软件测试总结

基本概念

软件的概念：

- 《GB/T 17544-1998》：信息处理系统的所有或部分程序、规程、规则和任何相关的文档的集合
  程序：源程序＋目标程序
  源程序：高级语言、汇编语言编写的程序
  目标程序：源程序经编译或解释加工以后可以由计算机直接执行的程序
  文档：用自然语言或形式化语言所编写的文字资料和图表，用来描述程序的内容、组成、设计、功能规格、开发情况、测试结果及使用方法

软件测试的对象：

不仅仅是程序，文档、数据和规程都是软件测试的对象

软件测试的定义：

使用人工或自动手段，来运行或测试某个系统的过程。其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别

软件测试的目的？是什么？
为了弄清楚模块规定的需求和预期的结果是否有偏差；

软件测试的目的是想证实在一个给定的外部环境中软件的逻辑正确性，即保证软件以正确的方式来做这个事件 √

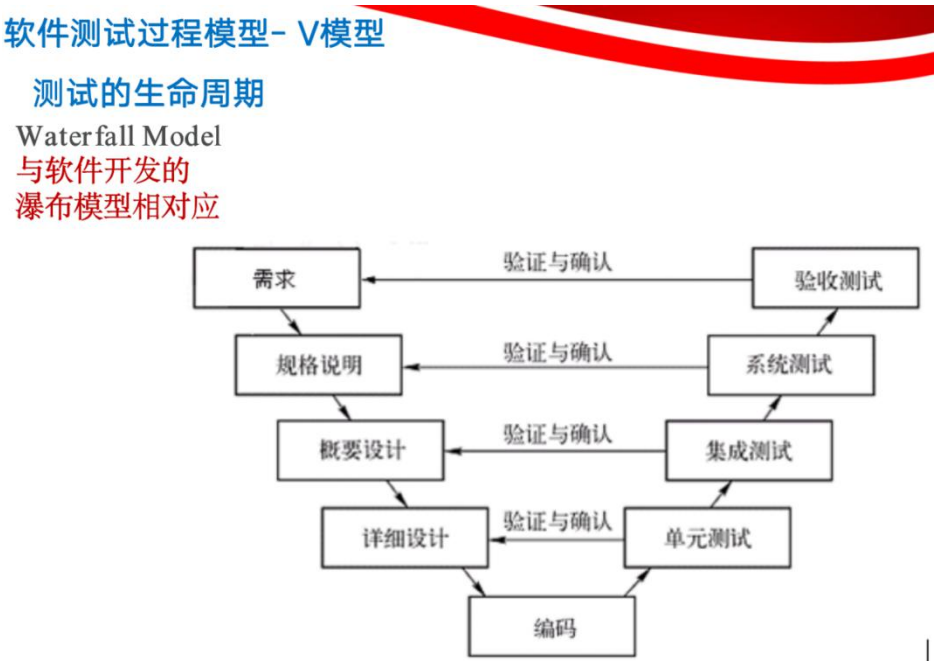
软件测试的原则：

- 不可能执行穷尽测试

- Zero bug 与 Good enough ——投入与产出平衡
- 测试应该尽早启动，尽早介入
- 软件测试应该追溯需求
- 缺陷存在群集现象——错误多的地方多投入
- 杀虫剂悖论（缺陷具有免疫性）——同化问题：交叉测试，利用不同人的观点
- 不存在缺陷的理论，测试无法显示潜伏的软件缺陷

A. 软件测试可以发现软件潜在的缺陷？ F
B. 所有的软件测试都可追溯到用户需求？T
C. 测试应尽早不断地执行 T
D. 程序员应避免测试自己的程序？T
E. 对发现错误较多的程序段，应进行更深入的测试？T
F. 测试的时机？尽早进行

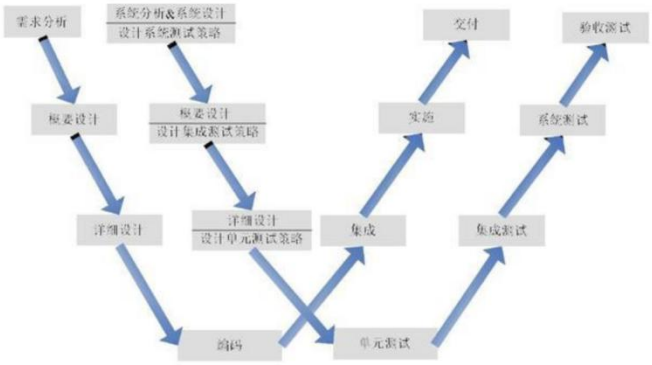
软件测试的过程模型：
V 模型：



优点:
测试 V 模型即包含了「底层测试」又包含了「高层测试」
V 模型清楚地标识出了软件开发的阶段
它采用自顶向下逐步求精的方式把整个开发过程分成不同的阶段，

缺点：测试无法尽早执行
V 模型造成需求分析阶段隐藏的问题一直到后期的验收测试才被发现
测试的开销会很大

W 模型：



W 模型是对 V 模型的改进
W 模型认为测试阶段是与开发阶段并行的
W 模型指出当需求被提交后，就需要确定高级别的测试用例来测试这些需求

W 模型优点:

开发强调测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求和概要设计同样要测试；
更早地接入测试，可以发现开发初期的缺陷，那么可以用更加低的成本进行缺陷修复。
同样是分阶段的工作，便于控制项目过程

W 模型缺点:
依赖于软件开发和软件测试依然保持一前一后的线性关系，依然无法支持迭代、自发性和需求等变更调整；
对于当前很多项目，在执行的过程中根本不产生文档，那么 W 模型基本无法适用；
使用起来技术复杂度很高，对于需求和设计的测试要求很高，实践起来困难。一般都是中型或者大型公司使用.

当详细设计编写完成后，即可执行单元测试？ F
就要确定测试条件来查找该阶段的设计缺陷
H 模型:
H 模型将测试活动分离出来，形成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

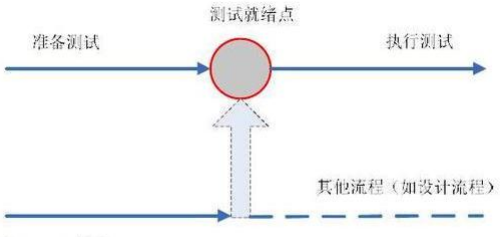
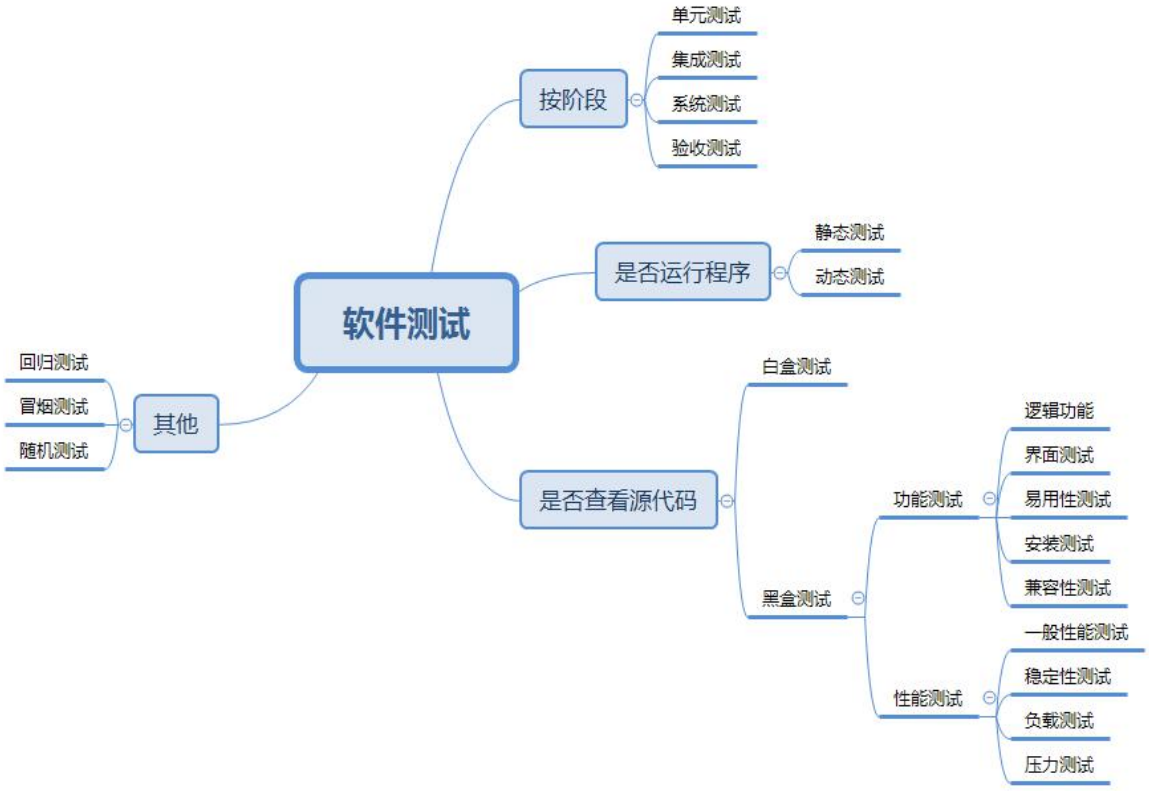


图 2-7 H 模型

软件测试的分类



核心：回归测试
测试用例是多少，是部分还是全部

回归测试是指重复以前的全部或部分的相同测试。
新加入测试的模组，可能对其他模组产生副作用，故须进行某些程度的回归测试。
回归测试的重心，以关键性模组为核心。

测试的内容是什么？

验收测试：交付前的最后一道关卡
什么样的人员来进行..：相关的用户和独立测试人员

是内部测试还是外部测试

Alpha 测试:
由用户、测试人员、开发人员等共同参与的内部测试

beta 测试:
内测后的公测．即完全交给最终用户测试。

软件测试的过程:
- 测试需求的分析和确定(是否包括测试计划的测试)

需求文档的测试

- 测试计划

对测试过程的整体设计
确定测试范围
制定测试策略
安排测试资源
进度制定
风险评估，应对策略
(测试设计及用例
测试数据、测试数据、测试结果
测试设计
用例设计
用例评估
)

- 测试执行(黑盒,白盒)

用例的选择（难的？复杂的？优先级高的？）
测试环境的搭建
每日构建（daily build）

- 测试记录和缺陷跟踪

缺陷包括哪些要素?
缺陷的状态

Bug 记录
Bug 管理
Bug 的报告（沟通，评审，提交)
Bug 的跟踪

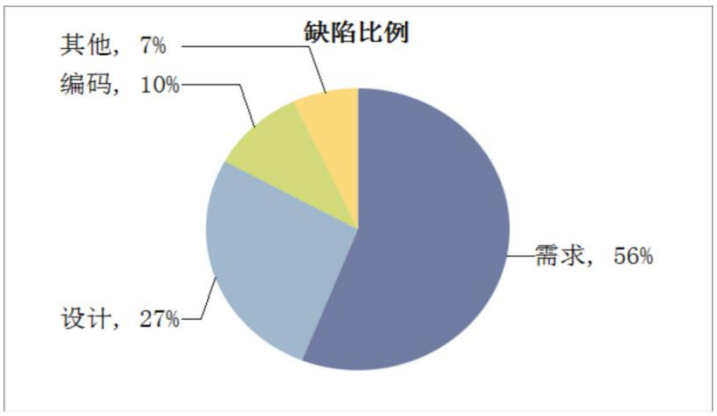
- 回归测试
- 测试总结报告

缺陷的分类报告（类型，区域，状态，趋势）
客观全面的报告生成（人员，用例，功能覆盖，时长）
经验总结

功能测试:
功能测试用例概念如下：
• 业务流程测试用例包括通过测试用例和失败测试用例。
• 功能测试用例一般包括业务流程测试用例和功能点测试用例。
• 通过测试用例是验证需求能否正确实现，打通流程的一类测试。
• 失败测试用例是模拟一些异常业务操作，测试系统是否具备容错性。
功能测试用例构成元素有哪些?
测试数据/测试步骤/预期结果

需求测试

需求的测试是重点

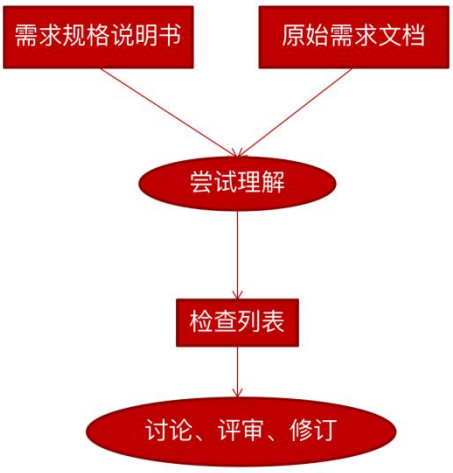


绝大部分缺陷发生在需求这个部分.参照需求规格说明书.
静态测试、动态测试???
静态测试:
1. 软件产品的需求
2. 设计规格说明书的评审
3. 对程序代码的审查
4. 静态分析

动态测试:
真正运行程序发现错误

可能存在的问题:
需求文档编写有问题、功能不明确，流程不清晰，不正确占 50%
余下 50%是需求的遗漏造成的

需求文档检查的步骤:



测试计划

1 确定测试范围
2 制定测试策略
3 测试资源安排
4 制定进度
5 评估风险及应对策略

黑盒测试

黑盒测试的基本概念:

黑盒测试是从一种从软件外部对软件实施的测试，也称功能测试或基于规格说明的测试。不知道软件是如何实现的，也不关心黑盒里面的结构，只关心软件的输入数据和输出结果。

延伸:

黑盒测试是从用户观点出发的测试，其目的是尽可能发现软件的外部行为错误。在已知软件产品功能的基础上，
检测软件功能能否按照**需求规格说明书**的规定正常工作，是否有功能遗漏；
检测是否有人机交互错误，是否有数据结构和外部数据库访问错误，是否能恰当地接收数据并保持外部信息（如数据库或文件）等的完整性；
检测行为、性能等特性是否满足要求等；
检测程序初始化和终止方面的错误等。

黑盒测试着眼于软件的外部特征，通过上述方面的检测，确定软件所实现的功能是否按照软件规格说明书的预期要求正常工作。

黑盒测试的优点:

① 黑盒测试与软件具体实现无关，所以如果软件实现发生了变化，测试用例仍然可以使用；
② 设计黑盒测试用例可以和软件实现同时进行，因此可以压缩项目总的开发时间。

穷举输入测试是不现实的。这就需要我们认真研究测试方法，以便能开发出尽可能少的测试用例，发现尽可能多的软件故障。

常用的黑盒测试方法有等价类划分、边界值分析、决策表测试等，每种方法各有所长，我们应针对软件开发项目的具体特点，选择合适的测试方法，有效地解决软件开发中的测试问题。

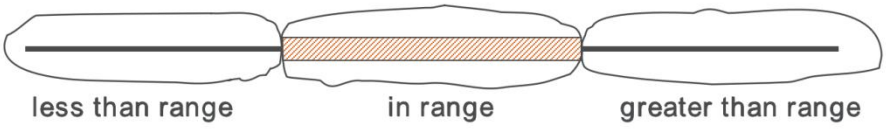
测试用例的设计方法:

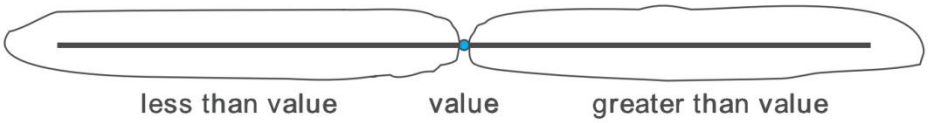
等价类划分法
边界值分析法(同时是黑盒、白盒测试都要用到的方法)
因果图法
等价类划分法
决策表法
场景设计法
错误猜测试
正交分解法 正交试验法

等价类划分法:

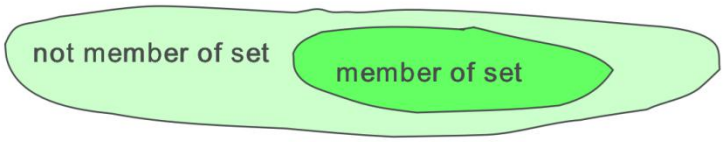
等价类划分法是一种典型的黑盒测试方法，它完全不考虑程序的内部结构，只根据程序规格说明书对输入范围进行划分，把所有可能的输入数据，即程序输入域划分为若干个互不相交的子集，称为等价类，然后从每个等价类中选取少数具有代表性的数据作为测试用例，进行测试。
理解:

- 1. 在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类

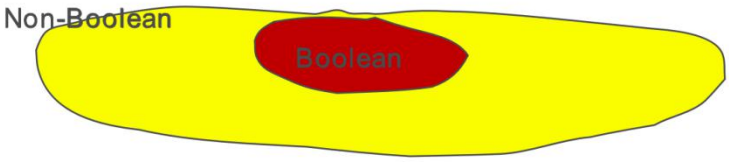




*2. 在输入条件规定了输入值的集合或者规定了“必须如何”的条件的情况下，可以确立一个有效等价类和一个无效等价类。



*3. 在输入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



- 4.在规定了输入数据的一组值(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。
- 5.在规定了输入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)。
- 6. 在确知已划分的等价类中，各元素在程序处理中的方式不同的情况下，则应再将该等价类进一步地划分为更小的等价类。——细分等价类

边界值分析法:

健壮性边界值测试将产生 6n+1 个测试用例. 七个

min-， min， min + ， nom， max-, max, max+,

非健壮性边界值测试: 4n+1 不需要超出范围的两个.

健壮性测试最有意义的部分不是输人，而是预期的输出，观察例外情况如何处理。

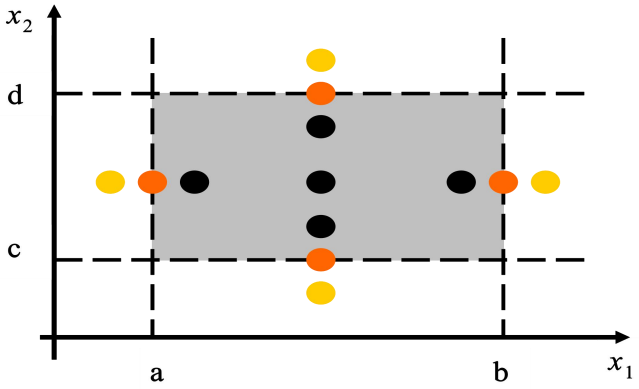
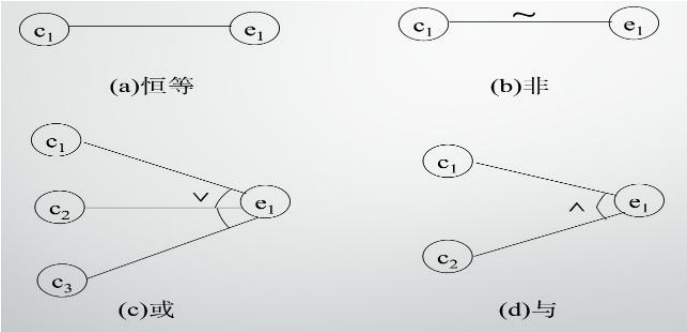


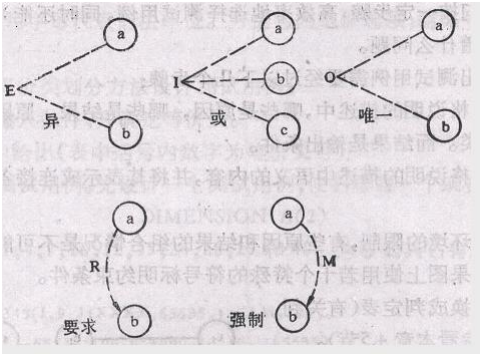
图2.2 健壮性边界值测试用例

因果图法:(因果关系描述)

因果之间的约束:



输入状态之间的约束:



E 约束（异）：a 和 b 中至多有一个可能为 1，即 a 和 b 不能同时为 1。
I 约束（或）：a、b 和 c 中至少有一个必须是 1，即 a、b 和 c 不能同时为 0。
O 约束（唯一）；a 和 b 必须有一个，且仅有 1 个为 1。
R 约束（要求）：a 是 1 时，b 必须是 1，即不可能 a 是 1 时 b 是 0。

输出条件约束类型

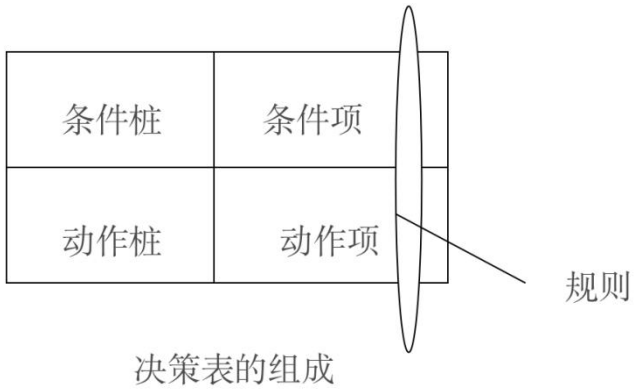
输出条件的约束只有 M 约束（强制）：若结果 a 是 1，则结果 b 强制为 0。

因果图法测试用例的设计步骤:
（1）确定软件规格中的原因和结果。分析规格说明中哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。
（2）确定原因和结果之间的逻辑关系。分析软件规格说明中的语义，找出原因与结果之间、原因与原因之间对应的关系，根据这些关系画出因果图。
（3）确定因果图中的各个约束。由于语法或环境的限制，有些原因与原因之间、原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号表明约束或限制条件。
（4）把因果图转换为决策表。
（5）根据决策表设计测试用例。
如何将因果图转换为决策表:如何缩减
----------不考----------

决策表法:

输入条件或输出结果都可以用成立或不成立来表示,即只有 1/0 两个取值.

决策表通常由条件桩、条件项、动作桩和动作项 4 部分组成。



决策表的组成

动作项和条件项紧密相关，指出在条件项的各组取值情况下应采取的动作

判定表的元素

条件桩，列出问题的所有条件
动作桩：列出可能针对问题所采取的操作

条件项：针对所列条件的具体赋值
动作项：列出在条件项（各种取值）组合情况下应该采取的动作。
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。

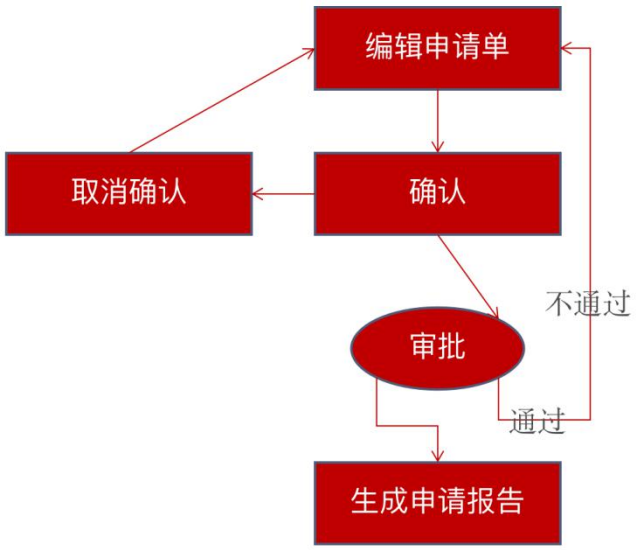
判定表法的使用步骤:
1. 列出条件桩
2. 列出动作桩
3. 填入条件项及其组合
4. 填入动作项，制定初始判定表；
5. 简化、合并相似规则或者相同动作

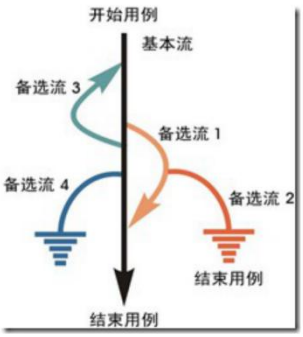
正交实验法:

一般用 L 代表正交表，例如，　L8(27) ——7 为列数（最多可安排的因子数），2 为因子水平数；8 为此表行数（试验次数）
因子:变换的维度,方面
水平:每个维度,方面总共有的取值
m_因子,n_水平　试验次数 ： m*(n-1)+1
eg:　7*（2-1）+1=8

场景设计法:

大部分软件是由事件触发来控制流程的，事件触发时的情景就是所谓的场景





基本流
基本流、备选流 4

基本流、备选流 1；
基本流、备选流 1、备选流 2；
基本流、备选流 3；
基本流、备选流 3、备选流 1；
基本流、备选流 3、备选流 1、备选流 2
基本流、备选流 3、备选流 4

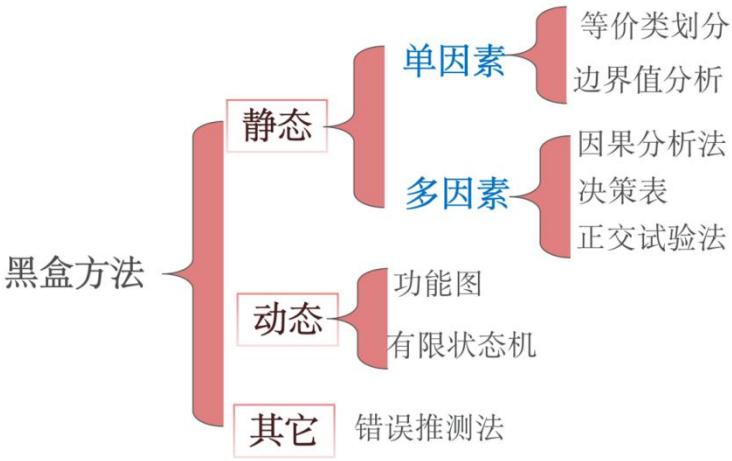
错误猜测法:

是基于经验的直觉推测程序中可能发生的各种错误，有针对性设计测试用例。
优点：充分发挥个人的经验和潜能，命中率高

缺点：覆盖率难以保证；过多的依赖个人的经验

注意：最重要的是要思考和分析测试对象的各个方面，多参考以前发现的 Bug 的相关数据、总结的经验，个人多考虑异常的情况、反面的情况、特殊的输入，以一个攻击者的态度对待程序，那么就能设计出比较完善的测试用例。

总结:



进行等价类划分，包括输入条件和输出条件的等价类划分，将无限测试变成有限测试，**这是减少工作量和提高测试效率最有效的方法。**

在任何情况下都必须使用边界值分析方法，经验表明，**用这种方法设计出的测试用例发现错误的能力最强。**
如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法和判定表法。
对于配置参数类软件，用**正交试验法选择较好的组合方式达到最佳效果**

功能图法也是很好的测试用例设计方法，可以通过不同时期条件的有效性设计不同的测试数据

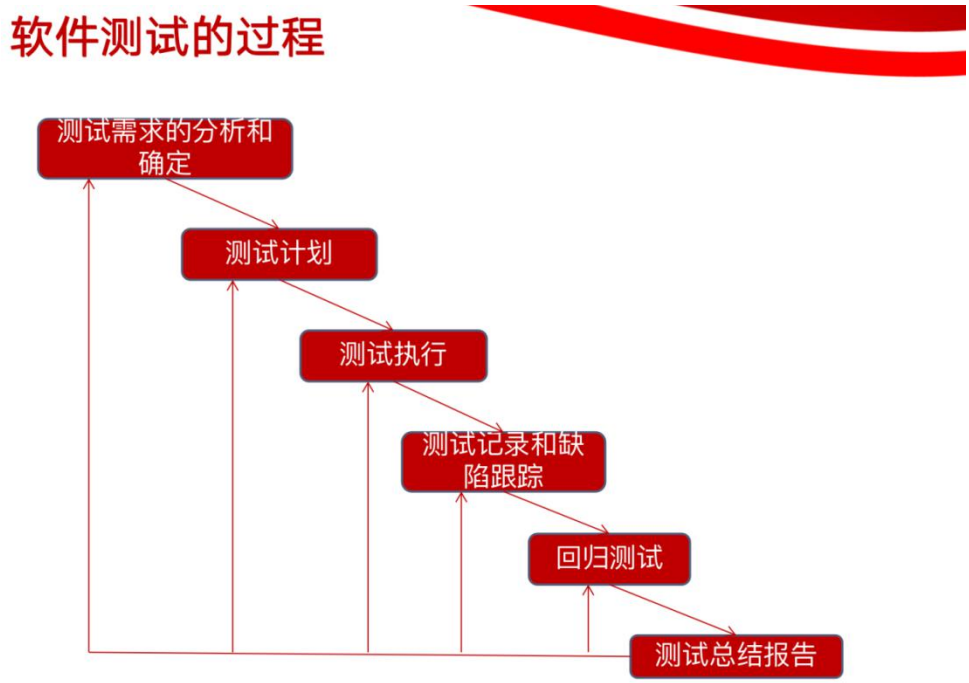
对于业务清晰的系统，可以利用场景法贯穿整个测试案例过程在案例中综合使用各种测试方

法

可以用错误推测法追加一些测试用例，这需要依靠测试工程师的智慧和经验

补充:
功能图法实际是一种黑盒、白盒混合用例设计方法

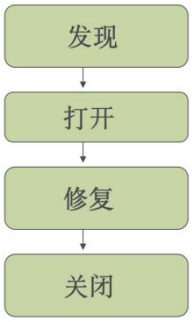
测试的执行



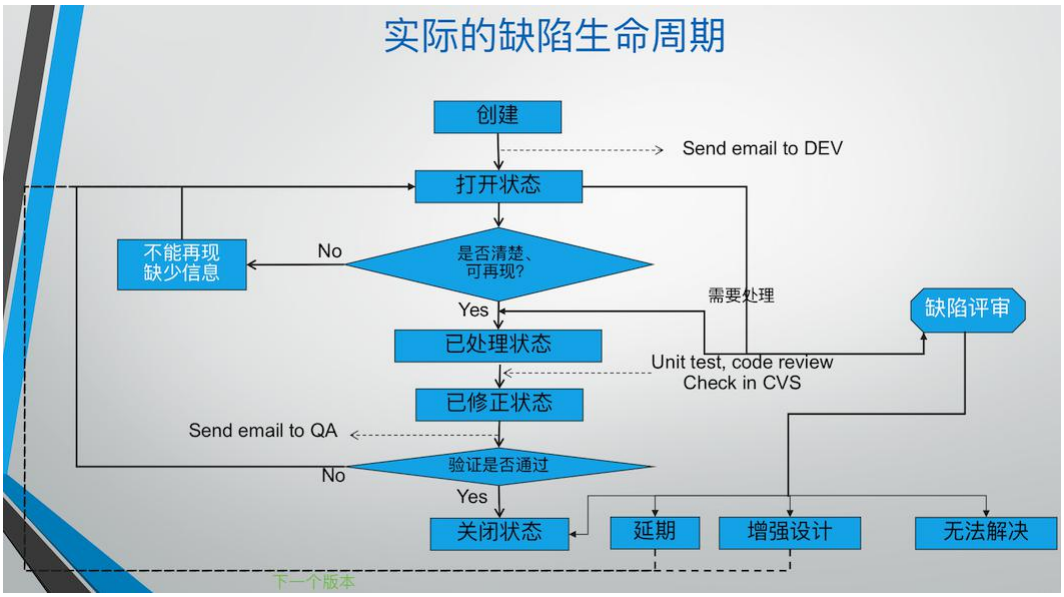
软件执行的关键:
测试环境的准备。
　　构建测试运行的平台和安装需要的软硬件系统。
人员的安排。
　　不仅包括指定哪些人参加功能测试，哪些人参加系统测试和谁负责测试环境的维护等，还要包括人员的培训，知识的传递。

测试用例包括哪些内容: ???
测试功能（目标），测试环境，测试要录入的数据，测试的具体操作，预期结果，
不包括实际结果

基本的缺陷生命周期:
发现-打开：测试人员找到软件缺陷并将软件缺陷提交给开发人员。——〉新打开。 open
打开-修复：开发人员再现、修复缺陷，然后提交给测试人员去验证。 ——〉已修正。fix
修复-关闭：测试人员验证修复过的软件，关闭已不存在的缺陷。——〉已关闭 close



实际的缺陷生命周期:



(每一步谁做)

白盒测试

静态白盒测试:(结构化分析)

基本概念

在不执行软件的条件下有条理地仔细审查软件的设计、体系结构和代码，从而找出软件缺陷的过程 ， 有时也称为结构化分析

原因

尽早发现软件错误;为黑盒测试人员提供建议

方式:

正式审查: 1.确定问题 2.遵守规则 3.准备期间 4.编写报告

方法:

互查、走查、会议评审

动态白盒测试

检查代码并观察运行状况.

利用查看代码（做什么）和实现方法(怎么做)得到的信息来确定哪些需要测试、哪些不要测试、如何开展测试

又称为结构化测试(structral testing)

白盒测试期望达到的目的

所有独立路径至少都能测试一遍；

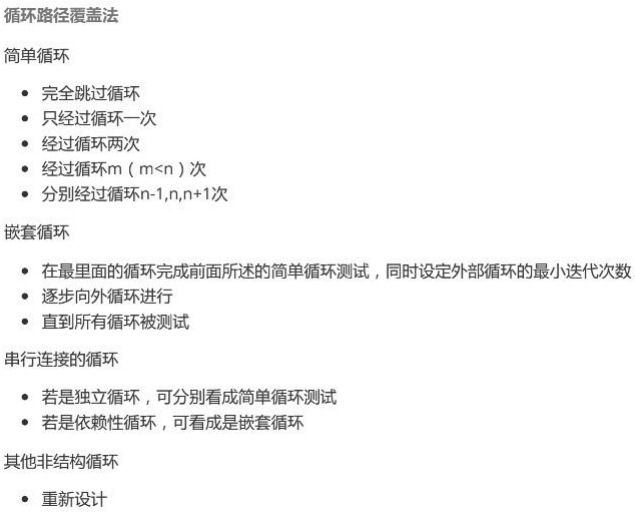
所有逻辑判断都能测试 True 和 False 两条路径；

所有循环结构都能测试到边界和循环域内的情况；

确保内部数据结构的有效性。

白盒测试主要方法:

1. 逻辑覆盖测试法
2. 基本路径测试法
3. 循环路径测试法



逻辑覆盖测试法

语句覆盖(SC):
每条语句至少执行一次
语句覆盖是最起码的结构覆盖要求，语句覆盖要求设计足够多的测试用例，使得程序中每条语句至少被执行一次。

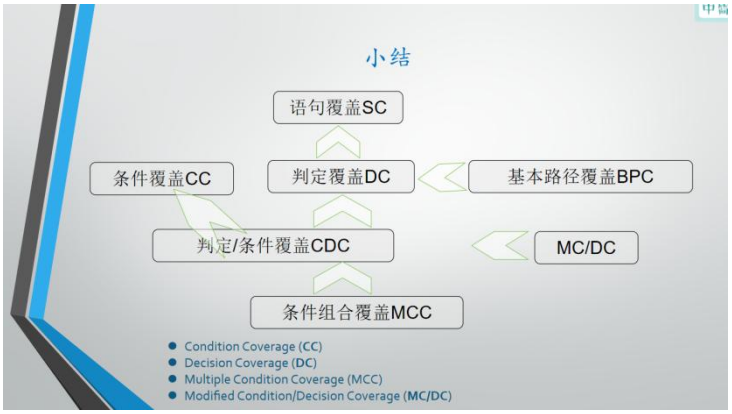
判定覆盖:(DC)
又称分支覆盖,每个判断 if 的分支真假都至少取一次
每个判定的每个分支至少执行一次
条件覆盖:(CC)
要使每个判断中每个条件的可能取值至少满足一次
每种条件下的语句都应该被执行。

判定/条件覆盖:(CDC) 包含了 判定、条件覆盖
同时满足判定覆盖和条件覆盖

使得判断条件中的所有条件可能取值至少执行一次,同时所有判断的可能结果至少执行一次
优点：判定/条件覆盖满足判定覆盖准则和条件覆盖准则，弥补了二者的不足。
缺点：判定/条件覆盖准则的缺点是未考虑条件的组合情况。

条件组合覆盖:(MCC)
每个判定中，各条件的每一种组合至少出现一次。
一个判断里面,所有条件组合一次.进而每个判断都进行一次

满足条件组合覆盖，一定满足判定 DC 覆盖、条件 CC 覆盖、条件判定 CDC 组合覆盖



修正条件逻辑判定(MC/DC)
1. 每一个判断的所有可能结果都出现过
2. 每一个判断中所有条件的取值都出现过
3. 每一个进入点及结束点都执行过
4. 判断中每一个条件都可以独立地影响判断的结果

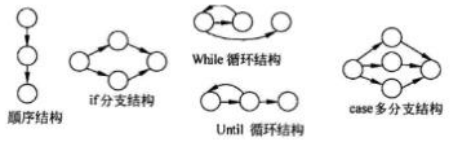
and	or
不需要 F F	不需要 T T

设计某一种覆盖的测试用例
修正条件/判定覆盖???
覆盖之间的包含关系,谁强谁弱?

基本路径覆盖法:

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。

控制流图是描述程序控制流的一种图示方法。其基本符号有圆圈和箭线：圆圈为控制流图中的一个结点，表示一个或多个无分支的语句；带箭头的线段称为边或连接，表示控制流。基本结构如下所示：



圈复杂度的计算？V-n+2 / 判断数+1 / 区域数

动态白盒测试步骤:

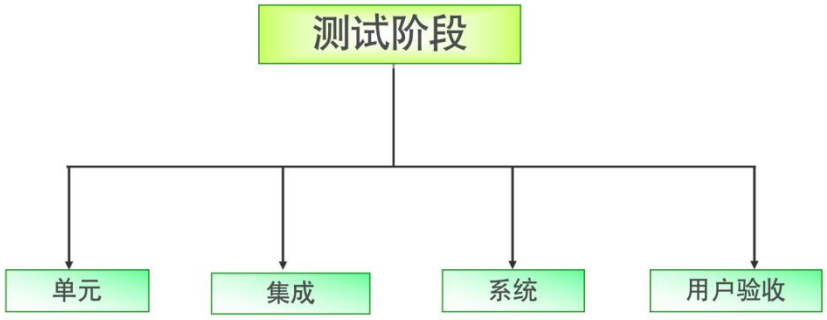
1 分析模块函数;
2 在模块中找到相应的关键点（函数）;
3 根据第二点，画出模块程序流程图;
4 计算圈复杂度;
5 根据圈复杂度算出测试用例的最优个数;
6 根据路径测试法和圈复杂度写出具体测试用例;
7 进行测试。

单元测试

将测试进行分段

测试越早发生越好。
代码分段构建和测试，最后合在一起形成更大的部分。

单元测试：接口，数据边界，路径，异常，局部变量（数据）
可测可不测：条件组合，性能，功能



单元测试的概念: 局部变量进行测试!!!

定义
单元测试是对软件基本组成单元进行的测试

时机
在代码完成后由开发人员完成,QA 人员辅助

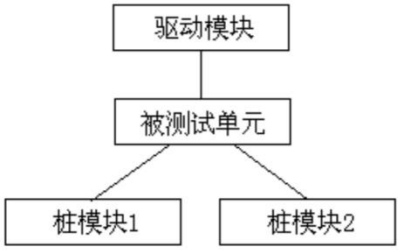
意义
尽早发现错误，发现的越早成本越低

越早发现错误,修改的成本越低.

单元测试的内容:
运行单元程序有时需要基于被测单元的接口，开发相应的驱动模块和桩模块。
静态测试技术的运用——单元测试最重要的手段:
1. 互查
2. 走查:采用讲解、讨论和模拟运行的方式进行的查找错误的活动
3. 评审

	走 查	审 查
准备	通读设计和编码	事先准备Spec、程序设计文档、源代码清单、代码缺陷检查表等
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员包括测试人员
主要技术方法	无	缺陷检查表
生成文档	会议记录	静态分析错误报告
目标	代码标准规范 无逻辑错误	代码标准规范 无逻辑错误

驱动模块（drive）:对底层或子层模块进行测试所编写的调用这些模块的程序。
桩模块（stub）：对顶层或上层模块进行测试时所编写的替代下层模块的程序。



集成测试

对全局变量进行测试.

集成测试的层次:
对于传统软件来说，按集成粒度不同，可以把集成测试分为 3 个层次，即:
（1）模块间集成测试

（2）子系统内集成测试
（3）子系统间集成测试

集成测试的模式

渐增式测试模式与非渐增式测试模式

渐增式测试模式

把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。

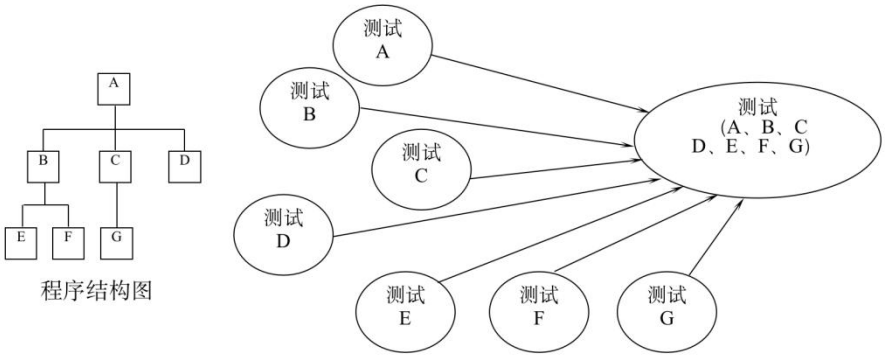
当使用渐增方式把模块结合到程序中去时，有自顶向下和自底向上两种集成策略。

自顶向下和自底向上集成策略 ：
那些模块的错误不容易被发现?各自的缺点、优点

驱动程序/驱动模块（driver），用以模拟被测模块的上级模块。驱动模块在集成测试中接受测试数据，把相关的数据传送给被测模块，启动被测模块，并打印出相应的结果。

桩程序/桩模块（stub），也有人称为存根程序，用以模拟被测模块工作过程中所调用的模块。桩模块由被测模块调用，它们一般只进行很少的数据处理，例如打印入口和返回，以便于检验被测模块与其下级模块的接口

非渐增式测试模式：
先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，如大棒方法。



程序结构图

非渐增式集成

总结:

①集成测试也叫做组装测试，通常是在单元测试的基础上，将模块按照设计说明书要求进行组装和测试的过程?
②自底向上的增殖方式是集成测试的一种组装方式，它能较早地验证主要的控制和判断点，对于输入输出模块、复杂算法模块中存在的错误能够较早地发现?

③单元测试的目的在于检查被测模块能否正确实现详细设计说明中的模块功能、性能、接口和设计约束等要求？
④集成测试需要重点关注各个模块之间的相互影响，发现并排除全局数据结构问题？

系统测试

使用的是黑盒测试的技术
1. 功能测试
2. 回归测试
3. 性能测试
4. 其它非功能性测试

目的,执行人是谁,所用的方法是什么

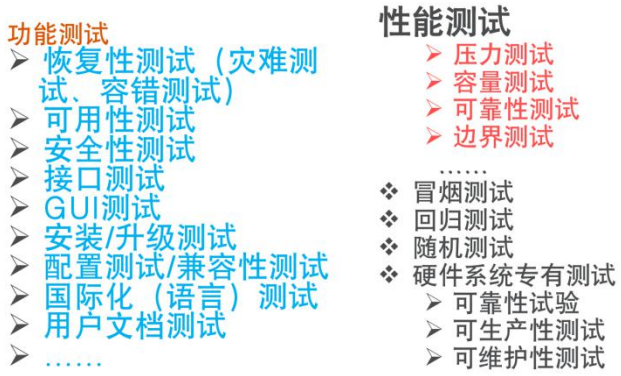
系统测试使用的方法是黑盒测试方法

测试阶段	目的	执行者	测试方法
单元测试	查找独立模块中逻辑错误、数据错误和算法错误	软件工程师	白盒测试
集成测试	查找模块之间接口错误	软件工程师 测试人员	白盒测试 自顶向下 或自底向上
系统测试	对系统中各个组成部分进行综合性检验	测试人员	黑盒测试 模拟用户操作
回归测试	确认软件变更后是否仍满足软件需求	测试人员	黑盒测试 模拟用户操作
验收测试	确认软件是否满足用户需求	用户 项目组测试人员	黑盒测试 模拟用户操作

系统测试的概念:
为了发现缺陷并度量产品质量，按照系统的功能和性能需求进行的测试

一般使用黑盒测试技术,一般由独立的测试人员完成

系统测试的内容:



回归测试:

回归测试的目的 ：
所做的修改达到了预定的目的，如错误得到了改正，新功能得到了实现，能够适应新的运行环境等；不影响软件原有功能的正确性。

一旦程序某些区域被修改了，就可能影响其它区域，导致受影响的区域出现新的缺陷（回归缺陷）。如果这时没有回归测试，产品就带着这样的回归缺陷被发布出去了，造成严重后果。回归测试就是为了发现回归缺陷而进行的测试。

回归测试应该执行初测时所用的全部测试用例。即使通过多次的回归测试，也很难发现所有缺陷。验收测试可能需要多次回归测试。

性能测试:
性能测试（performance test）就是为了发现系统性能问题或获取系统性能相关指标而进行的测试。一般在真实环境、特定负载（正常或峰值）条件下，通过工具模拟实际软件系统的运行及其操作，同时监控性能各项指标，最后对测试结果进行分析来确定系统的性能状况。

性能测试工具:

性能测试包括以下几个方面:

评估系统的能力。测试中得到的负荷和响应时间等数据可以被用于验证所计划的模型的能力，并帮助做出决策。

识别系统中的弱点。受控的负荷可以被增加到一个极端的水平并突破它，从而修复系统的瓶颈或薄弱的地方。

系统调优。重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能，检测软件中的问题。

负载压力测试:

负载压力测试，在一定约束条件下测试系统所能承受的并发用户量、运行时间、数据量，以确定**系统能承受的最大负载压力**。

负载压力测试是性能测试的重要组成部分，负载压力测试包括并发性能测试、疲劳强度测试、大数据量测试等。

压力测试&负载测试 ??

负载测试(正常情况下)，通过逐步增加系统负载，测试系统性能的变化，并最终确定在满足性能指标的情况下，系统能承受的最大负载量的测试。

压力测试，通过逐步增加系统负载，测试系统性能的变化，并最终确定在什么负载条件下系统性能处于**失效**状态，并以此来获得系统能提供的最大服务级别的测试。
目的：压力测试是为了发现在什么条件下系统的性能会变得不可接受。

压力测试类型
在一种需要反常（如长时间的峰值）数量、频率或资源的方式下，执行可重复的负载测试，以检查程序对异常情况的抵抗能力，找出性能瓶颈或其它不稳定性问题
1. 并发性能测试
2. 疲劳强度测试
3. 大数据量测试

不同测试的辨析
- 压力测试
- 负载测试
- 稳定性测试

Eg:
一般性能测试：背 1 袋米
稳定性能测试：背 1 袋米，操场跑步多久累到
负载测试：背 2 袋米，多久累到
压力测试：背 2 袋米，3 袋米，4 袋米....发现他最多能背 3 袋米。

负载测试（Load Test）和并发测试(理解):

负载测试是通过逐步增加系统工作量，测试系统能力的变化，并最终确定在满足功能指标的情况下，系统所能承受的最大工作量的测试。

压力测试实质上就是一种特定类型的负载测试。

并发性测试是一种测试手段，在压力测试中可以利用并发测试来进行压力测试。

自动化测试

为什么要进行自动化测试?
提高测试效率,
一系列的缺陷;

正确认识测试自动化(最难实现的部分,包括哪些流程)

最难的部分: 测试用例的设计.

不现实的期望注定测试自动化的失败

测试自动化能:

显著降低重复手工测试的时间
建立可靠、重复的测试，减少认为错误
增强测试质量和覆盖率

测试自动化不能:

完全替代手工测试和手工测试工程师
保证 100%的测试覆盖率
弥补测试实践的不足

说明各自应用范围:

在系统功能逻辑测试、验收测试、适用性测试、涉及物理交互性测试时，多采用手工测试(黑盒)方法;

单元测试、集成测试、系统负载或性能、稳定性、可靠性测试等比较适合采用 TA;

对那种不稳定软件的测试、开发周期很短的软件、一次性的软件等不适合测试自动化

功能测试时，工具更能发挥回归测试作用，因为工具缺乏想象力和灵活性而不能发现更多的新问题（自动测试只能发现 15%的缺陷，而手工测试可以发现 85%的缺陷），但可以保证对已经测试过部分进行测试的准确性和客观性

怎么来实施的?测试的流程是什么?

自动化性能测试原理

首先保证一个用户能正常访问，记录访问过程，记录通讯包

通过测试工具模拟更多用户同时发通讯包，后台无法区分是人 or 工具

通过测试工具模拟大量用户同时向后台发出请求，来达到产生压力和指定压力的目的，在产生指定压力的同时监控后台系统的资源消耗情况，监控客户端的请求处理时间

复制出客户端发往服务端的请求，模拟用户关注的是通讯包，协议，不关心客户端形式.

题目总结:

1. 简述性能测试的基本过程，文字描述或采用图示均可。
性能测试需求分析——〉性能测试计划——〉性能测试用例——〉测试脚本编写——〉测试场景设计——〉测试场景运行-〉场景运行监控——〉运行结果分析——〉系统性能调优-〉性能测试总结。

2. 简述软件开发-软件测试的对应关系。
答：软件开发过程是一个自顶向下逐步细化的过程（**1 分**）。软件测试是一个自底向上的集成过程（**1 分**）。对需求验证对应验收测试，客户需求的确认测试（**1 分**）；系统架构设计的验证对应系统的非功能性测试（**1 分**）；产品详细设计的验证对应功能测试（**1 分**）；代码的验证对应单元测试和集成测试（**1 分**）。

3、简述α测试和β测试。
答：α测试由用户、测试人员、开发人员等共同参与的内部测试（**3 分**）。β测试是内测后的公测. 即完全交给最终用户测试（**3 分**）。

4、介绍缺陷生命周期所涉及的缺陷状态，考虑的相关处理，并给出合理的缺陷生命周期图示。
答： 缺陷状态，包括新打开、已修复、已关闭等（2 分）。可以考虑缺陷重新打开、是否是缺陷？是否推迟缺陷处理等方面（4 分）。

5. 简述性能测试的基本过程，文字描述或采用图示均可。
性能测试需求分析——〉性能测试计划——〉性能测试用例——〉测试脚本编写——〉测试场景设计——〉测试场景运行-〉场景运行监控——〉运行结果分析——〉系统性能调优-〉性能测试总结。