软件测试知识点整理

测试基础知识

2.1.1 Error, Defect, fault,and Bug Terminology

错误、缺陷、bug区别

error错误：人类犯的错，如需求分析的错误，内因

bug是指程序中的错误或漏洞，是结果。bug的出现肯定是因为defect

defect缺陷：是程序本身的一些缺陷，是软件系统中导致系统以非预期方式运行的原因

failure失效：程序运行所获得的结果与预期不符，需求没有得到满足，导致因素有缺陷、硬件故障、辐射电磁波等环境因素

fault是程序设计上的功能错误，对系统影响较大

逻辑测试用例，物理测试用例（在逻辑基础上填入具体数据）

软件测试的目标

1. 检验产品需求
2. 发现软件缺陷

软件测试是对软件开发过程中形成的文档、数据以及程序进行的测试

软件质量可归纳为内部风险和外部风险两方面：

- 内部风险是软件开发商在即将发布时发现软件有重大缺陷
- 外部风险是指软件发布之后用户发现了不能容忍的缺陷

软件测试只能证明软件存在缺陷，不能证明软件没有缺陷

软件测试有两个基本职责：

- 验证：保证软件开发过程中某一具体阶段的工作产品与该阶段和前一阶段的需求的一致性
- 确认：保证最终得到的产品满足系统需求

用户对质量的需求可以用使用质量、外部质量和内部质量来反映：

- 使用质量：在规定的使用环境下，从用户角度，软件产品使特定用户在达到规定目标方面的能力
- 外部质量：软件产品在规定条件下使用时满足需求的程度
- 内部质量：反映软件产品在规定条件下使用时满足需求的能力的一种特性

使用质量可以用以下质量特性表述：功能性、可靠性、易用性、效率、维护性、可移植性

软件测试和软件质量保证的区别

软件测试是软件质量保证的一个环节，通俗地说，软件测试是对产品进行测试，质量保证是对软件过程和产品的质量都要保证

软件质量保证的基本目标：

- 软件质量保证工作有计划进行
- 客观地验证项目的产品和工作是否遵循恰当的标准、步骤和需求
- 将软件质量保证工作及结果通知给相关组别和个人
- 高级管理层了解在软件项目开发过程中存在的不能解决的问题

测试终止条件：

测试时间用尽、继续测试没有产生新失效、继续测试没有发现新缺陷、无法设计出新测试用例、继续测试回报很小、达到所要求的测试覆盖时、所有已发现的错误或缺陷都已被清除

软件测试原则：

- 所有软件测试都应追溯到用户需求
- 尽早和不断地进行软件测试
- 不可能完全地测试
  - 测试所有输入是不可能的
  - 系统或程序的所有路径不可能全部执行
  - 需求规格有缺陷时无法找出所有存在于分析和设计中的缺陷
  - 不可能穷尽所有输出
- 增量测试，由小到大
- 避免测试自己的程序
- 设计完善的测试用例
- 注意测试中的群集现象（大部分缺陷都集中在一小部分代码中）
- 确认缺陷有效性
- 合理安排测试计划
- 进行回归测试
- 测试结果的统计、分析及可视化
- 及时更新测试
- 测试成本占项目总成本的25%~50%

软件测试级别和模型

单元测试、集成测试、系统测试、验收测试。具体在最下面的章节中详细讲述

软件测试生命周期

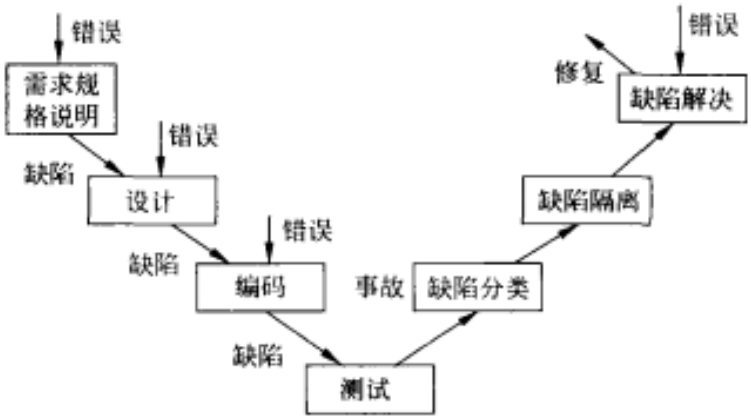


图 1-14　软件测试生命周期

开发和测试模型

- V模型

左边是开发阶段，右边是测试阶段，反映了测试阶段和开发阶段各阶段的对应关系

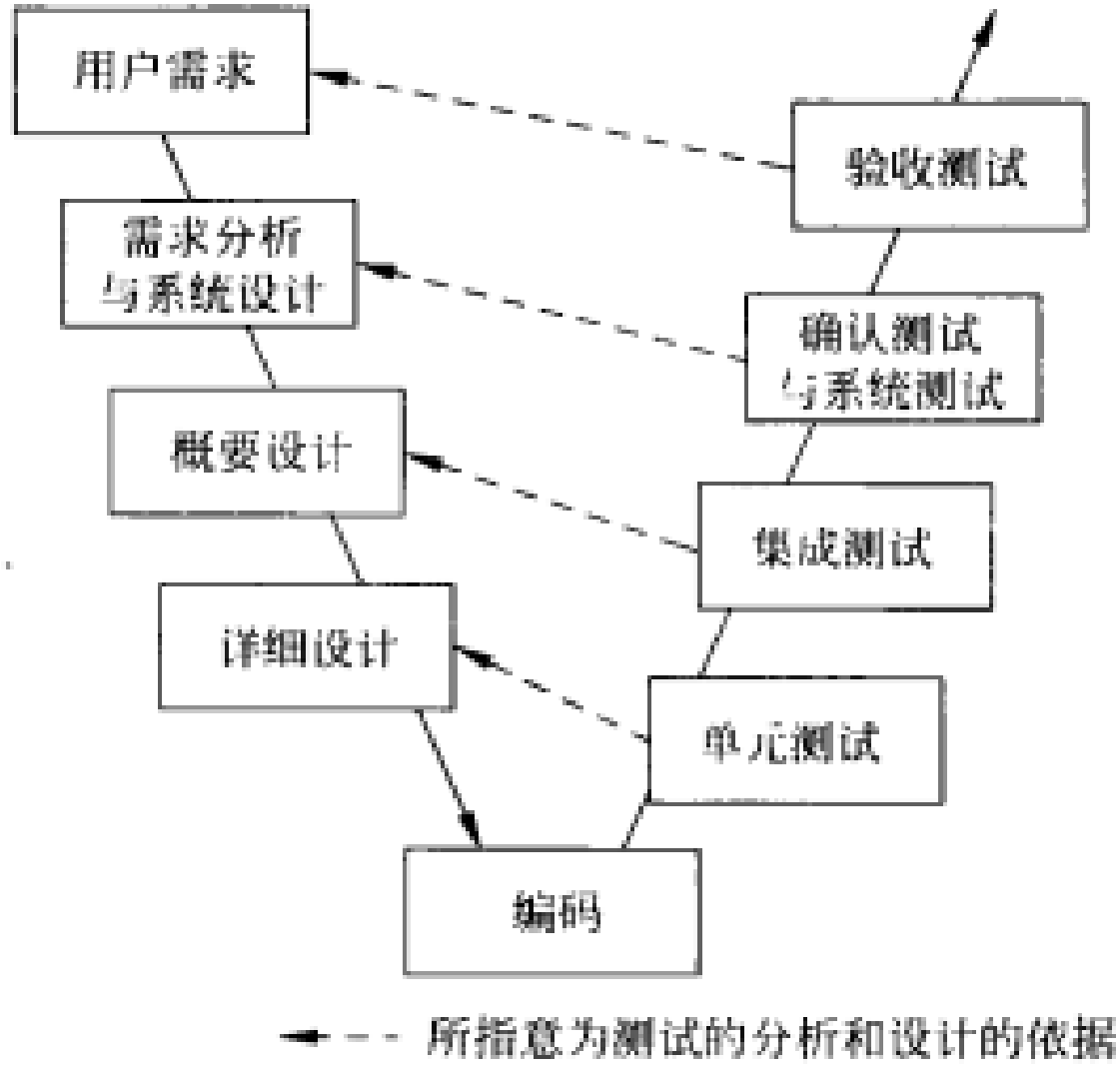


图 1-15　V 模型

- W模型

w模型是两个v的重叠，一个v表示开发过程，一个v表示测试过程。相比于V，W增加了软件各开发阶段中应同步进行的验证和确认活动。体现了尽早和不断进行软件测试的原则

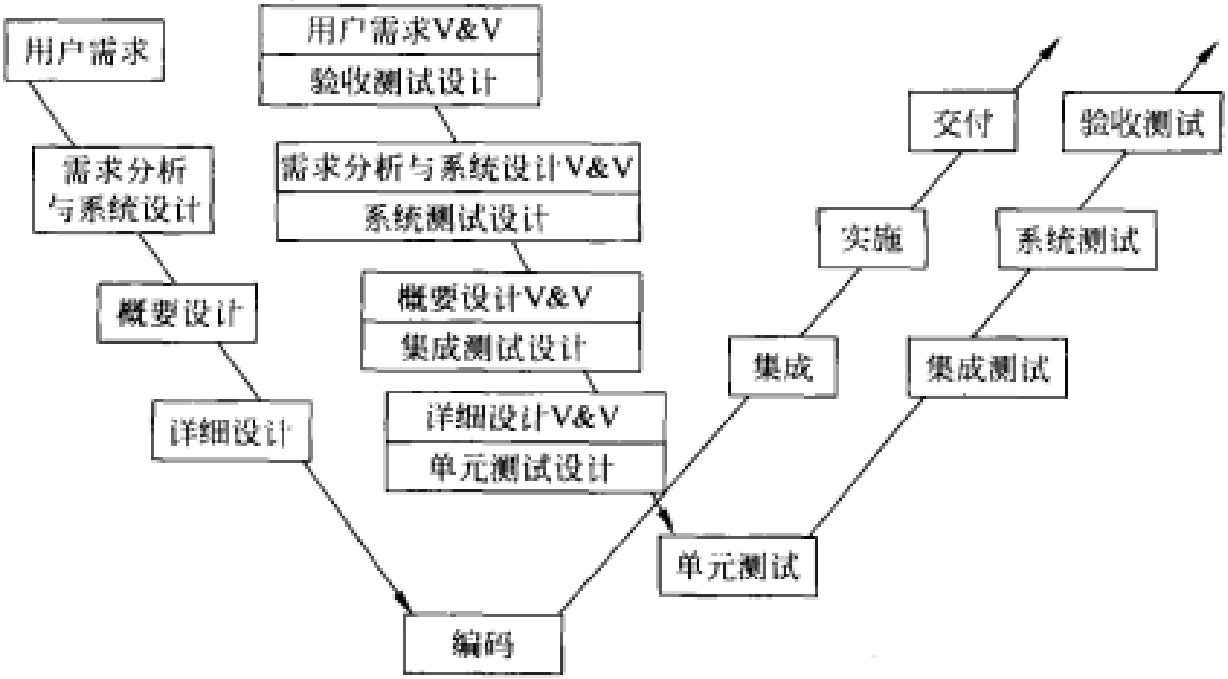


图 1-16　W 模型

- H模型

H 模型将软件测试看成一个独立的流程贯穿于软件产品的开发周期，当某个测试时间点就绪时，软件测试工作就可以开始。

V 模型和 W 模型均存在一些不妥之处。如前所述，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，这些活动在大部分时间内是可以交叉进行的，所以，相应的测试之间也不存在严格的次序关系。同时，各层次的测试（单元测试、集成测试、系统测试等）也存在反复触发、迭代的关系。为了解决以上问题，有专家提出了 H 模型。它将测试活动完全独立出来，形成了一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来，如图 1-17 所示。

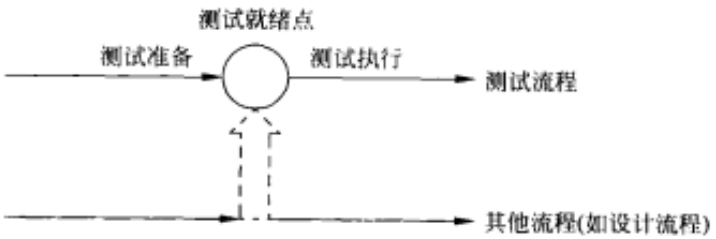


图 1-17　软件测试 H 模型

- 前置测试模型

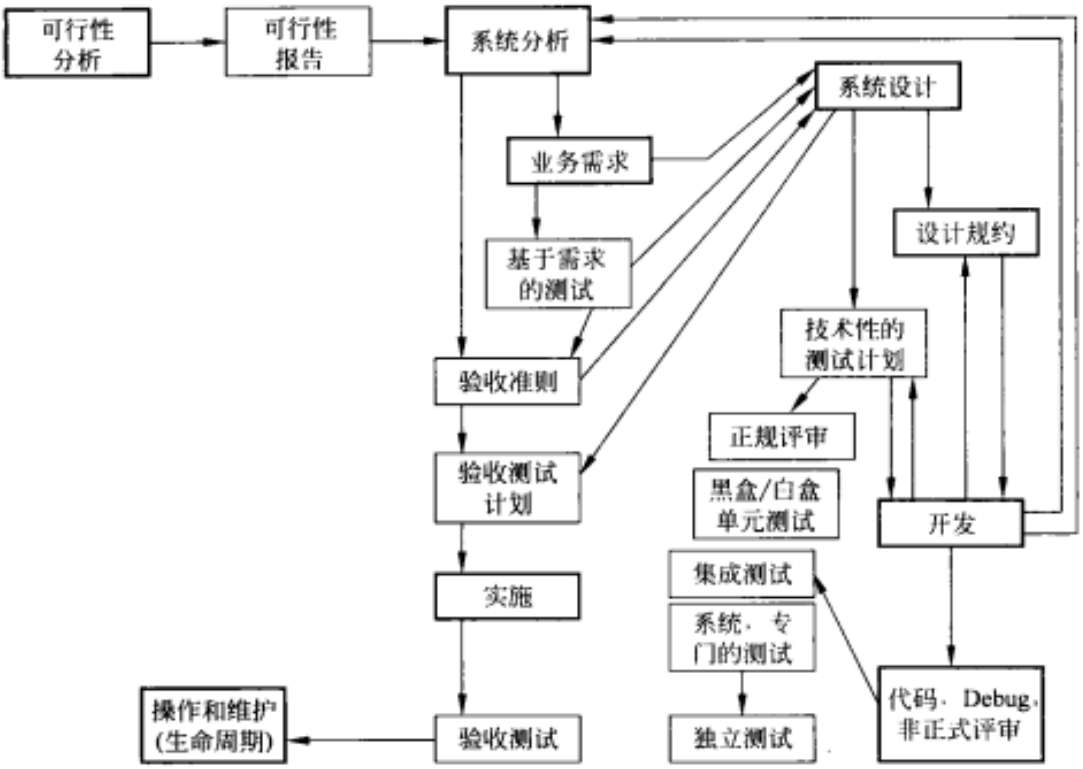


图 1-18　前置测试模型

软件测试的类型

功能性测试

（适用于单元、集成、系统测试）基于软件产品功能方面的测试

非功能性测试

（主要是系统测试阶段）

- 压力测试：用于检查系统在瞬间峰值负荷下正确执行的能力，目的是识别那些只有在这样不利条件下才出现的错误或缺陷。
- 性能测试：系统的性能是否满足需求规定的各种指标，如吞吐率或响应时间

软件结构性测试

白盒测试、基路径测试，了解程序内部逻辑结构

变更相关的测试（再测试和回归测试）

修复缺陷后要进行再测试确保缺陷被成功消除

系统变更可能会使原本工作正常的软件功能产生缺陷，所以要回归测试。回归测试适用于任何测试级别

回归测试集一般包括三种不同类型的测试用例：

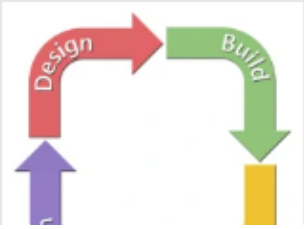
- 能测试软件所有功能的具有代表性的测试用例
- 专门针对可能因修改而影响某些软件功能的附加测试用例
- 针对修改过的软件部分的测试用例

测试概念名称	目标	范围	所有人	是否该持续集成	工具	备注
单元测试	验证源代码单元能够正确工作	class	开发人员	是	JUnit	应该产生100%的代码覆盖率报告
功能测试	面向代码，通过组件内示例数据测试特征	class	开发人员	是	JUnit	应该模拟依赖接口
集成测试	依赖一定的环境进行一些路径覆盖测试	components	开发人员	可选	JWebUnit TestNG	如果测试很快并且环境依赖可以很容易的自动安装，可以包含在持续集成中
冒烟测试	确保关键功能运行	application	开发人员	否	JWebUnit TestNG	代码提交给测试人员前运行，对于web-app模块等同于集成测试
回归测试	寻找回归的bugs，即只在新版本上出现的bugs。	application	测试人员	否	Selenium	重新运行之前版本运行过的测试
完备性测试	集中在修复的bugs和新特性上	application	测试人员	否	Selenium	为新特性增加新的测试用例
系统测试	针对功能需求规范（FRS）、系统需求规范（SRS）测试整个系统	application platform	测试人员	否	Selenium	可能包括更多测试，像可用性测试
平台测试	在不同的硬件和软件平台上运行	application platform	测试人员	否	Selenium	
性能测试	通过采集应用数据消除瓶颈	application platform	开发人员	否	JMon，ab，httperf，JMeter	
负载测试	暴露非表面bugs，确保达到性能指标	application platform	团队	否	ab，siege httpref	负载测试有时叫做大数据量测试，或者耐力测试
压力测试	通过耗尽资源或者删除资源来尽力破坏系统	application platform	测试人员	否	ab，httpref	也叫拒绝测试或者恢复测试
	需要的业务功能和正确的系					

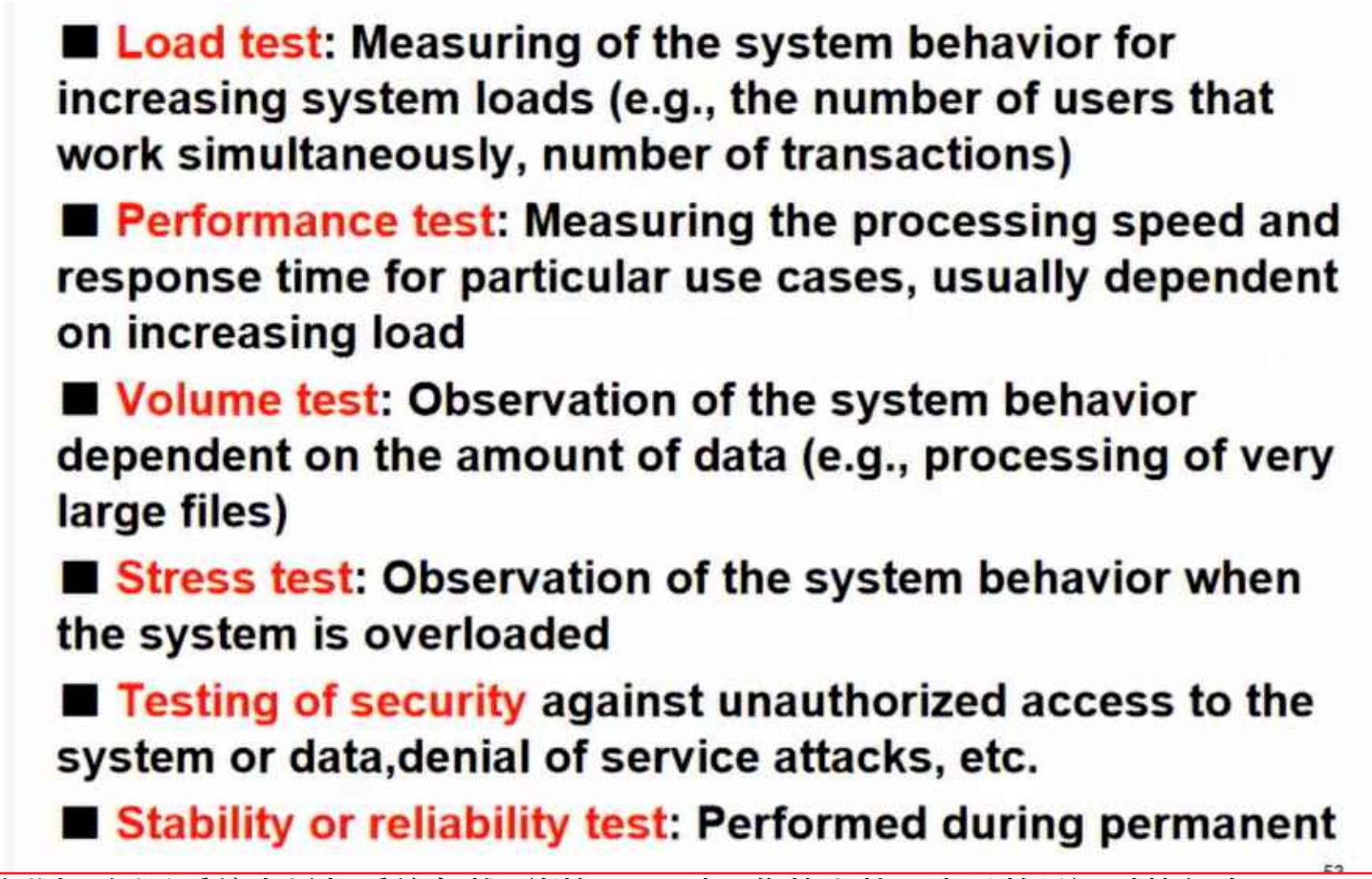
用户验证测试	统功能的一个最终验证，在类实际环境中进行	application platform	客户	否	客户决定	也叫拒绝测试或者恢复测试

[1]

词条图册



他说以下必问



负载测试：测量系统在增加系统负载（例如，同时工作的人数，交易数量）时的行为。
性能测试：测量特定用例的处理速度和响应时间，通常依赖于增加的负载。
容量测试：观察系统在数据量（例如，处理非常大的文件）依赖下的行为。
压力测试：观察系统在过载时的行为。
对系统或数据的未授权访问、拒绝服务攻击等的安全测试。
稳定性或可靠性测试：在持续运行期间执行。
鲁棒性测试：测量系统对操作错误、编程错误、硬件故障等的响应，以及异常处理和恢复的检查。
兼容性和数据转换测试：检查与现有系统的兼容性，数据的导入/导出等。
不同系统配置的测试：例如，不同的操作系统版本，用户界面语言，硬件平台等（并行测试）。
可用性测试：检查系统学习容易性，操作的便捷性和效率，可理解性。

■ **Robustness test**: Measuring the system's response to operating errors, bad programming, hardware failure, etc. as well as examination of exception handling and recovery

■ **Testing of compatibility and data conversion**: Examination of compatibility with existing systems, import/export of data, etc.

■ **Testing of different configurations of the system**: For example, different versions of the operating system, user interface language, hardware platform, etc. (back-to-back testing)

■ **Usability test**: Examination of the ease of learning the system, ease and efficiency of operation, understandability

软件测试的基本过程

软件测试的基本过程包括软件测试计划，测试分析和设计，测试实施，执行和监控，测试报告及测试活动结束等方面。

软件测试的基本过程包括软件测试计划，测试分析和设计，测试实施、执行和监控，测试报告及测试结束活动等方面。

软件测试计划一般开始于软件需求分析结束阶段，计划应该指明测试范围、方法、资源以及相应测试活动的时间进度安排表。具体应该包含测试目标、项目概述、组织形式、角色及职责、测试对象、测试通过和失败的标准（测试何时结束、度量的尺度如何、度量的评价标准等）、测试挂起的标准及恢复测试的必要条件、测试任务的安排、应交付的测试工作产品、工作量估计及退出测试的标准等。

退出测试的标准可以通过几个方面考虑，如计划中的测试用例是否执行完毕，是否达到功能、语句等计划的覆盖指标，继续测试发现缺陷的数量减少低于度量标准等。

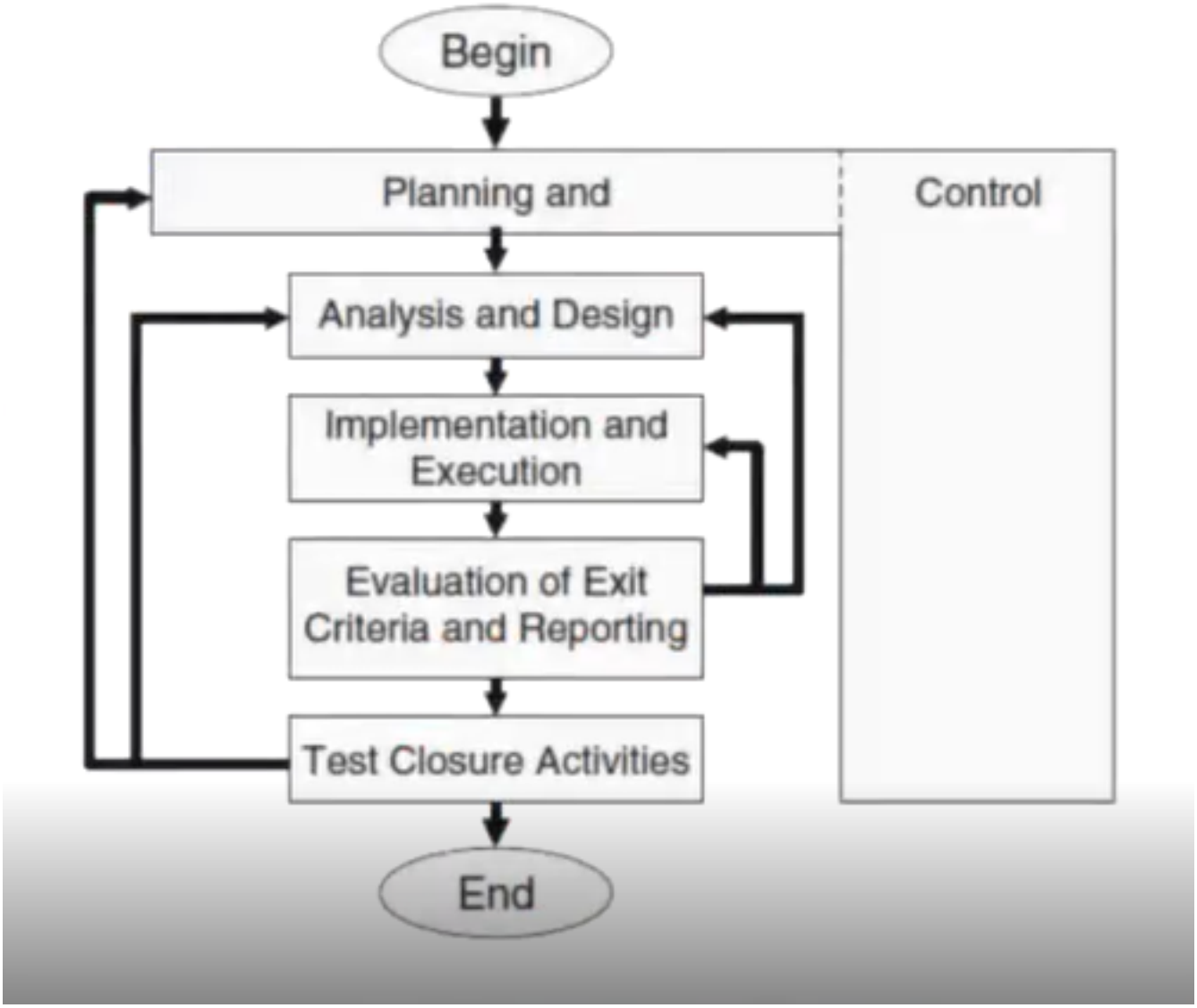
软件测试计划的制定不是不变的，在软件测试的过程中由于多种原因（计划本身的不完善、时间因素、技术因素等）可能导致对测试计划的调整，这也是测试控制的一个重要方面。

测试的分析和设计首先要有其分析和设计的依据，在软件测试的不同阶段测试分析的依据是不同的，在单元测试阶段分析的依据是详细设计规约，在集成测试阶段分析的依据是概要设计规约，在系统测试阶段分析的依据是需求分析规约，在验收测试阶段分析的依据是需求规约。在分析过程中规约不是唯一的依据，由于软件开发过程中的文档可能不是十分的完整，这就要求测试分析人员从其他方面进行分析作为补充，在分析的基础上再进行测试用例的设计。测试用例的设计可以以需求、分析和设计说明书为基础并结合测试方法和技术来进行，如从软件的流程图、功能点、状态图、use-case 图等方面并结合白盒和黑盒测试技术进行测试用例的设计。测试用例的设计应该遵循从逻辑测试用例到实际测试用例的设计过程，设计测试用例的多少或达到的覆盖指标应该充分考虑所测试的系统或子系统或模块的重要性，用例的设计同时要考虑用例执行应该具备的前提条件。

测试的实施和执行应该具备一些条件，如测试环境的具备、测试驱动器和测试桩的开发完成、测试模拟器及测试工具的准备等。在测试实施和执行过程中必须对测试结果进行记录，如记录用例运行是否通过、失败的原因及原因的分类、进一步测试的建议等。在测试的实施和执行过程中要进行必要的监控，如对测试的进度、用例执行的成功率、测试资源及测试人员的能力等进行监控，以便适时地调整计划。

测试结束活动主要涉及对测试结果的分析，如对测试发现的缺陷进行统计分类并分析其原因，制定的测试计划和实际的计划执行的差距并分析原因，在测试过程中哪些事件或风险没预料到，测试过程中好的经验总结等。

解释下面这张图



测试心理学

软件测试组的独立可能有以下几种存在方式：

- 测试分析和设计由开发人员自己完成
- 测试分析和设计由开发队伍其他人员完成
- 测试分析和设计独立于本项目开发团队
- 测试分析和设计独立于开发企业，即第三方机构

软件测试常被视为一种消极的活动，有以下几种方法改善测试人员与其他人员的关系：

- 采取合作方式，共识软件质量是共同目标
- 就事论事，非批评口吻
- 了解他人对缺陷的态度
- 共同确定问题存在性

静态测试技术

定义

静态测试：不要求在计算机上实际执行所测试的软件，主要以人工的模拟技术对软件进行分析和测试。有时候也被称白盒测试

相比于动态测试：

- 成本更低
- 效率较高
- 可以在软件开发生命早期阶段通过静态测试发现软件缺陷

评审

评审定义

由软件工作产品生产者的同行遵循已定义的规程对软件工作产品所做的审查，目的在于识别工作产品的错误或缺陷及需要改进之处。

好处：

更便宜的缺陷消除，缺陷容易更早被发现

开发时间更短

后续动态测试所用的开销更少

软件开发成本降低

故障率降低

改进工作方法，提高后续工作质量

要求：

评审的文件需要有一个正式的结构

评审要有一个清晰的目标

根据目标以及技能选择合适的评审人员

评审的文件属于评审团队而不是文件的作者，作者不应该为出现的问题辩护，评审团队也不应该将过错归咎于文件作者

分类：

- 非正式评审

    - 正在开发中
    - 不需要遵循明确定义
- 正式评审
    - 工作产品已经完成
    - 遵循一个已经明确定义
    - 参与评审人员有明确的职责与检查表

评审的可能问题：

所需的人员不可用或没有所需的资格或技术技能。

管理层在资源规划过程中估算不准确，可能会导致时间压力。

如果评论由于缺乏准备而失败，这主要是因为选择了错误的评论者。

审查也可能由于缺少或文件不足而失败。

评审过程

1. 计划阶段
2. 准备阶段
3. 自评审阶段
4. 评审会阶段
5. 重新修改阶段
6. 分析总结阶段

责任和角色

1. 协调负责人
2. 作者
3. 记录员
4. 评审者

代码检查

主要检查代码和设计的一致性，代码对标准的遵守、可读性、逻辑正确性、结构是否合理等。

人工检查过程中，会依据代码缺陷检查表来进行检查。

> 下边是java的例子的部分截图

表 2-1　Java 缺陷检查表

	重要性	检　查　项
命名	重要	命名规则是否与所采用的规范保持一致
	一般	是否遵循了最小长度最多信息原则
	重要	has/can/is 前缀的函数是否返回布尔型
注释	重要	注释是否较清晰且必要
	重要	复杂的分支流程是否已经被注释
	一般	距离较远的右大括号}是否已经被注释
	一般	非通用变量是否全部被注释
	重要	函数是否已经有文档注释(功能、输入、返回及其他可选)
	一般	特殊用法是否被注释
声明、空白、缩进	一般	每行是否只声明了一个变量(特别是那些可能出错的类型)
	重要	变量是否已经在定义的同时初始化
	重要	类属性是否都执行了初始化
	一般	代码段落是否被合适地以空行分隔
	一般	是否合理地使用了空格使程序更清晰
	一般	代码行长度是否在要求之内
	一般	折行是否恰当
	一般	包含复合语句的{}是否成对出现并符合规范
	一般	是否给单个的循环、条件语句也加了{}
	一般	if/if-else/if-else if-else/do-while/switch-case 语句的格式是否符合规范

代码检查类型

1. 桌面检查

程序员自己检查自己写的程序

2. 代码审查

一堆程序员和测试组成审查小组，一起阅读、讨论代码

步骤：

   a. 小组负责人提前把设计规约说明书、控制流程图、程序文本等相关要求发给组员，作为审查一句
   b. 开程序审查会（会上程序员逐字逐句解说自己的代码，别人提问）

3. 走查

   a. 发材料
   b. 开会（测试员准备用例，其余成员人力模拟）

代码检查内容

- 检查变量的交叉引用表
- 检查标号的交叉引用表
- 检查子程序、宏、函数
- 等价性检查
- 常量检查

- 标准检查
- 风格检查
- 补充文档
- 。。。。

编码规范

不说了，没意思

代码检查中的数据流与控制流

数据流错误：

读取未定义的值、变量被分配没有使用的值、变量第二次收到一个值，而第一个值未被使用。

控制流：

静态代码分析指标

ppt里只写了循环数，控制流图中画出的循环数越多，意味着之后测试的体量越大

正规技术评审

一组评审者按照规范步骤对软件需求、设计、代码或其他技术文进行仔细检查，找错误或缺陷。

评审小组

人数：>=3，一半[4,7]。概要性文档较多人评审，

1. 评审员

每个成员都叫评审员

2. 主持人

会议前负责正规技术评审计划和会前准备检查

会议中负责调动评审员热情

会议后负责问题分类，问题修改后的复核

3. 宣读员

朗读材料，（除了代码评审外）最好选择直接参与后续开发阶段的人员作为宣读员。

代码审查直接让作者读

4. 记录员

把发现的问题记录在”技术评审问题记录表“上

5. 作者

回答评审员提出的提问。如果真的有问题，必须得修改

技术评审活动过程

1. 计划

项目经理指定的主持人检查作者提交的材料是否齐全

主持人分发材料

2. 预备会

如果评审小组不熟悉评审材料和背景，主持人可以召开预备会

预备会上，作者介绍评审理由、被审材料的功能、用途、开发技术

3. 会前准备（自评审）

美味评审员根据检查要点逐行检查被审材料，对问题做好标注或记录。

主持人要了解评审员的准备情况

4. 评审会

宣读员读材料

评审员提问题

记录员把问题记录在”技术评审问题记录表“上

可以讨论问题，但如果讨一定时间无结果，主持人应当宣布该问题”未决“

评审会议结束时，全体评审员做出最后的评审结论

会议结束后，主持人对”技术评审问题记录表“上进行分类（1. 按问题种类 2. 按问题的严重性）

5. 修正错误

作者根据评审意见修正错误

6. 复审

被审材料问题过多、过复杂的花，主持人可以对修正后的材料再开一次评审会

7. 复核

主持人复核修真后的材料。

主持人完成”技术评审总结报告“

动态测试技术

黑盒测试

边界值分析法

对输入变量的定义域进行分析并设计测试用例

> 基本假设：单缺陷假设，即由于缺陷导致的程序失效极少是由两个（或多个）缺陷的同时作用引起

做法：

某一个变量取中间值，另一个取（正常值，最小值，较小值，最大值，较大值）

$$\{\langle x_{1nom}, x_{2min}\rangle, \langle x_{1nom}, x_{2min+}\rangle, \langle x_{1nom}, x_{2nom}\rangle, \langle x_{1nom}, x_{2max-}\rangle,$$
$$\langle x_{1nom}, x_{2max}\rangle, \langle x_{1min}, x_{2nom}\rangle, \langle x_{1min+}, x_{2nom}\rangle, \langle x_{1nom}, x_{2nom}\rangle,$$
$$\langle x_{1max\_}, x_{2nom}\rangle, \langle x_{1max}, x_{2nom}\rangle\}$$

因为存在两个<x_1norm,x_2norm>，因此2个变量只需要有9个测试用例

> 对n个变量，单变量假设情况下，存在4n+1个测试用例

如果确认的是值域y，则可以根据值域取值

> 基本边界值分析适用于边界量**相互独立**，且都是**物理量**

- 健壮性边界分析

在基本边界分析的五个取值上加上一个略超过最大值的值（max+）与略小于最小值的值（min-）

理论测试用例***6n+1***

- 最坏情况边界分析

即不满足**单缺陷**假设的情况，即程序的问题可能是由于多个变量值在其边界值附近取值共同引起的，而不是由单个变量在其边界值附近引起的。

> - 对于n个变量，会产生 5^n 个用例
> - 如果是健壮最坏情况边界分析，则会产生 7^n 个用例

原则

(1）如果输入条件规定了值的范围,则应取**刚达到**这个范围的边界的值,以及刚刚超越这个范围边界的值作为测试输入数据。

(2）如果输入变量规定了值的个数,则用最大个数、最小个数、比最小个数少1,比最大个数多1的数作为测试数据。

(3）边界值分析同样适用于输出变量,根据规格说明的每个输出条件,使用前面的原则(1）和(2)。

(4）如果程序的规格说明给出的输人域或输出域是有序集合,则应选取集合的第一个元素和最后一个元素来设计测试用例。

(5）如果程序中使用了一个内部数据结构,则应当选择这个内部数据结构边界上的值来设计测试用例。

(6）分析规格说明,找出其他可能的边界条件。

(7) 分析变量的独立性，以确定边界值分析法的合理性.

(8）在取中间值或正常值时,只要取接近取值范围中间的值就可以了。

(9)在取比最小值小的值时,根据情况可以取多个,可以取负值、0和小数。

(10)在取比最大值大的值时,根据情况可以取多个,当最大值非指定时,根据业务具体分析。

等价类测试法

希望达到的目的

1. 测试用例完善

2. 避免用力冗余

边界值分析法存在大量冗余与漏洞

边界值分析法没有考虑到同一个变量的多区间或多义性

基本思想

根据变量划分等价类，这些等价类构成不同的子集。这些子集的并为全集（保障**完备性**），子集互不相交（保障**无冗余**）

做法：

条件：

$$a \leqslant x_1 \leqslant e, \text{区间为} [a,b), [b,c), [c,d), [d,e]$$
$$f \leqslant x_2 \leqslant h, \text{区间为} [f,g), [g,h]$$

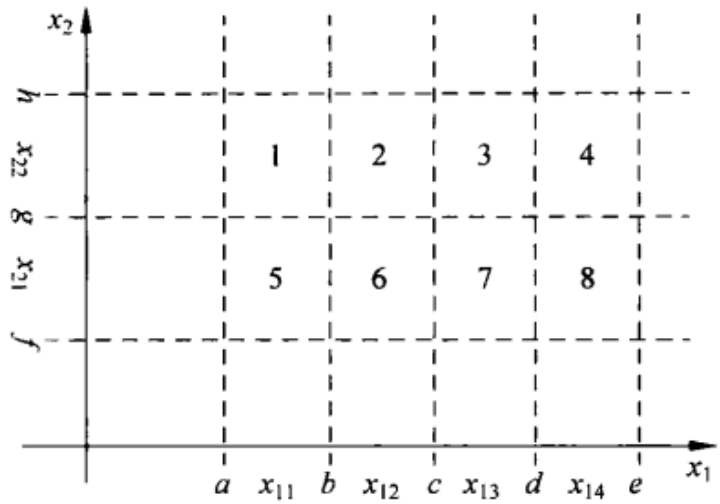


图 3-1　弱一般等价类测试用例分布

弱一般等价类

1-8为等价类

测试用例的个数

- 基于单缺陷假设
- 测试用例的个数是变量划分区间最多的那个变量的有效区间个数（x1方向有4个，x2方向有2个，应当选**max(4,2)=4**）
- 测试用例的选应该尽可能考虑均匀分布

如：1，6，3，8或5，2，7，4皆为较优的设计

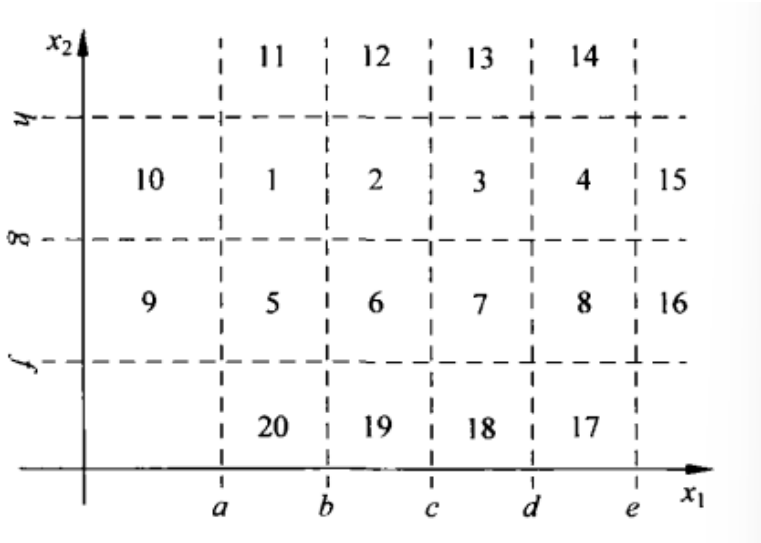
强一般等价类

- 基于多缺陷假设

测试用例应该覆盖 $A \times B$ 的区域，即全部1-8区域。计算方式为**2*4=8**

弱健壮等价类

- 有额外的测试用例



测试用构成：

1. 弱一般等价类部分测试用例
2. 额外弱健壮部分。

额外弱健壮部分的测试用例。对于n个变量而言,在这n个变量中每次取一个变量，分别取这个变量的所有**可能的无效值**和其他n—1个变量取有效值组合来构成测试用例的输入,保证如此取法涉及每个变量,即每个变量取一次。

如在{1，6，3，8}中取一个测试用例 + [9，20]区域内取一个测试用例

强健壮等价类

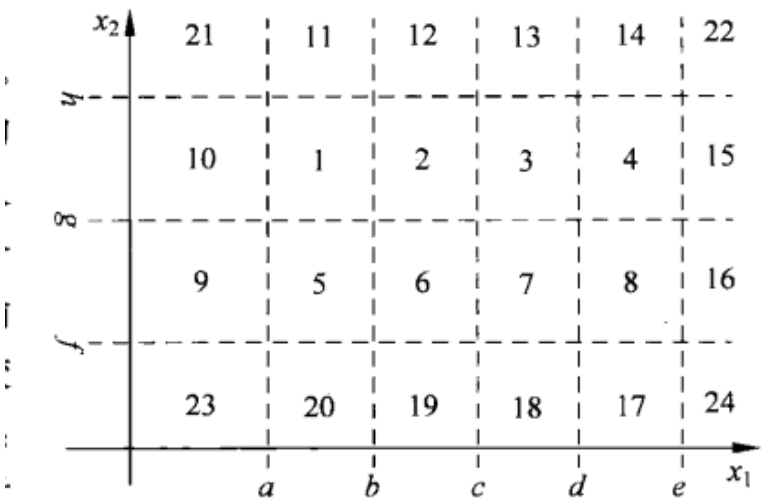


图 3-3　强健壮等价类测试用例分布

- 在上边的基础上进一步考虑多缺陷假设

做法：每个区间取值的笛卡尔积

即上图的1--23

设计原则

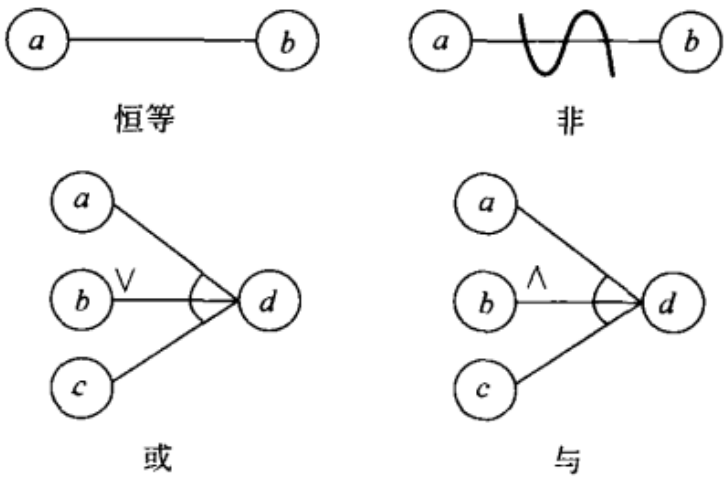
- 在输人或输出条件规定了取值范围或取值个数的情况下,可以确立--个有效等价类和两个无效等价类。
- 在输人或输出条件规定了输人或输出值的集合或者规定了“必须如何”的条件的情况下,可以确立一个有效等价类和一个无效等价类。
- 在输人或输出条件是一个布尔量的情况下,可确定一个有效等价类和一个无效等价类。
- 在规定了输人数据的一组值(假定n个),并且程序要对每一个输入值分别处理的情况下,可确立n个有效等价类和一个无效等价类。
- 在规定了输人数据必须遵守的规则的情况下,可确立一个有效等价类(符合规则)和若干个无效等价类(从不同角度违反规则)。
- 在确知已划分的等价类中,各元素在程序中的处理方式的不同,则应再将该等价类进一步地划分为更小的等价类。
- 一个输入条件或一个输出条件均可能划分成多个有效等价类和多个无效等价类。

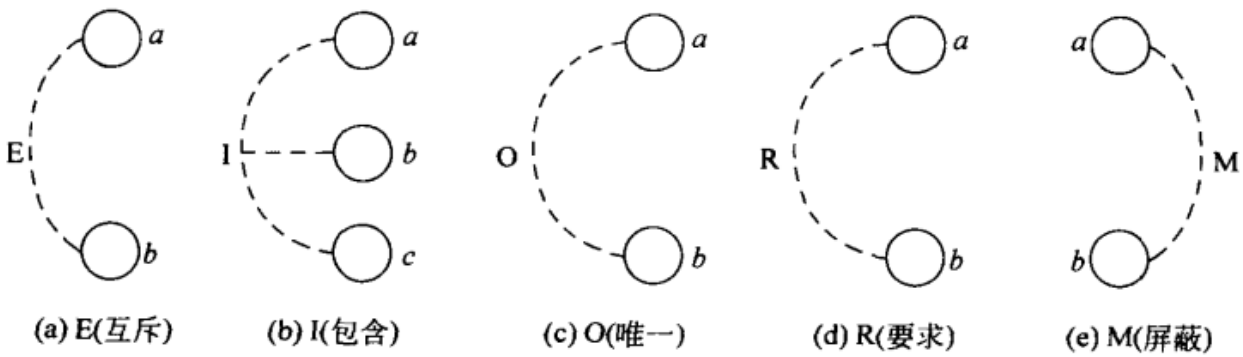
错误推测法

利用**经验和直觉**推测所有可能存在的错误或者缺陷，有针对地设计测试用例的方法。

因果图法

上述方法中，如果考虑条件之间的相互组合，则组合数量会非常大。可以使用因果图法解决该情况。





根据因果图画判定表。把所有原因作为输入条件，每一项原因安排为一行，而把所有的输出条件的组合一一列出（true为1，false为1），对于每种条件组合安排为一列。

如果原因有4项，则有4行，2^4=16列

例子：

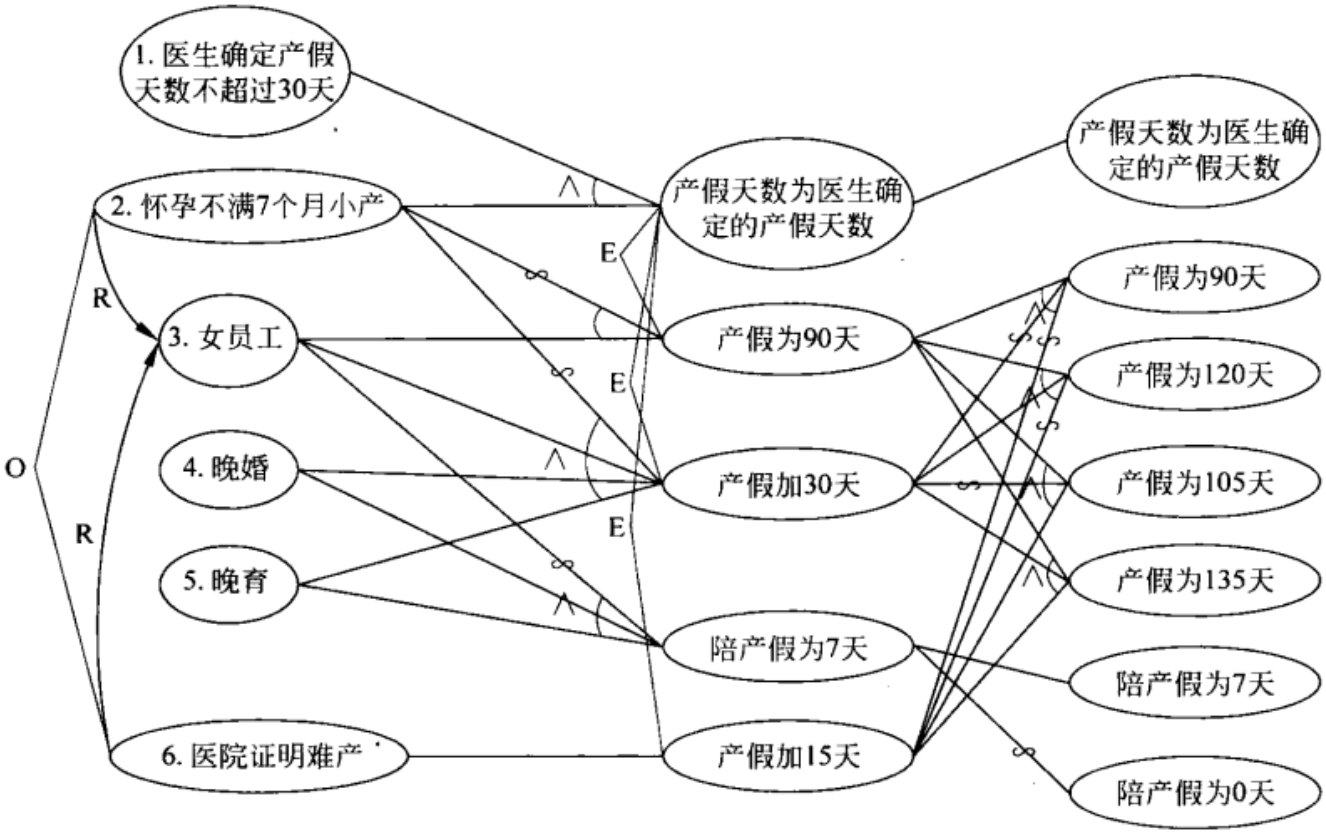


图 3-6　实例因果图

条件	女员工	1	1	1	1	1	1	1	1	1	0	0	0	0
	晚婚	/	1	1	0	0	1	1	0	0	1	1	0	0
	晚育	/	1	0	1	0	1	0	1	0	1	0	1	0
	怀孕不满 7 个月小产	1	0	0	0	0	0	0	0	0	/	/	/	/
	医院证明难产	0	1	1	1	1	0	0	0	0	/	/	/	/
	医生确定小产后的产假天数＜＝30 天	1	/	/	/	/	0	0	0	0	/	/	/	/
中间结果	普通产假天数(90)	0	1	1	1	1	1	1	1	1	0	0	0	0
	小产产假天数(医生确定天数)	1	0	0	0	0	0	0	0	0	0	0	0	0
	难产产假天数(15)	0	1	1	1	1	0	0	0	0	0	0	0	0
	晚婚晚育产假天数(30)	0	1	0	0	0	1	0	0	0	0	0	0	0
	陪产假天数(7)	0	0	0	0	0	0	0	0	0	1	0	0	0
结果	假期	90	135	105	105	105	120	90	90	90	7	0	0	0

- 可以清晰归纳条件之间的限制关系，直接将一些组合忽略掉

决策表测试法

- 与因果法有密切关联
- 是测试方法中最严格的
- 设计用例的过程比较麻烦
- 每列对应一个测试用例

- **条件桩(condition stub)：列出了问题的所有条件。通常认为列出条件的先后次序无关紧要。**
- **行动桩(action stub)：列出了所有可能采取的操作。这些操作之间的排列先后顺序没有约束。**
- **条件条目(condition item)：列出针对各条件桩的所有可能取值，这些值可能为真假值或其他取值。**
- **行动条目(action item)：列出在条件条目下的各种取值情况应该采取的动作或操作。**
- **规则：任何一个条件组合的特定取值及其相应要执行的操作。**

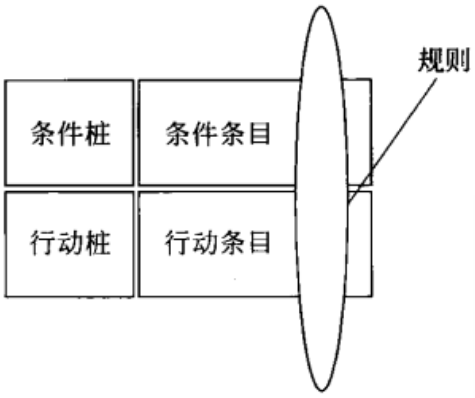


图 3-7　决策表组成示意图

- X代表行动桩对应的操作有效
- n/a代表不关心条目
- 下图规则1表示如果c1,c2,c3为T，则采取行动a1,a2
- 输入T/F为二叉条件决策表，也可以输入多个值（甚至等价类），叫做”扩展条目决策表“

表 3-7　决策表例子

桩	规则 1	规则 2	规则 3、4	规则 5	规则 6	规则 7、8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	—	T	F	—
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

表 3-8　三角形问题决策表

c1：a、b、c 构成三角形？	N	Y	Y	Y	Y	Y	Y	Y	Y
c2：a=b?	—	Y	Y	Y	Y	N	N	N	N
c3：a=c?	—	Y	Y	N	N	Y	Y	N	N
c4：b=c?	—	Y	N	Y	N	Y	N	Y	N
a1：非三角形	X								
a2：不等边三角形									X
a3：等腰三角形					X		X	X	
a4：等边三角形		X							
a5：不可能			X	X		X			

应当避免冗余与不一致

下图中1-4列与9列是冗余的

表 3-10　一个冗余的决策表

条件	1～4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	X
a2	—	X	X	X	—	—
a3	X	—	X	X	X	X

下图中1-4列与9列是不一致的

表 3-11　一个不一致的决策表

条件	1～4	5	6	7	8	9
c1	T	F	F	F	F	T
c2	—	T	T	F	F	F
c3	—	T	F	T	F	F
a1	X	X	X	—	—	—
a2	—	X	X	X	—	X
a3	X	—	X	X	X	—

步骤

(1）根据规约分析条件个数和条件的取值,决定用有限条目的决策表还是用扩展条目的决策表。

(2)分析理论规则的个数。假如用有限条目的决策表设计,每个条件取“真”和“假”两个值,那么对于n个条件的决策表规则数为2^n个；假如用扩展条目的决策表设计,规则的个数可以根据不同变量划分的等价类个数的积及结合其他方法来确定。

(3）列出所有可能的行动桩。

(4）列出所有的条件条目和行动条目,并考虑“不可能”条目和“不关心”条目。

(5)根据规则完成测试用例的设计。

(6）评审决策表和测试用例集。

场景法（use case法）

- 测试流程按照一定的时间流正确地实现某个软件功能的时候，这个流被称为该软件功能的基本流
- 出现故障或缺陷或例外的流程，被称为备选流
- 用例场景指从用例开始到结束遍历这条路径上所有基本流和备选流

场景 1：基本流；
场景 2：基本流、备选流 1；基本流；
场景 3：基本流、备选流 1、备选流 2；
场景 4：基本流、备选流 3；基本流；
场景 5：基本流、备选流 3、备选流 1；基本流；
场景 6：基本流、备选流 3、备选流 1、备选流 2；
场景 7：基本流、备选流 4；
场景 8：基本流、备选流 3、备选流 4；
场景 9：基本流、备选流 3；基本流、场景 3、基本流、场景 3、基本流。

图 3-8

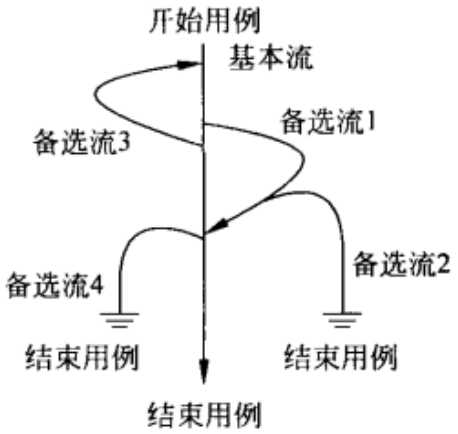


图 3-8　基本流和备选流

设计步骤

(1）根据说明书或规约,分析出系统或程序功能的基本流及所有可能的备选流。

(2)根据基本流和各项备选流设计不同的场景

(3)对每个场景生成相应的逻辑测试用例。

(4)根据逻辑浏试用例设计实际(物理）测试用例。

(5）对生成的测试用例集进行评审,基本的覆盖指标是**基本流和所有的备选流**在所设计的场景中**至少覆盖一次**。

正交实验法

根据正交表设计用例

$$L_{18}(3^7)$$

含义：

7个变量（因子），每个变量3种取值（每个因子取3个水平），按照正交表需要进行18次实验

$$L_{行数}(水平数^{因素数})$$

步骤：

(1）确定因素(变量);

(2）确定每个因素有几个状态(水平)(变量的取值)

(3）选择一个合适的正交表;

(4）把变量的值映射到表中;

(5）把每一行的各因素水平的组合作为一个测试用例

(7）评审测试用例集。

> 如果发现根据条件所得到的水平数，因素数组合找不到对应的表，可以采用包含的方式实现（选择方案水平数>=所有对应水平数，选择方案因素数>=所有对应因素数）

如：自己的组合

$$(3^2 \times 4^1 \times 2^1)$$

可以选择：

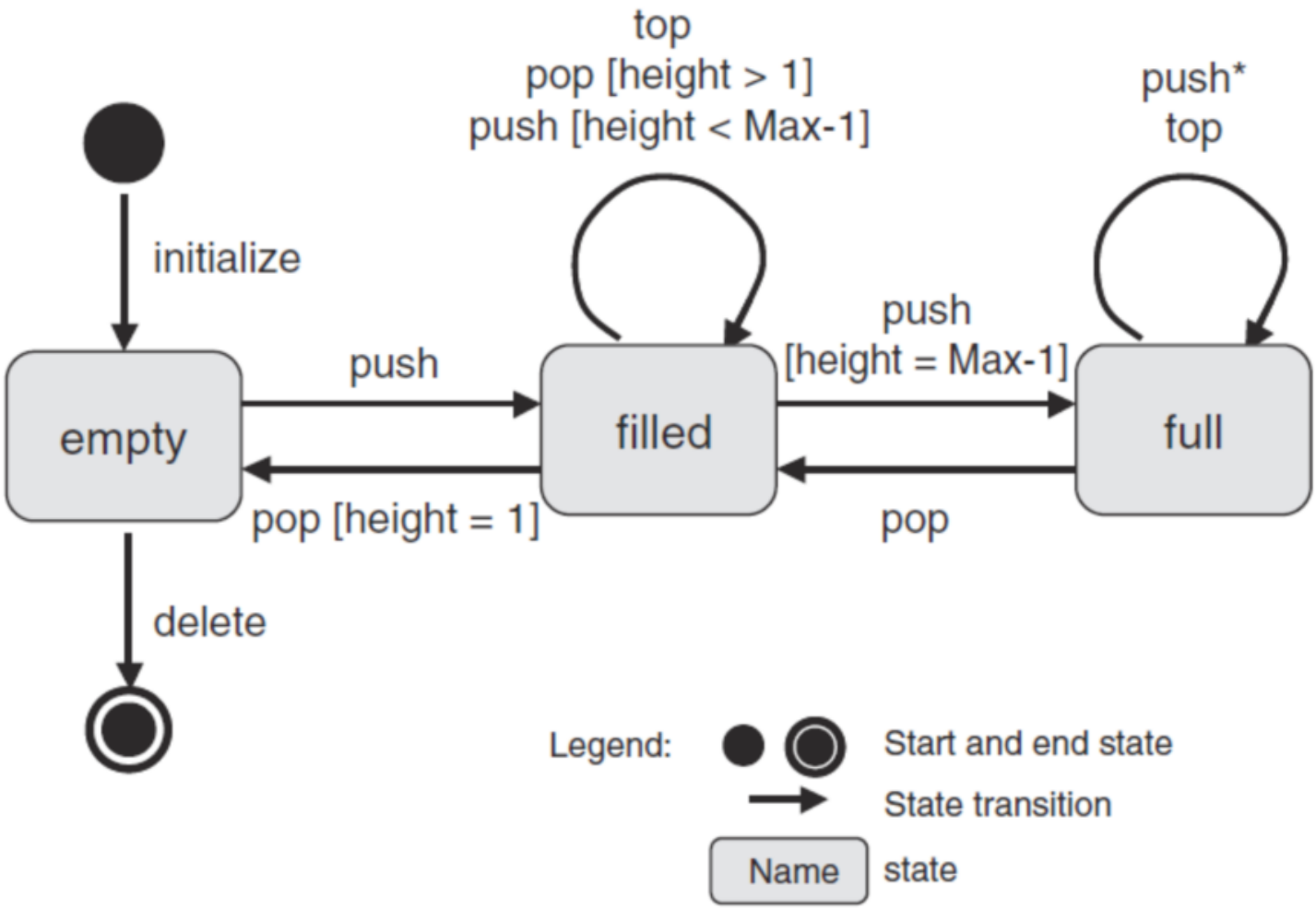
① 表中的因素数≥4。

② 表中至少有 4 个因素的水平数≥2。

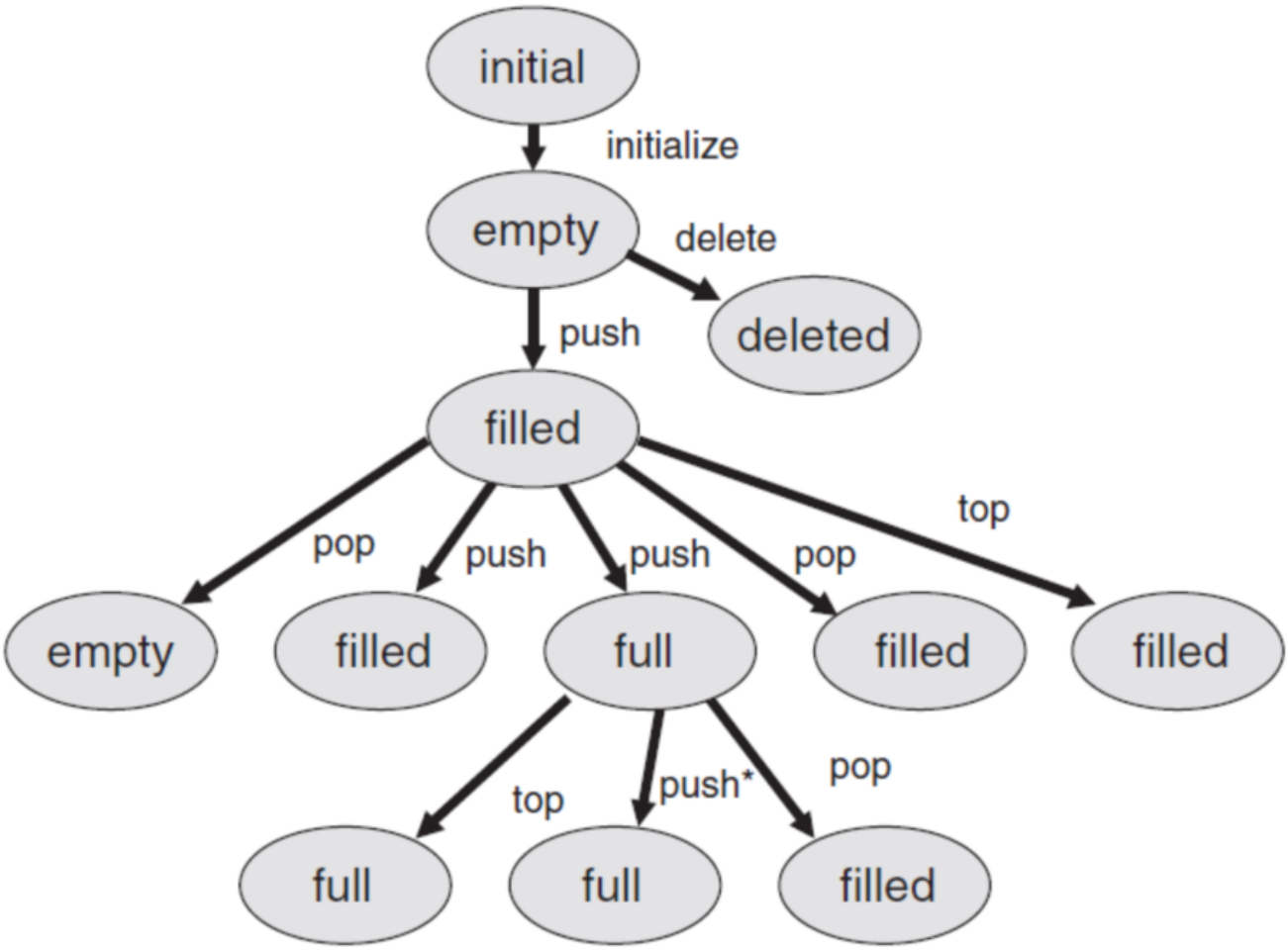
③ 行数取最少的一个，即满足$(3^2 \times 4^1 \times 2^1)$的最少行数 $2\times(3-1)+1\times(4-1)+1\times(2-1)+1=9$。

最后选中正交表公式 $L_{16}(4^5)$，对应的正交矩阵如表 3-23 所示。

State Transition Testing



利用状态图构造状态转移树



从根部到树尾部就是一个test case

白盒测试

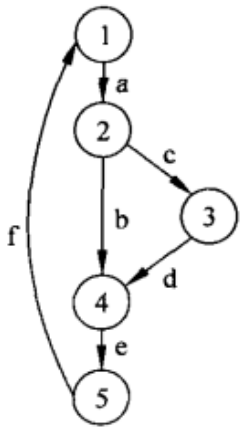
概念

1. 白盒测试是一种可视的测试软件的方法
2. 测试人员要了解程序结构和处理过程，按照程序内部逻辑测试程序，检查程序中的每条通路是否按照预定要求正确工作
3. 主要针对被测程序的源代码
4. 白盒测试能解决程序中的逻辑错误和不正确假设
5. 主要用于单元测试、集成测试及回归测试，也可用于系统级别的测试
6. 白盒测试，常常需要开发桩模块和驱动模块
    a. 桩模块：能够代替被测模块所调用的软件模块的程序
    b. 驱动模块：用于触发被测模块，一般要提供测试输入、控制和监测并报告测试结果。
        i. 驱动模块会被主方法调用

程序结构分析

1. 控制流分析

- 控制流图是对流程图的简化
- 由节点和控制流线（/弧线）组成

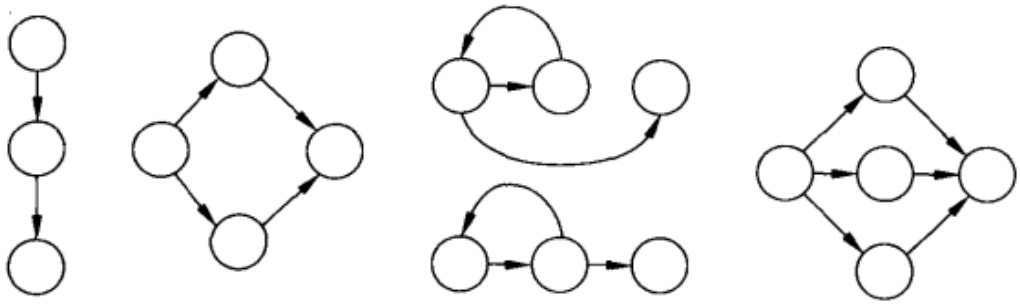


(b) 控制流图(程序图)

图和控制流图

	a			
		c	b	
			d	
				e
f				

图 3-14　控制流图矩阵



(a) 顺序结构　(b) IF选择结构　(c) WHILE循环结构　UNTIL循环结构　(d) CASE多分支结构

图 3-15　程序图的图形符号

程序结构的基本要求：

程序中不应该包含：

(1)转向并不存在的标号

(2）没有用的语句标号

(3)从程序入口进入后无法达到的语句

(4）不能达到程序出口语句的语句。

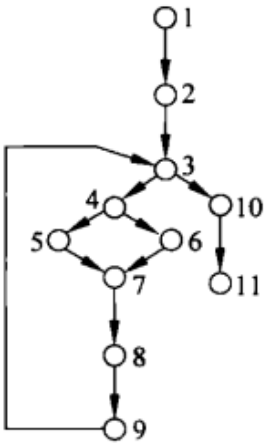
2. 数据流分析

- 用于查找如引用未定义变量、未使用变量重新赋值等错误
- 可用于常数传播

```
a := 4
b := a +
⋮
c := 3 * (a + b)
```

可以用下列程序段代替：

```
a := 4
b := 5
⋮
c :  = 27
```



节点	被定义变量	被引用变量
1	X,Y,Z	
2	X	W,X
3		X,Y
4		Y,Z
5	Y	V,Y
6	Z	V,Z
7	V	X
8	W	Y
9	Z	V
10	Z	Z
11		Z

图 3-18　控制流图及其定义和引用的变量

3. 信息流分析

- 主要用于验证程序变量间信息的传输遵循的保密要求

```
语句号    begin
  1          Q := 0;
  2          R := M;
  3          while R >= N do
               begin
  4                Q := Q + 1;
  5                R := R - N
               end
          end
```

图 3-19　整除算法

	M	N
1		
2	√	
3	√	√
4	√	√
5	√	√

(a) 第一个关系

	Q	R
1	√	
2	√	√
3	√	√
4	√	
5	√	√

(b) 第二个关系

	Q	R
M	√	√
N	√	√

(c) 检查

图 3-20　整除算法中输入值、语句与输出值的关系

第一个关系：执行语句所输入的变量

第二个关系：执行可能直接或间接影响输出变量终值的一些语句

第三个关系：提供一些检查

> 总体感觉不太重要，挺奇怪的内容

逻辑覆盖测试法

```
IF (( A > 1) AND ( B = 0 ))THEN
        X = X/A
IF (( A = 2) OR ( X > 1 ) THEN
        X = X + 1
```

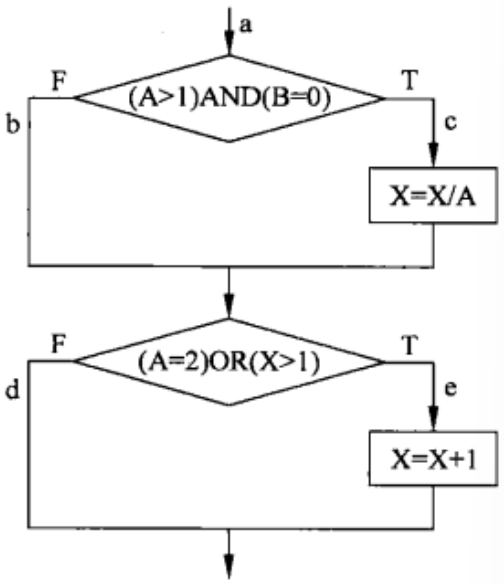


图 3-21　被测程序段流程图

语句覆盖

追求尽量覆盖所有语句

例如：输入A=2，B=0，C=3。即刻覆盖四个语句

- 也许会发现即使条件判断写错（比如and写成了or），也依然覆盖全语句，导致检查不出错
- 是一种比较弱的覆盖原则，没有排除程序包含的错误风险

判定（判断）覆盖

使程序中每个判断的取真分支和取假分支至少经历一次

可以覆盖全语句，

但需要更强的逻辑覆盖判断内部条件的错误

例如if(a && b)，整句if(a && b)只需要出现一次true与一次false

条件覆盖

设计若干测试用例，执行被测试程序以后，要使每个判断中每个条件的可能取值至少被满足一次

例如if(a && b)，a为true，a为false，b为true以及b为false都要满足一次

> - 满足了条件覆盖不一定满足判定覆盖

判定--条件覆盖

- 所有条件的所有可能必须满足一次
- 每个判断的判定结果必须出现一次

缺点：

例如判定语句 (a or b)，a为true。此时不会去检查b，则就无法查出逻辑表达式中隐藏的错误

条件组合覆盖

设计足够的测试用例，使得每个判断的所有可能条件取值组合至少执行一次

```
IF (( A > 1) AND ( B = 0 ))THEN
        X = X/A
IF (( A = 2) OR ( X > 1 ) THEN
        X = X + 1
```

例子中，存在4个条件以及两个判断，由8种组合，即S

(1) A>1,B=0,记为 T1,T2;
(2) A>1,B≠0,记为 T1,F2;
(3) A≤1,B=0,记为 F1,T2;
(4) A≤1,B≠0,记为 F1,F2;
(5) A=2,X>1,记为 T3,T4;
(6) A=2,X≤1,记为 T3,F4;
(7) A≠2,X>1,记为 F3,T4;
(8) A≠2,X≤1,记为 F3,F4。

- 但是没有走遍所有的路径

路径覆盖

设计足够测试用例，覆盖程序中所有可能的路径

- 实际测试中路径数量往往非常庞大，难以完全覆盖，应当尽可能对路径数量进行限制
- 即使路径覆盖了，也可能出bug

合理压缩路径数量的方法

1. 基路径计算法

空间向量的基：

对于一个向量空间Vn，找到一组y1,y2,...,y

p。各个yi之间相互独立，且Vn中的任意向量都可以由这组y线性表示。

算法步骤：

（1）根据程序图,利用公式Vg=e-n+2得到独立路径数。

（2）根据程序图找到一条基线路径,这条基线路径不唯一。基线路径应该满足:是从源节点开始到汇节点结束的路径；尽量长；尽量多地经过出度大于或等于2的节点；基线路径对应的业务最好是执行正常的业务流。

（3）以这条基线路径为基础,从这条基线路径的第一个节点开始,从头往后搜索以找到第一个出度大于或等于2的节点(包括第一个节点本身),以这个节点为轴进行旋转,将这个节点在基线路径中的儿子节点和不在基线路径中的其他儿子节点互换,如果不在基线路径中的其他儿子节点数大于一个,则任意选择一个即可,同时,尽量多地保留基线路径中的其他节点不变(称为回溯),这样就得到了另一条路径。

（4）下一步一般情况下仍然是以基线路径为基础,从刚才找到的出度大于或等于的节点开始往后继续寻找,执行③的同样流程得到其他路径。

（5）如果基线路径中的出度大于或等于2的节点全部找到,并根据③的流程得到了其他所有可能的路径,这时路径总数仍然没有达到Vg的值,则可以把以基线路径旋转得到的路径看成另一个基线路径,同样执行③一直到得到的路径数量等于Vc为止。

图 3-22 所示是一程序图，将根据基路径算法得出一组基路径。

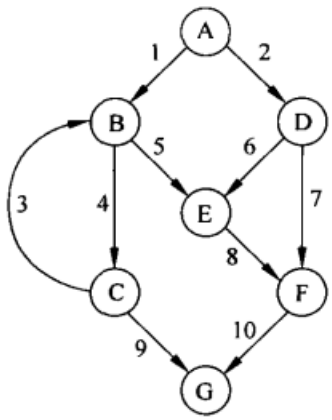


图 3-22　程序图

首先，依据 $V_G=e-n+2=10-7+2=5$，基路径中应该包含 5 条独立的路径。根据必须确定一条基线路径的原则，依据基路径的算法选择基线路径 A→B→C→B→E→F→G。这条基线路径中的第一个节点 A 就是出度等于 2 的节点。以节点 A 为轴进行旋转，以 D 来替换 A，基线路径中的其他节点尽量多的保留，这样得到了第二条路径 A→D→E→F→G。再以基线路径中的节点 B 为轴旋转，用节点 E 来替换节点 C，同样，基线路径中的其他节点尽量多地保留，这样得到第三条路径 A→B→E→F→G。再搜索基线路径发现第三个节点 C 也是出度等于 2 的节点，以 C 节点为轴旋转，用 G 来替换 B，得到第四条路径 A→B→C→G。再往后搜索基线路径发现节点 B 已经被旋转过，继续往后搜索找不到没有旋转过的出度大于或等于 2 的节点了，这时，可以在刚才旋转得到的其他路径中找出度大于或等于 2 的节点，这里可以在第二条路径 A→D→E→F→G 中搜索，发现节点 D 没被旋转过，同时该节点的出度为 2，以 D 为轴进行旋转，用节点 F 替换节点 E，得到路径 A→D→F→G。这样就得到了 5 条路径，这 5 条路径是独立的，构成了图 3-22 所示程序图的一组基路径：

- A→B→C→B→E→F→G；
- A→D→E→F→G；
- A→B→E→F→G；
- A→B→C→G；
- A→D→E→F→G。

最少用例数计算

基本控制结构：

1. 顺序型：构成串行操作
2. 选择型：构成分支操作
3. 重复型：构成循环操作

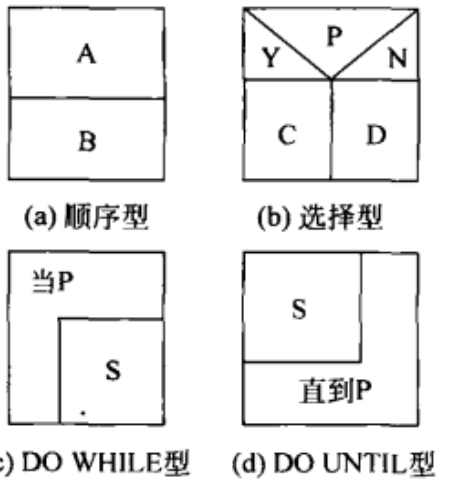


图 3-25　N-S 图表示的基本控制结构

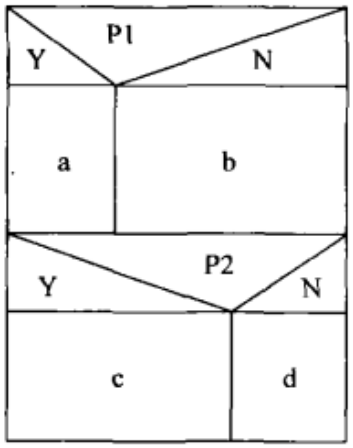
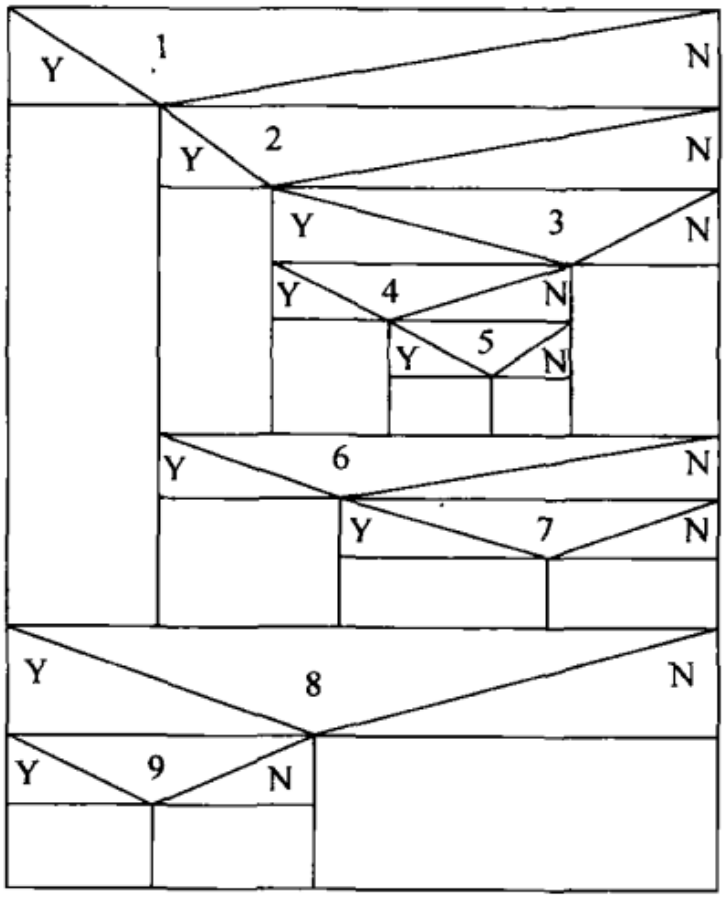


图 3-26　两个串行的分支结构的 N-S 图

> 为了简化问题，使用选择结构代替重复结构

图3-26中，第一个分支P1有2个操作，P2同理。由于两者并列，测试用例总数**至少**为2*2=4

下图中，测试用例**至少**为（（5*3）+1）*3=48



测试覆盖标准

先前介绍的逻辑覆盖无法做到很好的检测，foster通过大量实验订出了一套estca规则

- 规则1:对于A rel B(rel可以是<,=和>)型的分支谓词,应适当地选择A与B的值,使得测试执行到该分支语句时,A≤B，A一B和A>B的情况分别出现一次。
- 规则2:对于A rell C(rell可以是>或<,A是变量,C是常量)型的分支谓词,当rell为<时,应适当地选择A的值,使A=C一M(M是距C最小的容器容许正数,若A和C均为整型时,M = 1)。同样,当rell为≥时,应适当地选择A,使A=C+M。
- 规则3:对外部输入变量赋值,使其在每一测试用例中均有不同的值与符号,并与同一组测试用例中其他变量的值与符号不一致。

单元测试

程序单元通常指程序中定义的函数或子程序，有时也可把紧密相关的一组函数或过程看成一个单元，在面向对象系统中，一般以类为测试单元

测试驱动开发：先写测试程序，再进行功能开发

单元测试是集成测试和其他测试的基础

一般采用黑盒、白盒和静态测试

测试环境：

- 驱动模块：所测单元的主程序，接收测试输入，驱动被测单元执行，接收执行结果并进行分析和判断
- 桩模块：代替所测模块调用的子模块，需要实现代替子模块的部分功能（一次性的）
- 测试工具、网络、外部设备等其他因素

传统结构化开发单元测试策略：

- 自顶向下：以单元组件的层级关系为依据自顶向下测试（一棵树），组件更改要对其下的所有单元重新测试。驱动模块和桩模块都是需要的，可以根据下层单元设计桩模块。
- 自底向上的单元测试：驱动模块和桩模块都是需要的，可以根据上层单元设计驱动模块。
- 孤立测试：互不相干，每个单元独自开发驱动和桩，简单易用
- 综合测试：结合使用以上三种

面向对象开发单元测试策略

一般类测试

特殊类测试

测试过程

单元测试的主要过程如下：
(1) 详细设计说明书(规约)通过评审。

级软件测试技术

(2) 编制单元测试计划(测试经理)。
(3) 编制子系统单元测试计划(如果需要的话)(开发组)。
(4) 编写测试代码并开发单元测试用例(开发组)。
(5) 代码审查(开发组或测试组)。
(6) 测试用例评审(开发组或测试组)。
(7) 测试执行(开发组)。
(8) 缺陷提交(开发组)。
(9) 缺陷跟踪(开发组或测试组)。
(10) 测试报告及评审(开发组或测试组)(未通过回到第(4)步)。

集成测试

集成测试也叫组装测试、联合测试，是在单元测试的基础上将所有模块按照概要设计的要求组装成为子系统或系统的测试。是对模块间接口或系统的接口以及集成后的子系统或系统的功能进行正确性检验的一项测试工作。

一般采用黑盒测试，但随着技术的发展也常常使用黑白交加的方法

传统结构化开发的4个集成测试层次：

1. 模块内集成
2. 子系统内集成
3. 子系统间集成
4. 不同系统间集成

面向对象技术开发的4个集成测试层次：

1. 类内集成
2. 类间集成
3. 子系统间集成
4. 不同系统间集成

集成测试环境

- 硬件环境
- 操作系统环境
- 数据库环境
- 网络环境
- 测试工具环境
- 开发驱动器和桩
- 其他环境（如web服务器）

传统开发集成测试方法

- 基于分解的集成
  - 非增量式集成：按照结构组装好一起测，也叫大爆炸集成
  - 增量式集成：边组装边测，又可细分为自顶向下和自底向上（和单元测试思路一样）
- 三明治集成：综合了增量式集成和大爆炸集成，对于系统结构，先确定一个中间层，对于中间层上方的结构采用自底向上方法，对于中间层下面的部分采用自顶向下法。这种方法结合了两者的特点，不需要开发太多的驱动和桩。
- 基于调用图的集成

  成对集成

  相邻集成
- 基于路径的集成
- 其他集成测试方法（如分层集成、高频集成）

面向对象开发集成测试方法

由于面向对象程序具有动态特性，一般只能整个编译后做基于黑盒子的集成测试

测试过程

和标准测试过程类似（计划、用例分析和设计、实施、执行、分析评估）

接口的错误有哪几种

- 传输错误（传输语法错误或没有数据导致组件无法操作或崩溃），组件功能故障，接口格式不兼容，协议故障
- 通信正常，但相关组件对数据的解释不同
- 数据传输正确，但时间不符合要求，比如传的太慢

系统测试

系统测试是在整个系统投入运行前，对系统的各元素进行组装和确认，确保在系统实际运行时软件、硬件、外设、网络等系统元素能够相互配合。

系统测试除了验证系统功能外，还会涉及安全性、性能压力、可用性、可靠性、可恢复性等方面的测试。

系统测试属于黑盒测试范畴。

系统测试环境：

计算机数量、及其配置、外设、操作系统、服务器、数据库、中间件等等

测试过程：标准测试5步骤（计划、用例分析和设计、实施、执行、分析评估）

验收测试

验收测试是系统级别的测试，但是是客户进行的测试或客户参与的测试

阿尔法测试：软件开发公司组织内部人员模拟用户进行测试，开发环境进行

β测试：软件开发公司在日常生活中进行使用，对异常进行报告，实际环境进行

测试管理

团队要分工明确，测试要有人来做，越早开始越好

要定测试标准

测试工具

功能性测试：静态测试、动态测试

非功能性测试：负载测试，安全测试

稍微放一放

条件桩	规则													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
月	M1、M6	M2				M3				M4				
年	-	Y1、Y4	Y2、Y3			Y1、Y4	Y2、Y3			Y1、Y4	Y2、Y3		Y2	
日	-	-	D1、D7	D2~D5	D6	-	D1、D6、D7	D2~D4	D5	-	D1、D5~D7	D2	D3	D4
行动桩														
天数加一				X				X				X		

月份加一，天数重置					X				X				X	
年份加一，月份和天数重置														
输入无效	X	X	X			X	X			X	X			X

条件桩	**规则**													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
通话时间	T2	T3			T4			T5			T6			
未按时缴费次数	-	N1，N7	N2	N3~N6	N1，N7	N2,N3	N4~N6	N1，N7	N2~N4	N5,N6	N1，N7	N2~N4	N5,N6	N1 N7
行动桩														
折扣率0	X			X			X			X			X	
折扣率1%			X											
折扣率1.5%						X								
折扣率2%									X					
折扣率2.5%												X		
折扣率3%														
输入无效		X			X			X			X			X