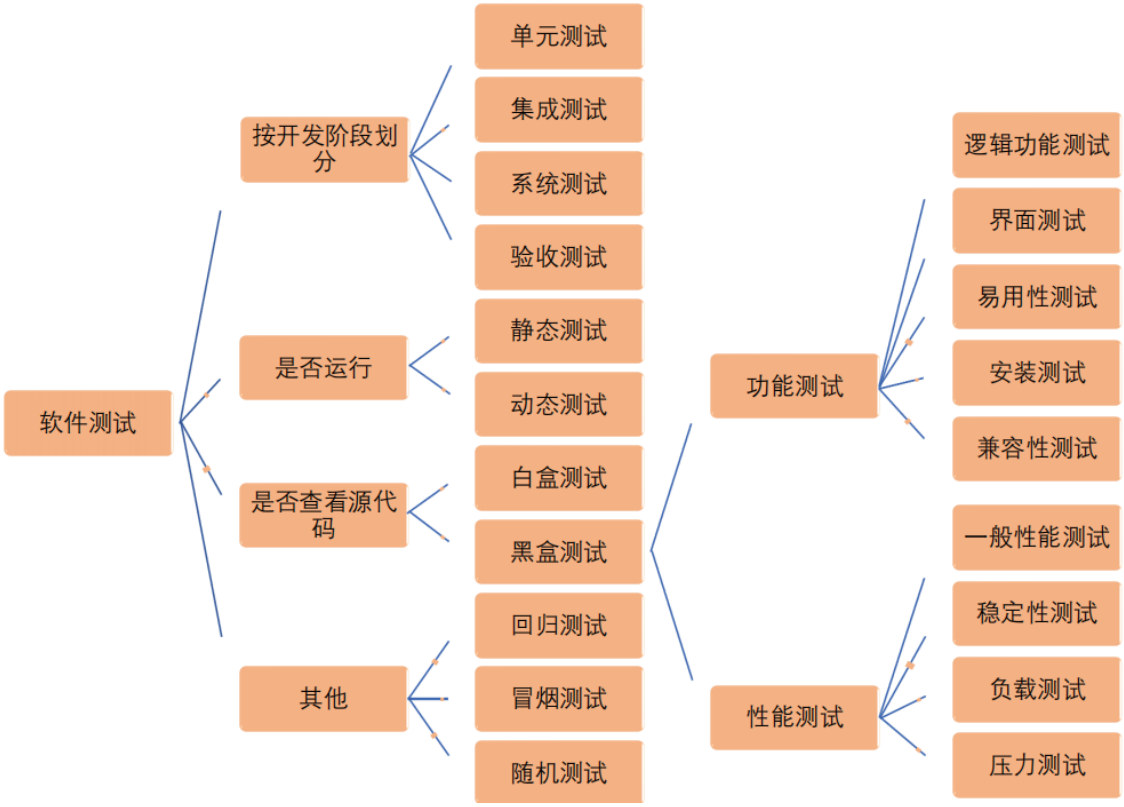
一、名词解释

测试方法：

1. **软件测试分类**：



2. **单元测试**

   又称模块测试，针对软件设计中的最小单位--程序模块，进行正确性检查的测试工作。单元测试需要从程序的内部结构出发设计测试用例。

   多个模块可以平行地独立进行单元测试。

3. **集成测试**

   集成测试又叫组装测试，在单元测试的基础上，将所有程序模块进行有序的、递增的测试。重点测试不同模块的接口部分。

   主要由白盒测试工程师或者开发工程师完成。

   集成测试的依据：单元测试的模块+《概要设计》文档。

4. **系统测试**：

   将整个软件系统看为一个整体进行测试，包括对功能、性能、以及软件所运行的软硬件环境进行测试。

   主要由黑盒测试工程师在系统集成完毕后进行测试。

   前期：测试系统的功能是否满足需求。　后期：测试系统运行的性能是否满足需求，系统在不同的软硬件环境中的兼容性。

5. **验收测试**

   按照项目任务书或合同、供需双方约定的验收依据文档进行的对整个系统的测试与评审，决定是否接收或拒收系统。

   ◆**α**测试：指的是指的是由用户，测试人员、开发人员等共同参与的内部测试。

   ◆**β**测试：指的是内测后的公测，即完全交给最终用户测试。

6. **静态测试**

   不实际运行被测软件，而只是静态地检查程序代码、界面或文档中可能存在的错误过程。

7. **黑盒测试**

   黑盒测试指测试人员通过各种输入和观察软件的各种输出结果来发现软件的缺陷，而不关心程序具体如何实现的一种测试方法。

8. **白盒测试**

   白盒测试又叫做结构测试，把程序看成装在一个透明的白盒子里，按照程序内部的逻辑测试程序，检测程序中的主要执行通路是否都能按预定要求正确工作。

9. **静态白盒测试**

   在**不执行软件**的条件下有条理地仔细审查软件设计、体系结构和代码，从而找出软件缺陷的过程。

10. **性能测试**

    测试软件是否达到需求规格说明中规定的各类性能指标，并满足相关的约束和限制条件。

    时间性能，空间性能，一般性能测试，可靠性测试，负载测试，压力测试。

11. **功能测试**

    是黑盒测试的一方面，它检查实际软件的功能是否符合用户的需求。

    逻辑功能测试，界面测试，易用性测试，安装测试。

12. **回归测试**

    回归测试是指软件被修改后重新进行的测试，如重复执行上一个版本测试时的用例，是为了保证对软件所做的修改没有引入新的错误而重复进行的测试。

13. **编译检查测试（ BVT测试）**

    检查源代码是否能正确编译成一个新的、完整可用的版本。

    优点是时间短，验证了软件的基本功能。

    缺点是测试的覆盖率很低。

14. **冒烟测试**

    冒烟测试是指在对一个新版本进行系统大规模的测试之前，先验证一下软件的基本功能是否实现，是否具备可测试性。

15. **随机测试**

    测试中所有的输入数据都是随机生成的。目的是模拟用户的真实操作，并发现一些边缘性的错误。

16. **条件组合覆盖**

    组合覆盖：通过执行足够多的测试用例，使得程序中每个判定的所有可能的条件取值组合都至少出现一次。
    满足组合覆盖的测试用例一定满足判定覆盖、条件覆盖和判定/条件覆盖。

17. **变异测试**

    当测试人员采用变异技术来评价测试集的充分性或是增强测试集时，这种活动就被称为是变异测试。

    通过对比源程序与变异程序在执行同一测试用例时差异来评价测试用例集的错误检测能力。

18. **兼容性测试**

    检查软件能否在不同组合的环境下正常运行，或者软件之间能否正常交互和共享信息。软件兼容性是衡量软件好坏的重要指标之一。

19. **第三方测试**

第三方测试主要是指由开发者和用户以外的第三方进行的软件测试，其目的是为了保证测试的客观性。

第三方定义：

狭义：独立的第三方测试机构
广义：非本软件的开发人员

20. **确认测试**

确认测试又称有效性测试。有效性测试是在模拟的环境下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。

验收测试指按照项目任务书或合同、供需双方约定的验收依据文档进行的对整个系统的测试与评审，决定是否接收或拒收系统。

21. **压力测试**

对系统不断施加压力的测试，通过确定一个系统的瓶颈或者不能接收的性能点，获得系统能提供的最大服务级别的测试。

22. **负载测试**

通过在被测系统上不断加压，直到性能指标达到极限，例如“响应时间”超过预定指标或某种资源已经达到饱和状态。

23. **安全测试**

测试软件在没有授权的内部或者外部的用户的攻击或者恶意的破坏时如何进行处理，是否能保证软件和数据的安全。

24. **自动化测试**

自动化测试就是通过测试工具或其他手段，按照测试工程师的预定计划对软件产品进行自动化的测试。

25. **软件质量保证**

软件质量保证是贯穿软件项目整个生命周期的有计划和有系统的活动，经常针对整个项目质量计划执行情况进行评估，检查和改进，向管理者、顾客或其他方提供信任，确保项目质量与计划保持一致。

26. **&&逻辑覆盖**

逻辑覆盖是一种以程序内部逻辑结构为依据的用例设计方法，包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖等几种覆盖强度各不相同的逻辑覆盖形式。

27. **开发者测试**

软件的持续快速迭代需求，大大压缩了软件开发的发布流程，使得一部分测试任务开始迁移，由软件开发人员担任这部分跟代码相关的软件测试工作。

开发者测试包括了传统的单元测试、集成测试、接口测试甚至部分系统测试相关的任务。（不是单纯的白盒测试或者黑盒测试）

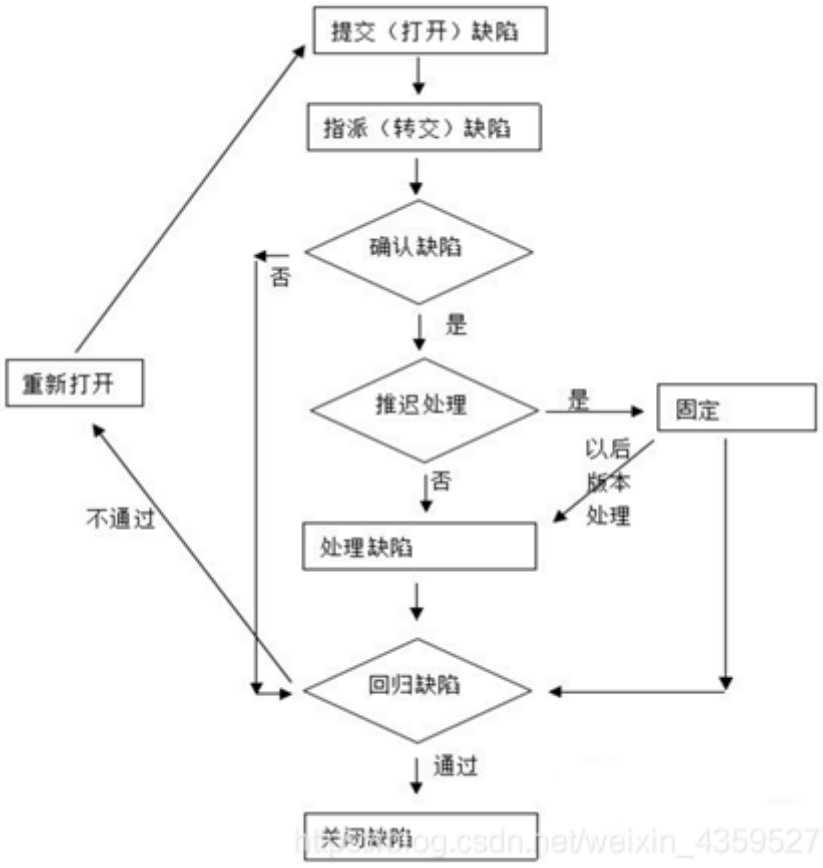
28. **大爆炸集成测试**

将所有的模块集合在被测系统之中，而不考虑构件之间的相依性或风险。应用一个系统范围的测试包以证明最低限度的可操作性。 （通过少数测试运行检测整个系统来论证系统的稳定性）

流程解释：

1. **软件缺陷（定义）**
   - 软件未达到产品说明书中标明的功能。
   - 软件出现了产品说明书中指明的不会出现的功能。
   - 软件功能超出了产品说明书中指明的范围。
   - 软件未达到产品说明书中指明应达到的目标。
   - 软件测试人员认为软件难以理解和使用、运行速度慢，或最终用户认为不好。

     符合以上任意一种情况，即为软件缺陷。
2. **软件缺陷生命周期**
   - 测试人员识别缺陷，其初始状态是“新建”
   - 分析缺陷，分配给合适的开发人员解决，状态流转为“待解决”
   - 工程师解决缺陷，将其状态跟踪到“已解决”
   - 测试人员回归该缺陷，若回归通过，则关闭缺陷，若回归不通过，则重新打开该缺陷（“Reopen”状态）



3. **静态质量特性**
   - 结构化的、可维护的、可测试的代码
   - 正确而又完整的文档
4. **动态质量特性**
   - 正确性：软件针对其**输入域**中的每个元素都如期望运行。
   - 可靠性：软件在给定时间间隔和给定条件下无故障运行的概率。
   - 完整性：全部达到软件需求规格说明或者用户手册中所有功能的可能性。
   - 一致性：软件对常规惯例和假设的遵循程度。
   - 易用性：软件使用的难易程度。
   - 性能：软件完成规定任务所花费的时间。
5. **软件测试特性**

黑盒测试的复杂性：穷举输入测试：

输入量太大，输出太多，实现的途径太多，规格说明没有客观标准

白盒测试的复杂性：穷举路径测试：

不能保证程序实现符合规格说明的要求。

不可能查出程序中因遗漏路径而出现的错误。

可能发现不了有关数据的故障。

经济性。

6. **测试用例设计的原则**

   正确性，全面性，连贯性，可判定性，可操作性。

7. **测试充分性准则**
   - **空测试**对于任何软件测试都是不充分的。
   - **有限性**：任何软件都存在有限的充分测试数据集。
   - **单调性**：如果一个测试数据集对一个软件系统的测试是充分的，那么在增加一些测试用例也是充分的。
   - **复杂性**：软件越复杂，需要的测试用例越多。
   - **回报递减律**：测试得越多，进一步测试所能得到的充分性增长就越少。
8. **测试数据集充分性公理**
   - **非外延性公理**：如果有两个功能相同而实现不同的程序，对其中一个是充分的测试数据集对另一个不一定是充分的。
   - **多重修改公理**：如果两个程序具有相同的语法结构，对一个是充分的测试数据集对另一个不一定是充分的。
   - **不可分解公理**：对一个程序进行了充分的测试，并不表示对其中的成分都进行了充分的测试。
   - **非复合性公理**：对程序各单元是充分的测试数据集并不一定对整个程序（集成后）是充分的。
9. **软件开发过程**

   是软件开发与维护的工作流程和工艺流程，是软件工程的重要组成部分。

   可行性研究->需求分析->概要设计->详细设计->实现->集成测试->确认测试->使用与维护

10. **软件可靠性**

    软件可靠性定义为在某个给定时间间隔内，程序按照规格说明成功运行的概率。

    1983年美国IEEE计算机学会对“软件可靠性”做出了明确定义，此后该定义被美国标准化研究所接受为国家标准，1989年我国也接受该定义为国家标准。该定义包括两方面的含义：
    （1）在规定的条件下，在规定的时间内，软件不引起系统失效的概率；
    （2）在规定的时间周期内，在所述条件下程序执行所要求的功能的能力；
    其中的概率是系统输入和系统使用的函数，也是软件中存在的故障的函数，系统输入将确定是否会遇到已存在的故障（如果故障存在的话）。

其它名词：

1. **驱动程序**（driver），对底层或子层模块进行（单元或集成）测试时所编制的调用被测模块的程序，用以模拟被测模块的上级模块。
2. **桩程序**（stub），也有人称为存根程序，对顶层或上层模块进行测试时，所编制的替代下层模块的程序，用以模拟被测模块工作过程中所调用的模块。
3. **测试用例**

   测试用例，英文名为TestCase，缩写为TC，指的是在测试执行之前设计的一套详细的测试方案，包括测试环境、测试步骤、测试数据和预期结果。

二、问答题

一、分析软件缺陷产生的原因。

（1）交流不充分及沟通不畅

（2）软件需求的变更

（3）软件开发工具的缺陷

（4）软件的复杂性

（5）软件项目的时间压力

（6）程序开发人员的错误

（7）软件项目文档的缺乏

二&、软件缺陷的类别和级别有哪些?

类别描述缺陷所属的类型，以便查找对应开发人员，以及后期缺陷分析。类别通常可以分为以下几种情况：

类别	描述
界面（UI）	界面错误，如界面显示不符合需求、提示信息不合规范等。
功能（Function）	系统功能无效、不相应、不符合需求
性能（Performance）	系统相应过慢、无法承受预期负荷等
安全性（Security）	存在安全隐患的缺陷
数据（Data）	数据导入或设置不正确
其它（Other）	不在上述类别范围的其它错误

按缺陷严重程度划分：

#	缺陷严重等级	描述
1	严重缺陷（Critical）	不能执行正常工作功能或重要功能。或者危及人身安全、系统安全
2	较大缺陷（Major）	严重地影响系统要求或基本功能的实现，且没有办法更正。（重新安装或重新启动该软件不属于更正办法）
3	较小缺陷（Minor	影响系统要求或基本功能的实现，但存在合理的更正办法。（重新安装或重新启动该软件不属于更正办法）
4	轻微缺陷（Trivial）	使操作者不方便或遇到麻烦，但它不影响执行工作功能或重要功能。
5	其它缺陷（Other）	其它错误

按缺陷必须被修复的紧急程度：

#	优先级（Priority）	描述
1	立即解决（Block）	严重阻碍测试进行，且没有方法绕过去
2	高优先级（High）	严重影响测试进行，但是有可选方案绕过该功能进行其它内容测试
3	正常排队（Normal）	缺陷需要正常排队等待修复或列入软件发布清单。
4	低优先级（Low）	缺陷可以在方便时纠正。

三、缺陷严重程度和优先级有什么区别?

严重性：该缺陷的影响范围。空间

优先级：该缺陷修复的紧急程度。时间

一般，严重性程度高的软件缺陷具有较高的优先级，但不总是一一对应。

四、当某一缺陷被反复Reopen时，可能会是哪些原因?

- 上一循环发现功能未实现的缺陷
- 因为其他缺陷的影响导致此缺陷无法验证
- 开发和测试人员未能达成一致

五、分别描述测试充分性、测试覆盖率、测试覆盖域的定义

测试充分性：度量一个给定的测试集T是否能验证软件P满足其需求R。

测试覆盖率：用来度量测试完整性的手段，是一个比率。分为白盒测试覆盖率、灰盒测试覆盖率、 黑盒测试覆盖率。需求覆盖率，代码覆盖率。

测试覆盖域：测试系统覆盖被测试系统的程度。表示测试集覆盖到的代码。

六&、软件测试的原则有哪些？（6种及以上）

（1）完全测试程序是不可能的：eg：软件实现途径太多

（2） 软件测试是有风险的：只能查出尽可能多的bug

（3） 测试无法显示隐藏的软件故障

（4）存在的故障数量与发现的故障数成正比：某些软件故障可能是冰山之巅，程序员犯错。

（5）杀虫剂现象：软件测试进行的越多，其程序免疫力越强的现象。

（6）并非所有软件故障都能修复：没有足够的时间，不值得修复，修复风险太大，不算真正的软件故障。

（7）一般不要丢弃测试用例

（8）应避免测试自己编写的程序

（9）软件测试是一项复杂且具有创造性的和需要高度智慧的挑战性任务

七、软件测试的终止准则有哪些?

（1）基于测试阶段的原则：四个阶段完成。

（2）基于测试用例的原则：测试用例通过率达到要求。

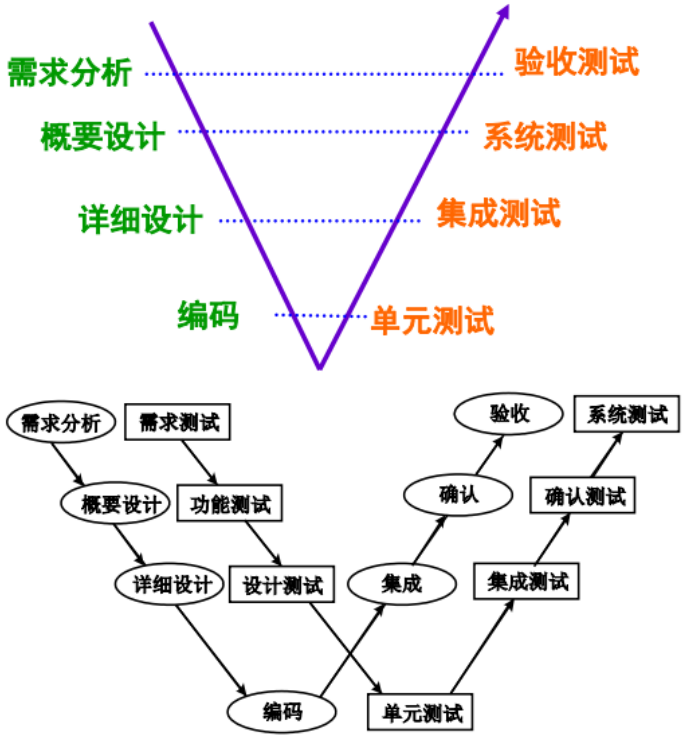
（3）基于缺陷收敛趋势及缺陷修复率原则

（4）基于验收测试的原则：通过验收测试。

（5）基于覆盖率的原则

（6）软件项目暂停或终止，则测试活动也相应暂停或终止

八、V字模型和W模型



V模型：

活动更加并行化，可减少生存周期结束进行测试所需的时间。

进行事先确认，降低最后一刻暴露严重问题的风险。

不足：

忽视了测试对需求分析，系统设计的验证。

没有明确的说明早起的测试，不能体现“尽早地、不断地进行软件测试”的原则

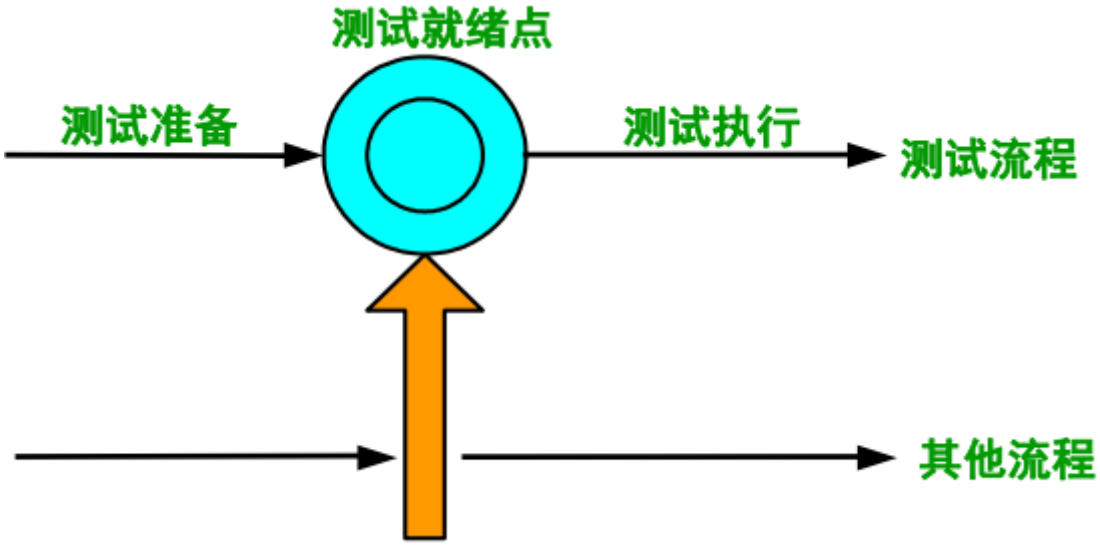
W模型：

- 强调测试伴随着整个软件开发周期。
- 测试的对象不仅仅是程序，需求、功能和设计同样要测试。

不足：开发活动看成是从需求开始到编码结束的串行活动，只有上一阶段完成后，才可以开始下一阶段的活动，不能支持迭代，自发性以及变更调整。

综合：

- V模型和W模型都把软件的开发视为需求、设计、编码等一系列串行的活动。
- 大部分时间内，活动是可以交叉进行的。
- 各层次之间的测试也存在反复触发、迭代和增量关系。V模型和W模型都没有很好地体现测试流程的完整性。
- 相应的测试之间也不存在严格的次序关系。



H模型：

H模型将测试活动完全独立出来，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。

九、为什么说螺旋模型适合于大型复杂系统的开发？

- 项目多期，迭代地进行
- 有助于获取用户需求，加强对需求的理解
- 需求获取、设计、编码和测试活动之间会有大量重叠
- 尽早发现软件中的错误
- 支持需求的动态变化
- 注重风险控制，降低在项目后期发现重大缺陷的风险

十&、软件开发模型在软件开发过程中起到什么作用？

软件开发模型可以直观地描述一个组织在开发某项软件过程中的全部活动，以及在开发过程中的人员组织管理，工作顺序和任务管理、软件质量保证等全部内容。

软件开发过程的主要活动有需求分析，系统设计，程序设计，程序编码，测试，运行维护。以上的各活动都在软件模型中有所体现，并能实时监控。

十一&、软件测试与软件开发有何关系？

- 没有开发过程就没有测试过程。
- 测试过程是为保证开发过程的产出进行验证和确认的一系列活动。
- 不同的软件开发过程模型中，测试在其中所处的位置不同。

瀑布模型：测试是一种开发后的活动。

螺旋模型：前进的一步，每个增量版本都可以单独测试。

十二、单元测试有哪些内容?测试中采用什么方法?

（1）单元测试内容（哪些方面）：

```
+ 模块接口测试：进出程序单元的数据流是否正确。个数、属性、顺序等等。
+ 局部数据结构测试：
+ 路径测试：不正确计算，比较和控制流错误。
+ 错误处理测试：模块的错误处理功能是否有错误或缺陷。描述，代码。
+ 边界测试
```

（2）单元测试类型（哪些方面）：

- 逻辑单元测试：针对单个方法进行代码正确性检查。
- 集成单元测试：针对组件之间的交互进行代码正确性检查。
- 功能单元测试：将集成单元测试的边界进行扩展，以确保正确地激发响应。

（3）测试的方法：

为被测试模块编写驱动模块和桩模块来实现被测试单元的可运行。

通过驱动模块来模拟被测试模块的上级调用模块，以上级模块调用被测模块的格式驱动被测模块，

接收被测模块的测试结构并输出。桩模块则用来代替被测试模块所调用的模块。它的作用是返回被测模块所需的信息。

十三、集成测试有哪些内容?测试中采用什么方法（应用方面）？

（1）集成测试方法：

- **非增式测试：**（配备辅助模块，先对所有模块进行个别的单元测试。）

  分散测试，再集中起来一次完成集成测试。

  如果在模块的接口处存在错误，只会在最后的集成时一下子暴露出来，便于找出问题和修改。

- **增式测试：**（较少的辅助模块，减少了辅助性测试工作）逐步实现测试
  - 自顶向下增式测试表示逐步集成和逐步测试是按结构图自上而下进行的。
    - 1、以主控模块作为测试驱动模块，把对主控模块进行单元测试时引入的所有桩模块用实际模块替代。（深度优先）
    - 2、依据所选的集成策略（深度优先或广度优先），每次只替代一个桩模块。
    - 3、 每集成一个模块立即测试一遍。
    - 4、 只有每组测试完成后，才着手替换下一个桩模块
    - 为避免引入新错误，须不断地进行回归测试。从第二步开始，循环执行上述步骤，直至整个程序结构构造完毕。
  - 自底向上增式测试表示逐步集成和逐步测试是按结构图自下而上进行的。
    - 1、把低层模块组织成实现某个子功能的模块簇（逆向广度优先）
    - 2、开发一个测试驱动模块，控制测试数据的输入和测试结果的输出。
    - 3、对每个模块簇进行测试。
    - 4、删除测试使用的驱动模块，用较高层模块把模块群组织成为完成更大功能的新模块群。
    - 从第二步开始循环执行上述各步骤，直至整个程序构造完毕。

十四、正式审查及其方法，方面有哪些?

正式审查：进行静态白盒测试的过程。（含义很广）

正式审查四个基本要素：

- 确定问题：找出软件的问题，不仅是出错的项目，还包括遗漏的项目。
- 遵守规则：遵守固定的规则，要审查的代码量、花费时间、审查的内容等。
- 准备：每一个参与者都为审查做准备，并尽自己的力量。
- 编写报告：做出审查结果的书面总结报告，便于开发小组的成员使用。

正式审查的方法：

- 同事审查：（要求最低）同事之间
- 走查：以小组方式进行，更技术，发现问题并记录。
- 检验（技术评审）：（最正式）由开发组，测试组和相关人员联合，综合运用走查，审查技术，逐行逐段的检查软件。

  表述者不是原来的编写代码的程序员。

代码审查涵盖方面：

- 业务逻辑的审查：按照规格说明书，逻辑思路是否清晰。
- 算法的效率
- 代码风格
- 编程规则：变量定义，语句完整，函数调用，内存管理，逻辑表达，运行稳定等。

十五、静态黑盒测试的内容

开发文档：软件需求说明书、数据库设计说明书、概要设计说明书、详细设计说明书、可行性研究报告。

用户文档：用户手册、操作手册、维护修改建议。

管理文档：项目开发计划、测试计划、测试报告、开发进度月报、开发总结报告。

需求文档测试：

需求明确

测试人员、开发人员、需求方对需求达成一致

需求文档规范：

- 正确性：对照原始需求规格说明书
- 必要性：不能回朔到出处的需求项可能是多余的
- 优先级：恰当地划分并标识
- 明确性：不能使用含糊的词汇
- 可测性：每项需求都必须是可验证的
- 一致性：内容前后一致
- 可修改性：良好的组织结构

用户文档测试：

- 文档走查：由测试部门进行静态走查，按错误类型对文档分块设计检查表进行测试。
- 数据校对：测试部门静态走查，按需求文档、设计文档、测试报告、用户文档结构要求设计检查表。
- 流程操作：模拟实际操作流程验证用户文档。安装 / 卸载操作过程、参数配置、功能操作和向导功能。
- 可用性测试：安排专家、技服或用户来验证用户文档。

时间上，程序测试完成后进行或与程序测试同时进行。

任务分派策略：

- **与程序测试一同进行：**测试者任务重。测试效率高，资源可以共享，但测试周期可能加长。
- **与程序测试分开进行：**专人进行文档测试。测试效果好，周期短，但不利于共享资源，增加资源占用量，导致成本增加。
- **不同类型的测试项，分派给不同的测试人员：**语言类错误和版面类错误不必由测试相关程序的人员承担。逻辑类错误、一致性错误的联机文档功能错误，由相关程序测试的人员完成。

产品说明书测试：

产品说明书：站在客户的角度讲产品功能。（静态，黑盒）

产品规格说明书：偏向于软件的概要设计。

检查产品说明书：

- **开始测试：** 产品说明书。
- **高级审查：**
  - 假设自己是客户：了解用户，研究用户所想，联系市场、销售人员。
  - 研究现有的标准和规范：行业的标准，公司的惯用语，安全标准，政府标准，图形界面。
  - 审查和测试类似软件：规模，复杂性，测试性，质量和可靠性。
- **详细审查：**
  - 说明书属性清单（8个）：完整，准确，精确、不含糊、清晰，一致，贴切，合理，代码无关，可测试性。
  - 说明书术语检查清单：总是，没有，明显，某些，通常，因此，等等，跳过。

十六、为什么需要测试用例?

- 在开始测试之前设计好用例，避免盲目测试并提高测试效率，减少测试的不完全性。
- 令软件测试的实施重点突出、目的明确。
- 测试用例的多少和执行难度，估算工作量，便于测试项目的时间和资源管理与跟踪。
- 减少回归测试的复杂程度，更新后只需修正少量用例便可，降低强度、缩短项目周期。
- 功能模块的测试用例的通用化和复用化则会使软件测试易于开展，效率提升。
- 根据测试用例的操作步骤和执行结果，为分析软件缺陷和程序模块质量提供依据。
- 方便书写软件测试缺陷报告。
- 根据测试用例的执行等级，实施不同级别的测试。

  总结：软件测试是有组织性、步骤性和计划性的，为了能将软件测试的行为转换为可管理的、具体量化的模式，需要创建和维护测试用例。

测试用例四性：代表性，针对性，可判定性，可重现性。

测试用例的组成元素（5W1H1E）：测试目标、测试对象、测试环境要求、测试前提、输入数据、操作步骤和预期结果。

还有例如：测试用例编号、名称、设计者，软件版本号，参考信息。

十七、变异测试的流程

变异耦合效应假设：指若测试用例能够杀死简单变异体，那么该测试用例也易于杀死复杂变异体。

变异的定义：

程序变异指基于预先定义的变异操作对程序进行修改，进而得到源程序变异程序（也称为变异体）的过程。

- 当源程序与变异程序存在执行差异时，则认为该测试用例检测到变异程序中的错误，变异程序被杀死。
- 当两个程序不存在执行差异时，则认为该测试用例没有检测到变异程序中的错误，变异程序存活。

变异测试过程：

程序与变异程序的执行差异主要表现为以下两个情形:

(1)执行同一测试用例时，源程序和变异程序产生了不同的运行状态

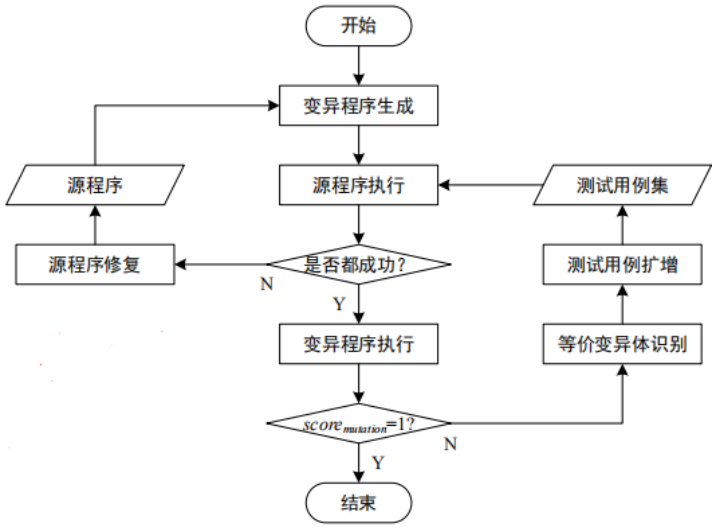
(2)执行同一测试用例时，源程序和变异程序产生了不同的执行结果

根据满足执行差异要求的不同，可将变异测试分为弱变异测试(WeakMutation Testing)和强变异测试(Strong Mutation Testing)

弱变异测试：当情形（1）出现时就可认为变异程序被杀死。

强变异测试：只有情形(1)和(2)同时满足才可认为变异程序被杀死。

变异测试基本流程图：



变异得分: score.mutation。非等价变异程序数量: num.equivalent

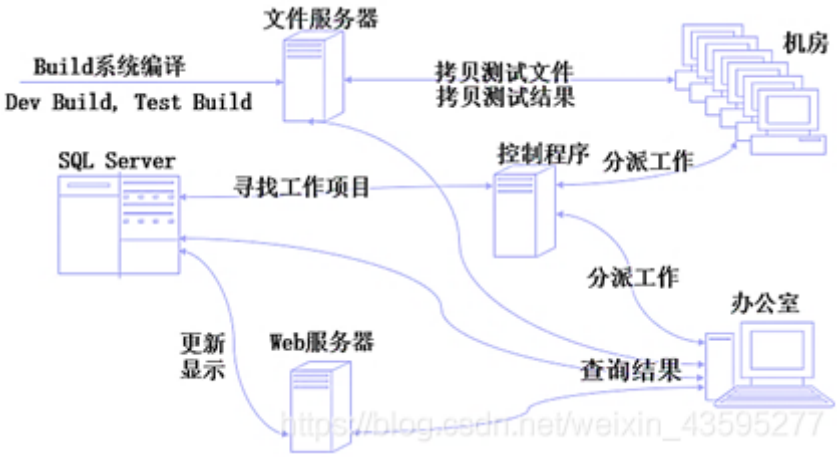
当$score_{mutation}$的值为0时，表明测试用例集没有杀死任何一个变异程序

当$score_{mutation}$的值为1时，表明测试用例集杀死了所有非等价变异程序

$$score_{mutation} = \frac{num_{killed}}{num_{total} - num_{equivalent}}$$

十八、自动化测试优缺点?

自动化测试的基本结构：



优点：

- 对程序回归测试更方便，尤其是程序修改比较频繁的情况。
- 建立可靠、重复的测试，减少人为失误，更好地利用资源。
- 增强测试质量和覆盖率。
- 执行手工测试不可能完成的任务。

缺点：

- 不能取代手工测试。
- 发现的问题和缺陷比手工测试要少。
- 不能用于测试周期很短的项目、不能保证100%的测试覆盖率、不能测试不稳定的软件和软件易用性等。

十九&、自动化测试与测试自动化有什么区别?

自动化测试定义：

由手工逐个地运行测试用例的操作过程被测试工具自动执行的过程所代替。

自动化测试的主要特征：测试工具的使用。

区别：

- 自动化测试焦点集中在**测试执行**。主要是由测试工具替代手动自动地完成测试。
- 测试自动化指一切可以由计算机系统自动完成的**测试任务**都已经**由计算机系统**或软件工具、程序来**承担并自动执行**。

自动化测试：测试工具，测试执行，单项活动。

测试自动化：是一种理念，全过程的，所有测试活动，包括测试设计和测试管理。

二十&、软件测试度量是出于什么原因才进行的？是不可或缺的吗？

目的：改进软件测试的质量，提高测试效率。

软件测试的度量是为了项目质量服务的，选择的度量标准和方法都是为了能让测试进行得更加科学和规范，以创造更多的测试价值。

二十一&、软件测试度量涉及哪几个关键问题？

软件测试度量的目的。

软件测试度量内容：进度(时间)度量，成本度量，规模度量，

测试质量(效率)度量，度量Bug的数量，测试覆盖率，

测试覆盖率:

代码行覆盖率：代码行覆盖率=（已执行测试的代码行/总的代码行*100%

功能模块覆盖率：功能覆盖率=（已执行测试的功能模块数/总的功能模块数）*100%

数据库覆盖率：数据库覆盖率=（SQL中出现的数据库的对象数/数据库总的对象数）*100%

需求覆盖率：需求覆盖率=（被验证到的需求数量/总的需求数量）*100%

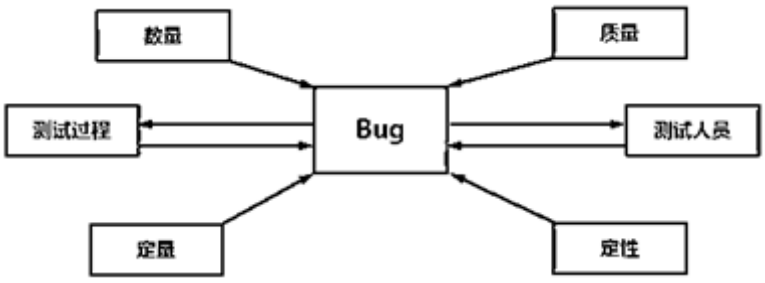
产品质量度量：发现的缺陷数量

二十二&、软件测试度量应遵循哪些重要的原则或方针？

测试的度量的原则：

- 要制定明确的度量目标；
- 度量标准的定义应该具有一致性、客观性；
- 度量方法应该尽可能简单、可计算；
- 度量数据的收集应该尽可能自动化。

二十三、bug综合评价模型包含哪些方面？



测试过程中发现的bug，测试人员对其中的bug进行发现、筛选、加权，可以得到最后的一个bug评估模型。

二十四&、软件测试分为哪几个阶段?

（1） 测试需求的分析和确定

（2） 测试计划

（3）测试设计

（4）测试执行

（5）测试记录和缺陷跟踪

（6）回归测试

（7）测试总结报告

二十五&、需要从哪几个方面对测试需求进行评审?

检查要点：

- **正确性**：对照原始需求检查需求规格说明书。
- **必要性**：检查是否存在多余的需求项。
- **优先级**：检查是否恰当地划分并标识需求项。
- **明确性**：不使用含糊的词汇。
- **可测性**：每项需求都必须是可验证的，输入输出格式，软件系统的约束条件是否完整描述。
- **完整性**：找到用户的原始需求素材来进行对照检查。
- **可修改性**：检查是否具有良好的组织结构，良好的组织结构使需求易于修改。
- **一致性**：主要检查需求是否自相矛盾，包括需求描述的前后不一致和需求规格说明书和原始需求的不一致。

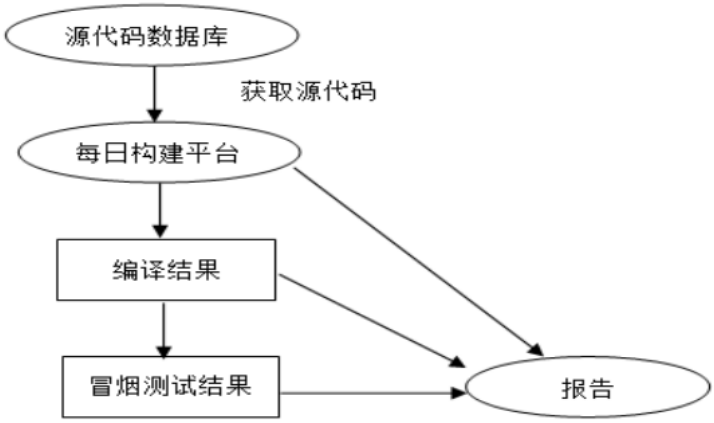
检查步骤：

- 获取最新版本的软件需求规格说明书，同时尽量取得用户原始需求文档。
- 阅读和尝试理解需求规格说明书中描述的所有需求项。
- 对照需求规格说明书检查列表进行检查并记录。
- 针对检查结果进行讨论、修订需求规格说明书后回到第一步，直到检查列表中的所有项通过。

二十六&、请说明测试执行过程中所要做的主要工作

- 测试用例的合理选择与测试分工
- 测试环境的搭建：数据，外部设备，参数，操作系统，网络。
- BVT测试与冒烟测试：是检查程序是否完整，是否实现了最基本的可测试性要求。这两种测试是所有正式测试执行之前的第一个步骤。
- 每日构建的基本流程：每天定时把所有文件编译、连接、组成一个可执行的程序的过程。（利用空余时间）

每日构建流程图：



二十七&、软件测试对工作人员有什么要求?

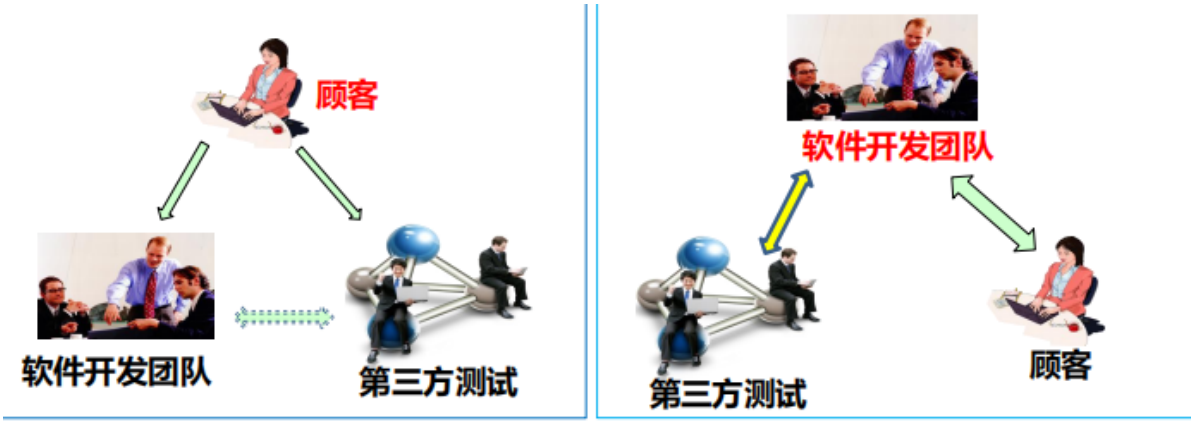
良好的沟通能力，扎实的工作作风，全面的技术基础，高级的综合素质，工作细致认真，

二十八&、请说明什么是第三方测试

第三方软件测试的模式：

用户主导模式：例如开发团队为某公司开发产品后，公司请第三方测试来测试产品。此时，软件开发团队与第三方测试无利益关系。

开发团队主导：开发团队在推出产品之前，聘请第三方测试，为开发团队测试产品。



模式一　　模式二

第三方软件测试的定义与实施主体：

由开发者和用户以外的第三方进行的软件测试，其目的是为了保证测试的客观性。

- 狭义：独立的第三方测试机构，软件评测中心，企业、
- 广义：非本软件的开发人员，QA部门人员测试、公司内部交叉测试。

第三方测试的职责：

- 验证软件是否符合需求和设计。
- 检测错误。
- 对错误进行分类分析，将分析结果反馈给开发人员以改进软件过程管理。

请简述第三方测试的意义：

- 弥补项目组缺少专业测试经验的问题。
- 补充项目业务人员参与项目测试精力不足问题；
- 实现开发方、用户方、测试三方的权责分开，避免出现用户、开发方双方纠缠不清的矛盾，使得许多问题能得到比较客观的处理
- 引入权威的第三方测试可以降低项目失败的风险。
- 引入第三方测试团队可以提高系统稳定性，避免出现系统测试不到位，系统带严重缺陷上线。

第三方测试的核心内容：

按合同条款与系统需求说明书对工程项目进行全面质量验收评测，验证是否满足需求，功能实现与性能指标是否达到业主的要求。

二十九&、请简述第三方测试的测试过程及范围

过程：

- **制定测试计划：**功能，环境，范围，测试观点，工具，用例，缺陷预估，日程。
- 测试观点设计与Review的要求：
    - 测试数据设计是否合理(等价类划分，因果图法等)
    - 预期测试结果是否正确
    - 各种条件组合是否考虑
    - 自动化测试用的脚本是否正确
- **实施过程中的故障处理：**
    - 每天及时记录测试过程
    - 给开发部门提交测试发现的问题点
    - BUG修正后的确认测试
- **测试总结报告**（给开发方）：
    - 测试整体情况说明性文档
    - 测试整体结果分析：例如缺陷严重程度，在各个功能的分布状况等， 缺陷的分类分析
    - 软件质量方面的改善建议
    - 测试指标(BUG检测率等)
- **测试总结报告**（给第三方测试）：
    - 主要分析测试过程中的不足、原因以及改善的方法。
    - 总结本次测试比较成功的地方。

第三方软件测试的涵盖测试范围：

- 各个测试阶段（单元测试通常由开发方实施）：黑白，集成，系统，验收。
- 软件：功能性，易用性，容错性，安全性，性能。
- 文档：正确性与一致性。

三、测试用例的设计

（一）黑盒测试用例设计方法

等价类划分，边界值划分，错误推测法，因果图法，正交表试验法，场景图，功能图。

等价类方法

（1）概述

选择适当的数据子集代表整个数据集，通过降低测试的数目去实现合理覆盖，覆盖了更多的可能数据，以发现更多的软件缺陷。

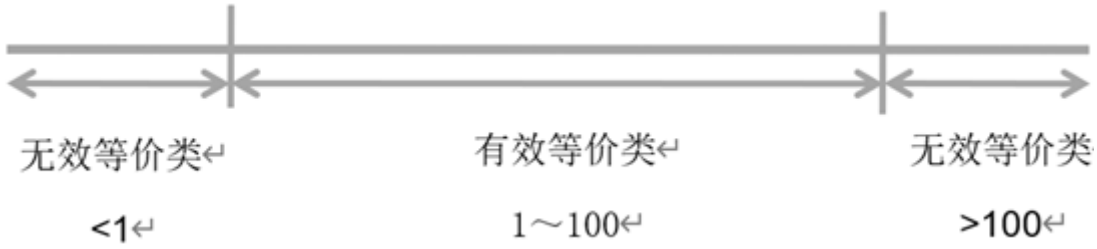
将程序所有可能的输入数据（有效的和无效的）划分成若干个等价类。然后从每个部分中选取具有代表性的数据作为测试用例进行合理的分类。

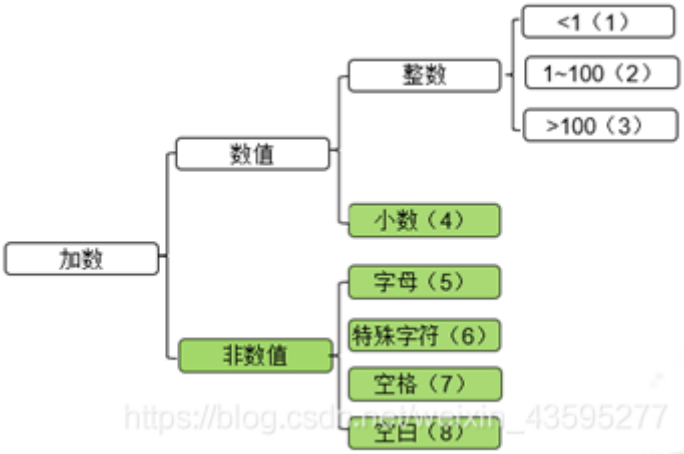
（2）等价类和等价类划分

等价类是指某个输入域的子集合。

等价类划分为：

有效等价类：指符合《需求规格说明书》，输入合理的数据集合。
无效等价类：指不符合《需求规格说明书》，输入不合理的数据集合。

> **示例：计算两个1～100之间整数的和。**
>
> 等价类划分：
>
> 
>
>
> 将输入域分成一个有效等价类（1～100）和两个无效等价类（<1,>100），并为每一个等价类进行编号，然后就可以从每一个等价类中选取一个代表性的数据来测试，设计如下表所示的测试用例：
>
编号	所属等价类	加数1	加数2	和
1	2（有效等价类）	3	40	43
2	1（无效等价类）	0	-1	提示"请输入1~100之间的整数"
3	3（无效等价类）	110	101	提示"请输入1~100之间的整数"
>
> 刚才的等价类还不完善，只考虑了输入数据的范围，没有考虑输入数据的类型（我们认为只输入数据，可是最终用户输入什么都有可能）。综合考虑输入数据的类型和范围划分等价类，如下图所示：



等价类划分的测试用例，最终即为：

编号	所属等价类	加数1	加数2	结果
1	2（有效等价类）	3	40	43
2	1（无效等价类）	0	-1	提示“请输入1~100之间的整数”
3	3（无效等价类）	110	101	提示“请输入1~100之间的整数”
4	4（无效等价类）	1.2	1.2	提示“请输入1~100之间的整数”
5	5（无效等价类）	A	B	提示“请输入1~100之间的整数”
6	6（无效等价类）	@	%	提示“请输入1~100之间的整数”
7	7（无效等价类）	空格	空格	提示“请输入1~100之间的整数”
8	8（无效等价类）			提示“请输入1~100之间的整数”

（3）等价类划分的步骤

- 先考虑输入数据的数据类型（合法和非法的）
- 再考虑数据范围（合法类型中的合法区间和非法区间）
- 画出示意图，区分等价类
- 为每一个等价类编号
- 从一个等价类中选择一个测试数据构造测试用例

（4）等价类实例

有一个档案管理系统，要求用户输入以年月表示的日期。

条件：日期限定在1990年1月~2049年12月，并规定日期由6位数字字符组成，前4位表示年，后2位表示月。

测试用例：

a. 划分等价类并编号,下表为等价类划分的结果:

输入等价类	有效等价类	无效等价类
日期的类型及长度	①6位数字字符	②有非数字字符 ③少于6位数字字符 ④多于6位数字字符
年份范围	⑤在1990～2049之间	⑥小于1990 ⑦大于2049
月份范围	⑧在01～12之间	⑨等于00 ⑩大于12

b. 设计测试用例，以便覆盖所有的有效等价类。表4列出了3个有效等价类，编号分别为①、⑤、⑧，设计的测试用例如下：

测试数据	期望结果	覆盖的有效等价类
200211	输入有效	①、⑤、⑧

c. 为每一个无效等价类设计一个测试用例，设计结果如下：

测试数据	期望结果	覆盖的无效等价类
95June	无效输入	②
20036	无效输入	③
2001006	无效输入	④
198912	无效输入	⑥
205001	无效输入	⑦
200100	无效输入	⑨
200113	无效输入	⑩

边界值方法

（1）概述

边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。通常边界值分析法作为对等价类划分法的补充，其测试用例来自等价类的边界。

（2）设计方法

- 确定边界情况（输入或输出等价类的边界）。
- 选取正好等于、刚刚大于或刚刚小于边界值作为测试数据。

（3）边界值方法实例：

计算两个1～100之间整数的和。

输入要求是1～100的整数，自然产生了1和100两个边界。设计测试用例时，重点考虑这两个边界问题。

根据边界值方法，测试用例修改如下：

编号	所属等价类	加数1	加数2	预期结果
1	2（有效等价类）	1	1	2
2		100	100	200
3	1（无效等价类）	0	0	提示“请输入1~100之间的整数”
4	3（无效等价类）	101	101	提示“请输入1~100之间的整数”
5	4（无效等价类）	1.2	1.2	提示“请输入1~100之间的整数”
6	5（无效等价类）	A	B	提示“请输入1~100之间的整数”
7	6（无效等价类）	@	%	提示“请输入1~100之间的整数”
8	7（无效等价类）	空格	空格	提示“请输入1~100之间的整数”
9	8（无效等价类）			提示“请输入1~100之间的整数”

https://blog.csdn.net/weixin_

（4）边界值与等价划分的区别

- 边界值分析不是从某等价类中随便挑一个作为代表，这个等价类的每个边界都要作为测试条件。
- 边界值分析不仅考虑输入条件，还要考虑输出空间产生的测试情况。
- 等价划分属于确认有有效区间，边界值属于确认边界。

（5）常见的边界值

- 文本框接受字符个数，比如用户名长度，密码长度等。
- 报表的第一行和最后一行。
- 数组元素的第一个和最后一个。
- 循环的第 1 次、第 2 次和倒数第 2 次、最后一次。

等价类和边界值的综合示例（ppt13讲）

某保险公司保费计算方式为投保额*保险率，保险率又依点数不同而有差别，10点以上费率为0.6%，10点以下费率为0.1%。保险率和以下参数有关：

- 年龄：数字 0-150
- 性别：字符组合，区分大小写
- 婚姻：字符组合
- 扶养人：数字 1-9人（选填项）

选项	参数	点数
年龄	20～39岁	6点
	40～59岁	4点
	60岁以上，20岁以下	2点
性别	MALE	5点
	FEMALE	3点
婚姻	已婚	3点
	未婚	5点
扶养人数	一人扣0.5点最多扣3点(四舍五入取整数)	

输入	输入条件	有效等价类	无效等价类
年龄	非负整数	非负整数 (1)	负整数 (7) 小数 (8) 字母 (9) 特殊字符 (10)
	0～150	00～19 (2) 20～39 (3) 40～59 (4) 60～150 (5)	<0 (11) >150 (12)
	必填	填 (6)	不填 (13)
性别	字符组合	字符组合 (1)	非字符组合 (6)
	区分大小 写	大写 (2)	小写 (7) 大小写混合 (8)
	MALE 或 FEMALE	MALE (3) FEMALE (4)	非MALE、FEMALE (9)
	必填	填 (5)	不填 (10)
婚姻	字符组合	字符组合 (1)	非字符组合 (5)
	已婚或未婚	已婚 (2) 未婚 (3)	非已婚、未婚 (6)
	必填	填 (4)	不填 (7)
抚养人数	正整数	正整数 (1)	非正整数 (6) 小数 (7) 字母 (8) 特殊字符 (9)
	1～9	1～6 (2) 7～9 (3)	<1 (10) >9 (11)
	选填	填 (4) 不填 (5)	

输入	有效值	无效值
年龄	15 (覆盖1、2、6) 25 (覆盖1、3、6) 50 (覆盖1、4、6) 80 (覆盖1、5、6) 边界值：0、19、20、39、40、59、60、150	-20 (覆盖7) 15.5 (覆盖8) a (覆盖9) & (覆盖10) -999.5 (覆盖11) 180 (覆盖12) 不填 (覆盖13) 边界值： -1、151
性别	MALE (覆盖1、2、3、5) FEMALE (覆盖1、2、4)	6553 (覆盖6) male (覆盖7) female (覆盖8) 男 (覆盖9) 不填 (覆盖10)
婚姻	已婚 (覆盖1、2、4) 未婚 (覆盖1、3、4)	1234 (覆盖5) 离婚 (覆盖6) 不填 (覆盖7)
抚养人数	5 (覆盖1、2、4) 8 (覆盖1、3、4) 不填 (覆盖3) 边界值：1、6、7、9	-6 (覆盖6) 5.1 (覆盖7) A (覆盖8) & (覆盖9) -100 (覆盖10) 100 (覆盖11) 边界值：0、10

因果图法

(1) 概述

因果图法比较适合输入条件比较多的情况，测试所有的输入条件的排列组合。所谓的原因就是输入，结果就是输出。

(2) 测试用例设计步骤

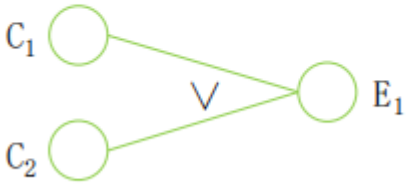
- 分析程序规格说明的描述中，哪些是原因，哪些是结果。原因常常是输入条件或输入条件的等价类，而结果是输出条件。
- 分析程序规格说明的描述中的语义内容，并将其表示成连接各个原因与各个结果的“因果图”。
- 标明约束条件。由于语法或环境的限制，有些原因和结果的组合情况是不可能出现的。
- 把因果图转换成判定表（决策表）。
- 为判定表中的每一列表示的情况设计测试用例。

(3) 因果图基本图形符号

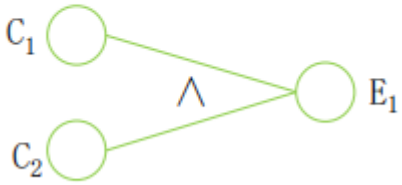
- 恒等：若原因出现，则结果出现；若原因不出现，则结果不出现。
- 非（～）：若原因出现，则结果不出现；若原因不出现，则结果出现。
- 或（∨）：若几个原因中有一个出现，则结果出现；若几个原因都不出现，则结果不出现。
- 与（∧）：若几个原因都出现，结果才出现；若其中有一个原因不出现，则结果不出现。

（a）恒等

（b）非



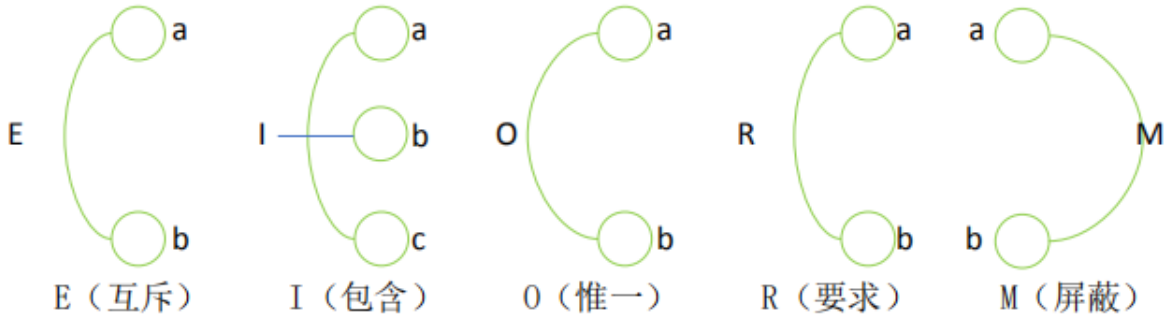
（c）或



（c）与

（4）因果图的约束符号

- E（互斥）：表示两个原因不会同时成立，两个中最多有一个可能成立。
- I（包含）：表示三个原因中至少有一个必须成立。
- O（惟一）：表示两个原因中必须有一个，且仅有一个成立。
- R（要求）：表示两个原因，a出现时，b也必须出现，a出现时，b不可能不出现。
- M（屏蔽）：两个结果，a为1时，b必须是0，当a为0时，b值不定。



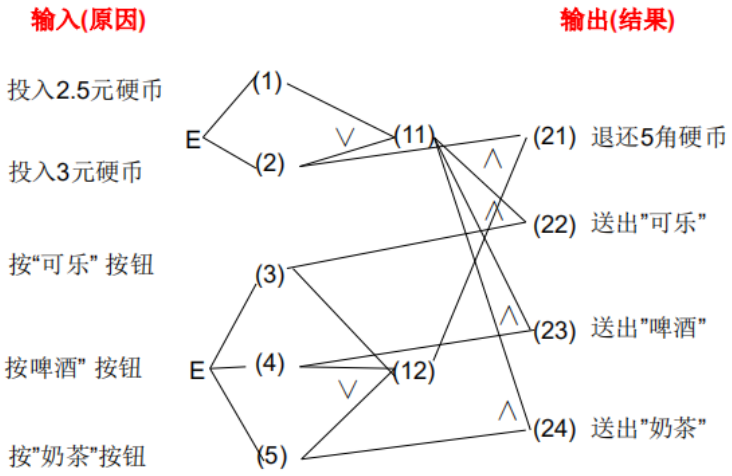
E（互斥） I（包含） O（惟一） R（要求） M（屏蔽）

（5）因果图测试用例

有一个处理单价为2.5元的盒装饮料的自动售货机软件。若投入2.5元硬币，按“可乐”、“啤酒”、或“奶茶”按钮，相应的饮料就送出来。若投入的是3元硬币，在送出饮料的同时退还5角硬币。

分析这一段说明，我们可列出原因和结果：

> 原因(输入)：
> ① 投入2.5元硬币；
> ② 投入3元；
> ③ 按“可乐”按钮；
> ④ 按“啤酒”按钮；
> ⑤ 按“奶茶”按钮。
>
> 中间状态： ① 已投币；②已按钮
>
> 结果(输出)：
> ① 退还5角硬币；
> ② 送出“可乐”饮料；
> ③ 送出“啤酒”饮料；
> ④ 送出“奶茶”饮料；
>
> a. 根据原因和结果,可以设计这样一个因果图：



b. 因果图转换为判定表：
共有十一种输入，即3*4-1

c. 根据判定表设计测试用例

正交试验设计法

（1）概述

根据正交性从全面试验中挑选出部分有代表性的点进行试验，这些有代表性的点具备了“均匀分散、齐整可比”的特点，正交试验是一种高效率、快速、经济的实验设计方法。

利用正交实验设计方法设计测试用例， 可以控制生成的测试用例数量；设计的测试用例也具有一定的覆盖率和代表性。

（2）正交表的构成

- 行数：正交表中的行的数量，即试验的次数，也是通过正交实验法设计的测试用例的个数。
- 因素数：正交表中列的数量，即要测试的功能点。
- 水平数：任何单个因素能够取得的值的最大个数，即要测试功能点的取值个数。

正交表的形式：$L_{行数}(水平数^{因素数})$ 如：$L_8(2^7)$

正交表$L_n(s^c)$计算公式：$n=c\times s-c+1$

（3）用正交表设计测试用例的步骤

- 有哪些因素（功能点），每个因素有哪几个水平（功能点的取值）
- 选择一个合适的正交表
  - 考虑因素（功能点）的个数，因素水平（功能点的取值）的个数。
  - 考虑正交表的行数，取行数最少的一个。
- 把变量的值映射到表中
- 把每一行的各因素水平的组合作为一个测试用例
- 加上你认为可疑且没有在表中出现的组合

（4）设计测试用例的三种情况

- 因素数(变量)、水平数(变量值)相符：因素数与水平数刚好符合正交表。
- 因素数不相同：如果因素数不同的话，可以采用包含的方法，在正交表公式中找到包含该情况的公式，如果有N个符合条件的公式，那么选取行数最少的公式。
- 水平数不相同：采用包含和组合的方法选取合适的正交表公式。

（5）案例一

视图选项卡上的“显示/隐藏”组中有3个可用选项：

- 有3个因素：网格线、编辑栏、标题
- 每个因素有2个水平：选与不选

> **a. 选择正交表的分析**
>
> 表中的因素数>=3；
>
> 表中至少有3个因素数的水平数>=2；
>
> 行数取最少的一个。
>
> 从正交表公式中开始查找，结果为：L4(23)
>
> 利用正交表设计测试用例，我们得到的测试用例个数是n=3*(2-1)+1=4（这个公式就是(因素数*(最大水平数-1)+1)），对于三因素两水平的刚好有L4(2^3)的正交表可以套用。
>
> **b. 正交表变量的映射**
>
> - 网格线：0 → 选，1 → 不选
> - 编辑栏：0 → 选，1 → 不选
> - 标 题：0 → 选，1 → 不选

		列号		
		1	2	3
行号	1	0	0	0
	2	0	1	1
	3	1	0	1
	4	1	1	0

→

		列号		
		网格线	编辑栏	标题
行号	1	选	选	选
	2	选	不选	不选
	3	不选	选	不选
	4	不选	不选	选

> **c. 测试用例**
>
> 1. 选中网格线、选中编辑栏、选中标题
> 2. 选中网格线、不选编辑栏、不选标题
> 3. 不选网格线、选中编辑栏、不选标题
> 4. 不选网格线、不选编辑栏、选中标题
>
>   增补测试用例
> 5. 不选风格线、不选编辑栏、不选标题
>
>   测试用例的减少数：8 → 5

（6）案例二

根据PowerPoint的打印功能的描述设计测试用例，功能描述如下：

打印范围分：全部、当前幻灯片、给定范围

打印内容分：幻灯片、讲义、备注页、大纲视图

打印颜色/灰度分：颜色、灰度、黑白

打印效果分：幻灯片加框、幻灯片不加框

a. 案例分析

根据以上提到的功能说明，构造因子状态表，得到因子状态：

状态/因素	A打印范围	B打印内容	C打印颜色/灰度	D打印效果
0	全部	幻灯片	颜色	幻灯片加框
1	当前幻灯片	讲义	灰度	幻灯片不加框
2	给定范围	备注页	黑白	
3		大纲视图		

将中文字转换成字母的因子状态表：

状态/因素	A	B	C	D
0	A1	B1	C1	D1
1	A2	B2	C2	D2
2	A3	B3	C3	
3		B4		

选择正交表的分析：

1. 表中的因素数>=4；
2. 表中至少有4个因素数的水平数>=2；
3. 行数取最少的一个。
4. 从正交表公式中开始查找，结果为： L16(45)

注：此案例中有四个被测对象，每个被测对象的状态都不一样。

b. 正交表

状态/因素	1	2	3	4	5
1	0	0	0	0	0
2	0	1	1	1	1
3	0	2	2	2	2
4	0	3	3	3	3
5	1	0	1	2	3
6	1	1	0	3	2
7	1	2	3	0	1
8	1	3	2	1	0
9	2	0	2	3	1
10	2	1	3	2	0
11	2	2	0	1	3
12	2	3	1	0	2
13	3	0	3	1	2
14	3	1	2	0	3
15	3	2	1	3	0
16	3	3	0	2	1

https://blog.csdn.net/weixin_43595277

c. 用字母代替的正交表

状态/因素	1	2	3	4	5
1	A1	B1	C1	D1	0
2	A1	B2	C2	D2	1
3	A1	B3	C3	D1	2
4	A1	B4	C1	D2	3
5	A2	B1	C2	D1	3
6	A2	B2	C1	D2	2
7	A2	B3	C2	D1	1
8	A2	B4	C3	D2	0
9	A3	B1	C3	D2	1
10	A3	B2	C3	D1	0
11	A3	B3	C1	D2	3
12	A3	B4	C2	D1	2
13	A1	B1	C1	D2	2
14	A2	B2	C3	D1	3
15	A3	B3	C2	D2	0
16	A1	B4	C1	D1	1

**

通过分析：第5列没有意义可以删掉，由于四个因素里有三个的水平值小于3，所以从第13行到16行的测试用例可以忽略。

d. 测试用例

状态/因子	打印范围	打印内容	打印颜色/灰度	打印效果
1	A1全部	B1幻灯片	C1颜色	D1幻灯片加框
2	A1全部	B2讲义	C2灰度	D2幻灯片不加框
3	A1全部	B3备注页	C3黑白	D1幻灯片加框
4	A1全部	B4大纲视图	C3黑白	D2幻灯片不加框
5	A2当前幻灯片	B1幻灯片	C2灰度	D1幻灯片加框
6	A2当前幻灯片	B2讲义	C3黑白	D1幻灯片加框
7	A2当前幻灯片	B3备注页	C1颜色	D2幻灯片不加框
8	A2当前幻灯片	B4大纲视图	C1颜色	D1幻灯片加框
9	A3给定范围	B1幻灯片	C3黑白	D2幻灯片不加框
10	A3给定范围	B2讲义	C1颜色	D1幻灯片加框
11	A3给定范围	B3备注页	C2灰度	D1幻灯片加框
12	A3给定范围	B4大纲视图	C2灰度	D2幻灯片不加框

场景图设计法

用例场景是用来描述流经用例路径的过程，这个过程从开始到结束遍历用例中所有基本流和备选流。

流程图法

算法流程图是针对程序的内部结构的，而黑盒测试的流程图是针对整个系统业务功能流程的。

流程图法的步骤：

- 详细了解需求。
- 根据需求说明或界面原型，找出业务流程的各个页面以及各页面之间的流转关系。
- 画出业务流程。
- 写用例，覆盖所有的路径分支。

（二）白盒测试用例的设计

白盒测试常用测试用例设计方法

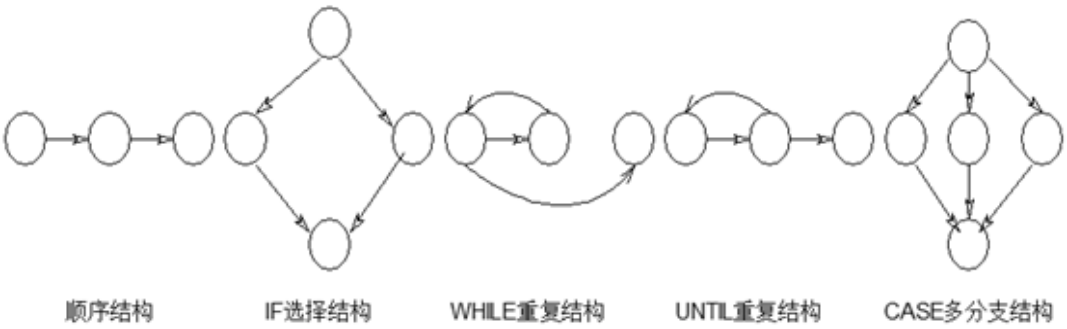
逻辑覆盖法（逻辑驱动测试）。

基本路径测试方法。

白盒测试的基本概念

（1）控制流图

- 控制流图（可简称流图）是对程序流程图进行简化后得到的，它可以更加突出的表示程序控制流的结构。
- 控制流图中包括两种图形符号：节点和控制流线。
  - 节点由带标号的圆圈表示，可代表一个或多个语句、一个处理框序列和一个条件判定框（假设不包含复合条件）。
  - 控制流线由带箭头的弧或线表示，可称为边。它代表程序中的控制流。



其中，包含条件的节点被称为判定节点（也叫谓词节点），由判定节点发出的边必须终止于某一个节点，由边和节点所限定的范围被称为区域。

- 对于复合条件，则可将其分解为多个单个条件，并映射成控制流图。

（2）环形复杂度

环形复杂度也称为圈复杂度，它是一种为程序逻辑复杂度提供定量尺度的软件度量。

环形复杂度的应用——可以将环形复杂度用于基本路径方法，它可以提供：程序基本集的独立路径数量；确保所有语句至少执行一次的测试数量的上界。

独立路径是指程序中至少引入了一个新的处理语句集合或一个新条件的程序通路。采用流图的术语，即独立路径必须至少包含一条在本次定义路径之前不曾用过的边。

测试可以被设计为基本路径集的执行过程，但基本路径集通常并不唯一。

b. 计算环形复杂度的方法

环形复杂度以图论为基础，为我们提供了非常有用的软件度量。可用如下三种方法之一来计算环形复杂度：

- 控制流图中区域的数量对应于环形复杂度。
- 给定控制流图G的环形复杂度—V(G)，定义为

$$V(G) = E-N+2$$

其中，E是控制流图中边的数量，N是控制流图中的节点数量。

- 给定控制流图G的环形复杂度—V(G)，也可定义为

$$V(G) = P+1$$

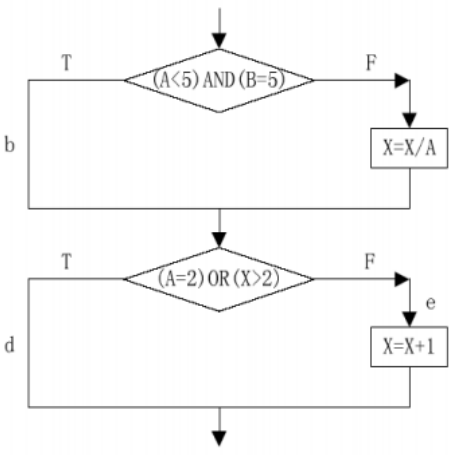
其中，P是控制流图G中判定节点的数量。

(3) 图矩阵

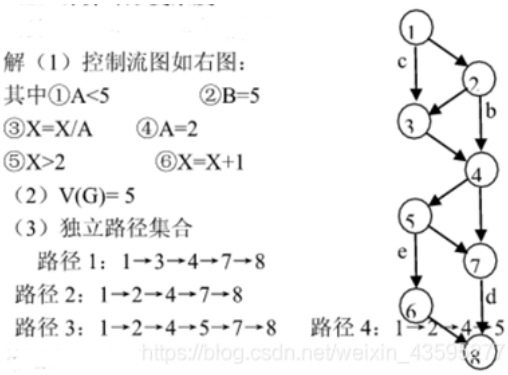
- 图矩阵是控制流图的矩阵表示形式。
- 图矩阵是一个方形矩阵，其维数等于控制流图的节点数。矩阵中的每列和每行都对应于标识的节点，矩阵元素对应于节点间的边。
- 通常，控制流图中的结点用数字标识，边则用字母标识。如果在控制流图中从第 i 个结点到第 j 个结点有一个标识为 x 的边相连接，则在对应图矩阵的第 i 行第 j 列有一个非空的元素 x 。

(4) 习题

> 根据左图给出的程序流程图，
>
> 完成以下要求：
>
> (1) 画出相应的控制流图。
>
> (2) 计算环形复杂度。
>
> (3) 给出相应的图矩阵。
>
> (4) 找出程序的独立路径集合。



解：



对于复合条件，则可将其分解为多个单个条件，并映射成控制流图。

覆盖测试

（1）测试覆盖率

测试覆盖率：用于确定测试所执行到的覆盖项的百分比。
测试覆盖率可以表示出测试的充分性，在测试分析报告中可以作为量化指标的依据，测试覆盖率越高效果越好。但覆盖率不是目标，只是一种手段。

测试覆盖率包括功能点覆盖率和结构覆盖率：

- 功能点覆盖率大致用于表示软件已经实现的功能与软件需要实现的功能之间的比例关系。
- 结构覆盖率包括语句覆盖率、分支覆盖率、循环覆盖率、路径覆盖率等等。

（2）逻辑覆盖法

根据覆盖目标的不同，逻辑覆盖又可分为语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、组合覆盖和路径覆盖。

语句覆盖：选择足够多的测试用例，使得程序中的每个可执行语句至少执行一次。

判定覆盖（分支覆盖）：通过执行足够的测试用例，使程序中的每个取“真”分支和取“假”分支至少均一次。

条件覆盖：使得程序中**每个判定**包含的**每个条件**的可能取值（真/假）都至少满足一次。

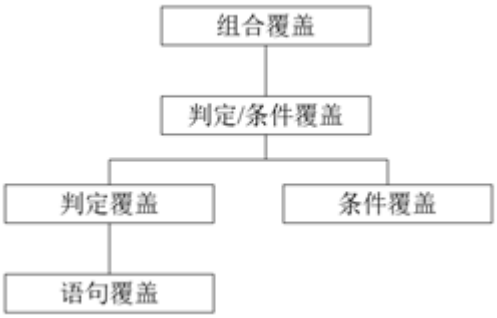
判定/条件覆盖：使得程序中**每个判定**包含的**每个条件**的所有情况（真/假）至少出现一次，并且**每个判定本身**的**判定结果**（真/假）也至少出现一次。

满足判定/条件覆盖的测试用例一定同时满足判定覆盖和条件覆盖。（**判定/条件覆盖也不一定能够完全检查出逻辑表达式中的错误。**）

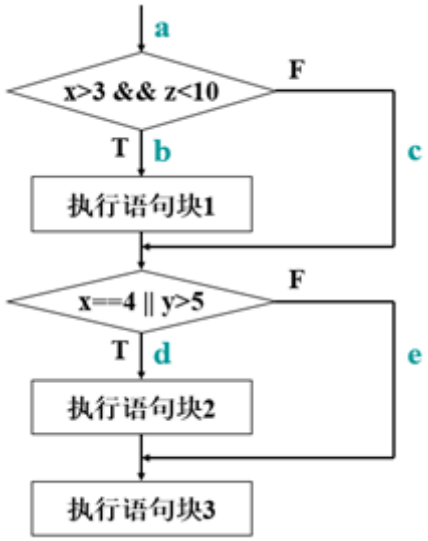
组合覆盖：使得程序中每个判定的所有可能的条件取值组合都至少出现一次。

满足组合覆盖的测试用例一定满足判定覆盖、条件覆盖和判定/条件覆盖。（但也有可能丢失路径）

路径覆盖：设计足够多的测试用例，要求覆盖程序中所有可能的路径。（**满足路径覆盖的测试用例并不一定满足组合覆盖。**）



示例：



a. 语句覆盖：

一个测试用例{ x=4、y=5、z=5 }

b. 判定覆盖：

两个测试用例。路径分别是：abd；ace

c. 条件覆盖：

两个测试用例。

（3）面向对象的覆盖

a. 继承上下文覆盖

由于传统的结构化度量没有考虑面向对象的一些特性（如多态、继承和封装等），所以在面向对象领域，传统的结构化覆盖必须被加强，以满足面向对象特性。

继承上下文覆盖：考虑在每个类的上下文内获得的覆盖率级别。它是扩展到面向对象领域里的一种覆盖率度量方法，用于度量在系统中的多态调用被测试得多好。

继承上下文定义：将基类上下文内例行程序的执行作为独立于继承类上下文内例行程序的执行。同样，它们在考虑继承类上下文内例行程序的执行也独立于基类上下文内例行程序的执行。为了获得100%继承上下文覆盖，代码必须在每个适当的上下文内被完全执行。

b. 基于状态的上下文覆盖

- 在绝大多数面向对象的系统中存在这样的一些类：这些类的对象可以存在于众多不同状态中的任何一种，并且由于类的行为依赖于状态，每个类的行为在每个可能的状态中其性质是不同的。
- 基于状态的上下文覆盖对应于被测类对象的潜在状态。
- 这样基于状态的上下文覆盖把一个状态上下文内的一个例行程序的执行认为是独立于另一个状态内相同例行程序的执行。为了达到100%的基于状态的上下文覆盖，例行程序必须在每个适当的上下文（状态）内被执行。

路径测试

（1）路径表达式

为了满足路径覆盖，必须首先确定具体的路径以及路径的个数。我们通常采用控制流图的边（弧）序列和节点序列表示某一条具体路径，更为概括的表示方法为：

- 弧a和弧b相乘，表示为ab，它表明路径是先经历弧a，接着再经历弧b，弧a和弧b是先后相接的。
- 弧a和弧b相加，表示为a+b，它表明两条弧是“或”的关系，是并行的路段。

路径数的计算：

在路径表达式中，将所有弧均以数值1来代替，再进行表达式的相乘和相加运算，最后得到的数值即为该程序的路径数。

（2）基本路径测试方法

a. 概述

路径测试就是从一个程序的入口开始，执行所经历的各个语句的完整过程。从广义的角度讲，任何有关路径分析的测试都可以被称为路径测试。

完成路径测试的理想情况是做到路径覆盖，但对于复杂性大的程序要做到所有路径覆盖（测试所有可执行路径）是不可能的。

在不能做到所有路径覆盖的前提下，如果某一程序的每一个独立路径都被测试过，那么可以认为程序中的每个语句都已经检验过了，即达到了语句覆盖。这种测试方法就是通常所说的基本路径测试方法。

b. 步骤

基本路径测试方法是在控制流图的基础上，通过分析控制结构的环形复杂度，导出执行路径的基本集，再从该基本集设计测试用例。基本路径测试方法包括以下4个步骤：

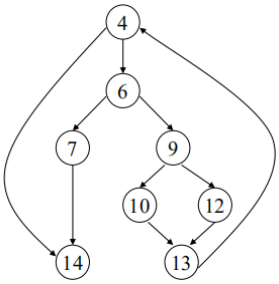
- 画出程序的控制流图。
- 计算程序的环形复杂度，导出程序基本路径集中的独立路径条数，这是确定程序中每个可执行语句至少执行一次所必须的测试用例数目的最小数目。
- 导出基本路径集，确定程序的独立路径。
- 根据（3）中的独立路径，设计测试用例的输入数据和预期输出。

c. 示例

```
void Sort ( int iRecordNum, int iType )
{
    int x=0;
    int y=0;
    while ( iRecordNum-- > 0 )
    {
        If ( iType==0 )
        {x=y+2;break;}
        else
```

```
10          If ( iType==1 )
11          x=y+10;
12          else
13          x=y+20;
14      }
15  }
```

画出控制流图：



计算环形复杂度：

10（条边）- 8（个节点）+ 2 = 4

导出独立路径（用语句编号表示）

路径1：4→14

路径2：4→6→7→14

路径3：4→6→9→10→13→4→14

路径4：4→6→9→12→13→4→14

（3）循环测试方法

从本质上说，循环测试的目的就是检查循环结构的有效性。

通常，循环可以划分为简单循环、嵌套循环、串接循环和非结构循环4类。

a. 测试简单循环：

设其循环的最大次数为n ，可采用以下测试集：

跳过整个循环；只循环一次；只循环两次；

循环 m 次，其中m<n；分别循环 n-1、n 和 n+1 次。

b. 测试嵌套循环：

如果将简单循环的测试方法用于嵌套循环，可能的测试次数会随嵌套层数成几何级数增加。 此时可采用以下办法减少测试次数：

- 测试从最内层循环开始，所有外层循环次数设置为最小值；
- 对最内层循环按照简单循环的测试方法进行；
- 由内向外进行下一个循环的测试，本层循环的所有外层循环仍取最小值，而由本层循环嵌套的循环取某些“典型”值；
- 重复上一步的过程，直到测试完所有循环。

c. 测试串接循环：

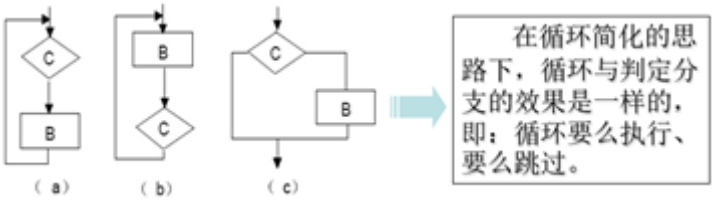
若串接的各个循环相互独立，则可分别采用简单循环的测试方法；否则采用嵌套循环的测试方法。

d. 对于**非结构循环**这种情况，无法进行测试，需要按结构化程序设计的思想将程序结构化后，再进行测试。

e. Z路径覆盖下的循环测试方法

Z路径覆盖是路径覆盖的一种变体，它是将程序中的循环结构简化为选择结构的一种路径覆盖。

循环简化的目的是限制循环的次数，无论循环的形式和循环体实际执行的次数，简化后的循环测试只考虑执行循环体一次和零次（不执行）两种情况，即考虑执行时进入循环体一次和跳过循环体这两种情况。



（4）产生测试用例

在实践中，除了前面给出的各种方法外，通常还可以采用以下三种方法来补充设计测试用例：

- 通过非路径分析得到测试用例：
  这种方法得到的测试用例是在应用系统本身的实践中提供的，基本上是测试人员凭**工作经验**的得到，甚至是**猜测**得到的。
- 寻找尚未测试过的路径并生成相应的测试用例：
  这种方法需要穷举被测程序的所有路径，并与前面已测试路径进行对比。
- 通过指定特定路径并生成相应的测试用例

最少测试用例数计算

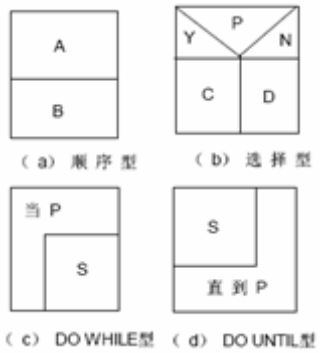
为实现测试的逻辑覆盖，必须设计足够多的测试用例，并使用这些测试用例执行被测程序，实施测试。我们关心的是：对于某个具体的程序来说，至少需要设计多少个测试用例。

结构化程序是由 3种基本控制结构组成：顺序型（构成串行操作）、选择型（构成分支操作）和重复型（构成循环操作）。

避免出现测试用例极多的组合爆炸，把构成循环操作的重复型结构用选择结构代替。这样，任一循环改造成进入循环体或不进入循环体的分支操作了。

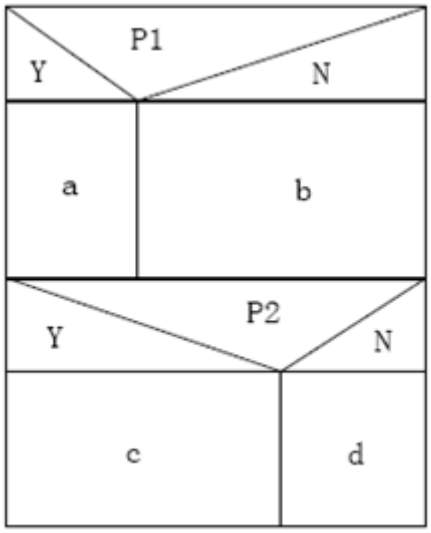
用N-S图表示程序的3种基本控制结构：

- 图中A、B、C、D、S均表示要执行的操作，P是可取真假值的谓词，Y表真值，N表假值。
- 图中的（c）和（d）两种重复型结构代表了两种循环。在做了简化循环的假设以后，对于一般的程序控制流，我们只考虑选择型结构。事实上它已经能体现顺序型和重复型结构了。



a. 例一

下图表达了两个顺序执行的分支结构。当两个分支谓词P1和P2取不同值时，将分别执行a或b及c或d操作。



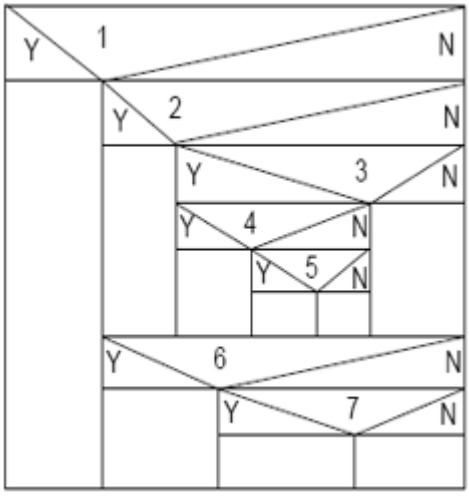
显然，需要至少提供**4个测试用例**才能作到逻辑覆盖，使得ac、ad、bc及bd操作均得到检验。其实，这里的4是图中的第1个分支谓词引出的两个操作，及第2个分支谓词引出的两个操作组合起来而得到的，即2×2=4。并且，这里的2是由于两个并列的操作，即1+1=2 而得到的。

对于一般的、更为复杂的问题，估算最少测试用例个数的原则也是同样的：

- 如果在N-S图中存在有**并列**的层次A1、A2，A1和A2的最少测试用例个数分别为a1、a2，则由 A1、A2 两层所组合的 N-S图对应的最少测试用例数为a1×a2。
- 如果在N-S图中不存在有并列的层次，则对应的最少测试用例数由并列的操作数决定，即N-S图中除谓词之外的操作框的个数。

b. 例二

如下图所示的N-S图，至少需要多少个测试用例完成逻辑覆盖?



> 分析该N-S图：
> 图中的2345和67是并列的两层。
> 其中，2345层对应的最少测试用例数为1+1+1+1+1=5，67层对应的测试用例数为1+1+1=3，2345和67这两层组合后对应的测试用例数为5×3=15。
>
> 最后，由于两层组合后的部分是不满足谓词1时所要做的操作，还要加上满足谓词1要做的操作，因此整个程序所需测试用例数为15+1=16。