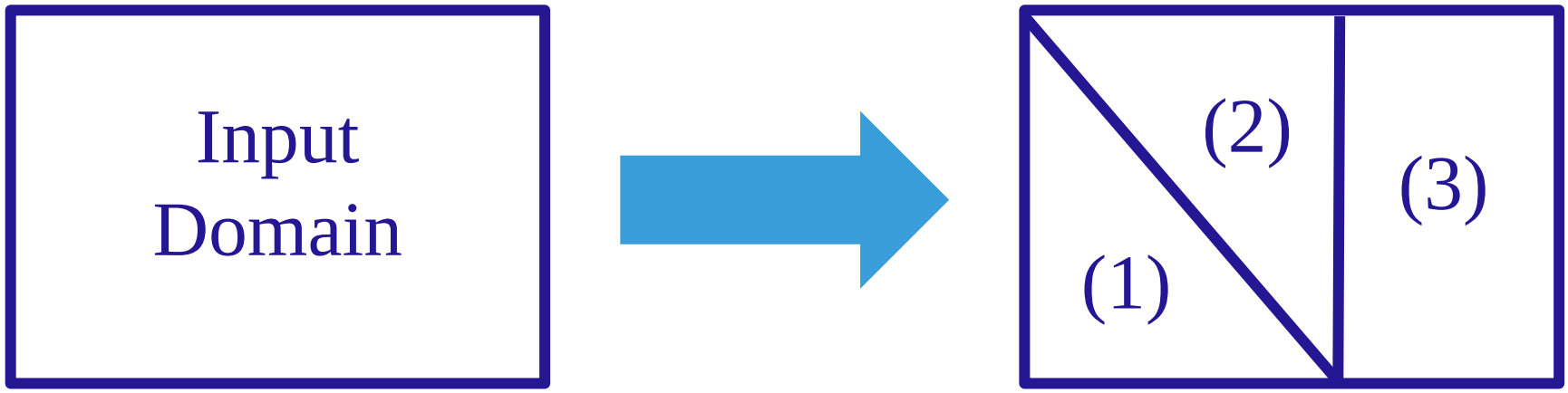
- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table
- Cause-Effect Diagram
- Error Guessing

Techniques

- **Equivalence Partitioning**
- Boundary Value Analysis
- Decision Table
- Cause-Effect Diagram
- Error Guessing

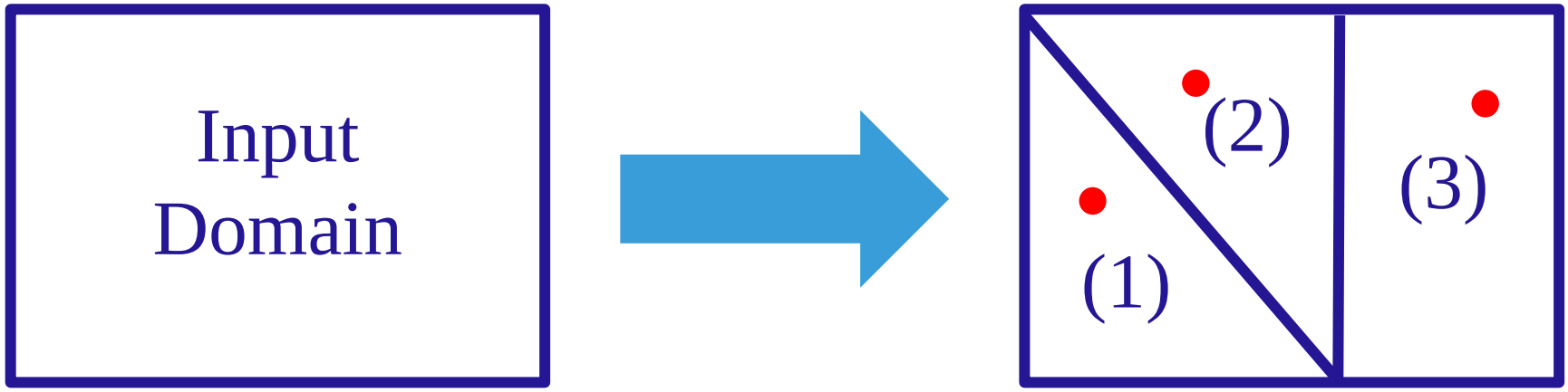
Equivalence Class

- An equivalence class is a subset that consists of test cases, testing the same thing or revealing the same bug.
- If we view the input domain as the universe, a certain subset is an equivalence class.



Equivalence Partitioning

- Upon the division of the input domain into several equivalence classes, we select only a few (maybe only one) representative test cases from each class, reducing the inputs and thus avoiding redundant full testing. This is called the equivalence partitioning.



Valid Equivalence Class

- In the issue, a legal input consists of 6 digits, in which the former 4 and the latter 2 represent the year from 2003 to 2008 and the month, respectively. Thus, we have the legal input classes as follows.

① 6位数字字符

② 前4位范围2003~2008

③ 后2位范围01~12

- These three legal input classes are called **valid equivalence classes (有效等价类, VECs)**.

Invalid Equivalence Class

- By applying the NOT operation to the legal inputs, we can get the following illegal inputs.

④ 有非数字字符；⑤ 输入少于6位；⑥ 输入多于6位
⑦ 前4位小于2003；⑧ 前4位大于2008
⑨ 后2位小于01；⑩ 后2位大于12

- These seven illegal input classes are called **invalid equivalence classes (无效等价类, IECs)**.

Equivalence Class List

	有效等价类	无效等价类
输入格式	6位数字字符 ①	有非数字字符 ④ 少于6个数字字符 ⑤ 多于6个数字字符 ⑥
前4位范围	2003～2008 ②	小于2003 ⑦ 大于2008 ⑧
后2位范围	01～12 ③	小于01 ⑨ 大于12 ⑩

Establishing VECs and IECs

- Equivalence classes (ECs) are established according to the input conditions in an issue. First, analyze the conditions to extract several factors, which are used to establish the VECs. Then by using the NOT operation, we can establish the IECs.
- There are two points here. First, how can we determine the range of an ECs? Second, one VEC may correspond several IECs, i.e. this is not a one-to-one correspondence, so why is it like that? Here some empirical principles are used.

Basic Principles

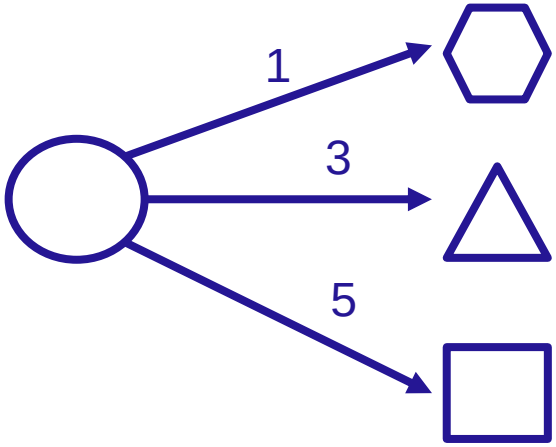
- Value Range (取值范围): one VEC & two IECs
- Value Number (取值个数): one VEC & two IECs
- Multiple Branching (分支不同输入值): one VEC for each branch value & one IEC
- "Must-Be (必须是)": one VEC & one IEC
- If the elements are thought to be treated unequally for any reason, or if an EC is too general, then we should divide it into some smaller ECs.

Value Range and Value Number

- Value range is an open or closed interval mathematically (e.g. $2 < x \leq 10$). This is an AND relationship and by applying the NOT operation AND turns out be OR (e.g. $x \leq 2$ or $x > 10$). Hence, we should establish two IECs.
- Value number is the special case where the boundaries of the range are the same (e.g. $x = 2$). By applying the NOT operation to "equal", two situations should also be considered: "larger than" and "smaller than" ($x < 2$ or $x > 2$).

Multiple Branching

- Multiple branching indicates that different values corresponds to different cases. For instance, if the legal input set is {1, 3, 5} and the elements corresponds to three different output cases, then we should establish three VECs and one IEC (unequal to the three numbers).



"Must Be / Should be"

- "Must be / should be" is the special case where the legal input set contains only one element, in which case we need to establish only one VEC and one IEC.
- Sometimes, the condition "*x* is sth." in an issue infers "*x* must be sth.". This should be distinguished from "be equal to" a number. In a word, multiple branching and "must be" emphasize the cases.

Equivalence Class List

	有效等价类	无效等价类
输入格式	6位数字字符 ①	有非数字字符 ④ 少于6个数字字符 ⑤ 多于6个数字字符 ⑥
前4位范围	2003～2008 ②	小于2003 ⑦ 大于2008 ⑧
后2位范围	01～12 ③	小于01 ⑨ 大于12 ⑩

Methods for Giving out Test Cases

- Test cases can be either legal or illegal. Legal test cases are given according to the VECs while illegal test cases are give according to IECs.
- If an input implies the AND relationship, then the corresponding test case should cover as many ECs as possible. If an input implies a single factor, then the corresponding test case only need to cover this single factor.

Legal Test Cases

测试用例	期望结果	覆盖范围
200609	输入有效	①②③

- Here the legal input includes three factors with relationship AND: ①, ② and ③, and hence the corresponding test case should cover these three.

Illegal Test Cases

测试数据	期望结果	覆盖范围
2006ab	输入无效	④（非数字字符）

- Here the illegal input includes only one factor ④, and hence the corresponding test case only need cover this EC with the other factors remaining legal.
- Good Test Cases: 2006ab / 20051x / a00707
- Bad Test Cases: 2006abc /2005a

Illegal Test Cases

测试数据	期望结果	覆盖范围
2006ab	输入无效	④（非数字字符）
20065	输入无效	⑤（少于6位）
2006005	输入无效	⑥（多于6位）
200105	输入无效	⑦（前4位小于2003）
201005	输入无效	⑧（前4位大于2008）
200600	输入无效	⑨（后2位小于01）
200614	输入无效	⑩（后2位大于12）

Example 2

- **Description**
  - 程序从一个输入对话框中读取3个正整数值。这3个整数值代表了三角形三边的长度。程序显示提示信息，指出该三角形究竟是不等边三角形、等腰三角形还是等边三角形。
- **Please design test cases by using the equivalence partitioning technique.**

Equivalence Class List

	有效等价类	无效等价类
数据性质	数字字符 ①	
数据数量	等于3个 ②	
数据格式	正整数 ③	

Equivalence Class List

	有效等价类	无效等价类
数据性质	数字字符 ①	有非数字字符 ④
数据数量	等于3个 ②	少于3个 ⑤ 多于3个 ⑥
数据格式	正整数 ③	负数 ⑦ 分数 ⑧ 小数 ⑨ 零 ⑩

Equivalence Class List

	有效等价类	无效等价类
数据性质	数字字符 ①	有非数字字符 ④
数据数量	等于3个 ②	少于3个 ⑤ 多于3个 ⑥
数据格式	正整数 ③	负数 ⑦ 分数 ⑧ 小数 ⑨ 零 ⑩

Equivalence Class List

	有效等价类	无效等价类
数据性质	数字字符 ①	有非数字字符 ④
数据数量	等于3个 ②	少于3个 ⑤ 多于3个 ⑥
数据格式	正整数 ③	零 ⑦

Absorption and Division

- 在三角形类型判定例子中，“有非数字字符”和“负数”“分数”“小数”存在包含关系，因此可以被统一吸收。
- 但这样使得“有非数字字符”变得很笼统，而键盘上包含有符号字符和字母字符两种，在程序处理中可能这两者处理方式并不等价。这时就需要把这个大类划分成一些更细节的等价类。

Equivalence Class List

	有效等价类	无效等价类
数据性质	数字字符 ①	有字母字符 ④ 有符号字符 ⑤
数据数量	等于3个 ②	少于3个 ⑥ 多于3个 ⑦
数据格式	正整数 ③	零 ⑧

Limitations

- Equivalence partitioning is based on the hypothesis that all the elements in the class are in equal status and that one element can represent the class. But this is not always true.
- For instance, if the inequality $1 \leq x \leq 5$ is mistaken as $1 < x < 5$, and the legal input to the VEC is selected as 4 while the illegal ones to the IECs are selected as 0 and 6, then the mistake (probably vital and fatal) will not be detected.

Homework

- **被测程序的描述**

  某城市的电话号码由三部分组成。这三部分的名称和内容分别是：

  地区码：空白或3位数字；
  前　缀：非“0”或“1”开头的3位数；
  后　缀：4位数字。

- **假定被测试的程序能接受一切符合上述规定的电话号码，拒绝所有不符合规定的号码，请使用等价类划分方法列出等价类编号列表，然后举出适当的测试用例，并列表说明这些测试用例覆盖了哪些编号的等价类。**

THANK YOU!