1. 软件及其分类

软件是计算机中与硬件相结合的一部分，包括程序和文档。

按照功能划分：系统软件；应用软件

按照技术架构划分：单机软件；C/S 软件；B/S 软件

按照用户划分：产品软件；项目软件

按照开发规模划分：小型；中型；大型

2. 软件测试定义以及过程模型

使用人工和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楚预期结果与实际结果之间的差别。

1. 按测试方法分类（是否查看源代码）

- 白盒测试
- 黑盒测试

2. 按测试方式分类（是否运行程序）

- 静态测试
- 动态测试

3. 按测试过程分类（软件测试阶段）

- 单元测试
- 集成测试
- 系统测试
- 验收测试

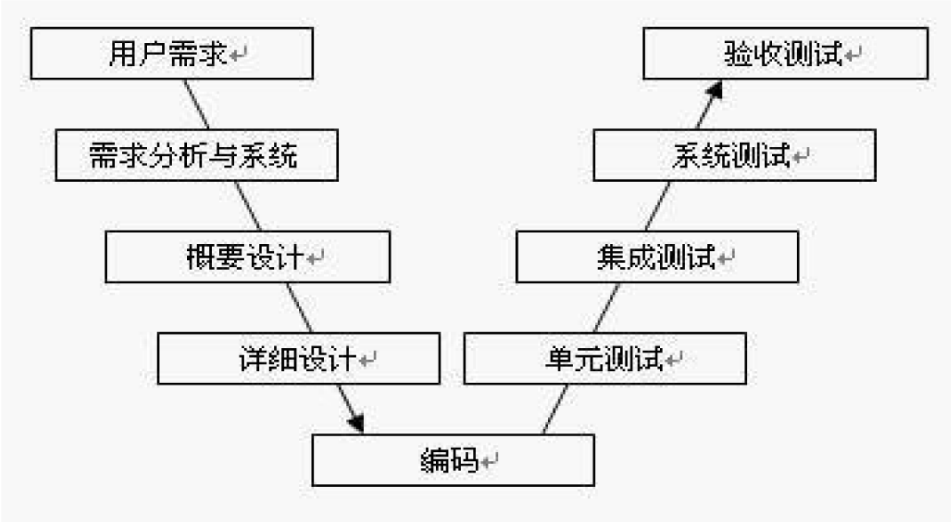
4. 黑盒测试细分

- 功能测试（用户需求）
    - 逻辑功能测试
    - 界面测试
    - 易用性测试
    - 安装测试
    - 兼容性测试
- 性能测试（软件的寿命）
    - 一般性能测试
    - 稳定性测试
    - 负载测试
    - 压力测试

软件测试过程模型是对测试过程一种抽象，用于定义软件测试的流程和方法；

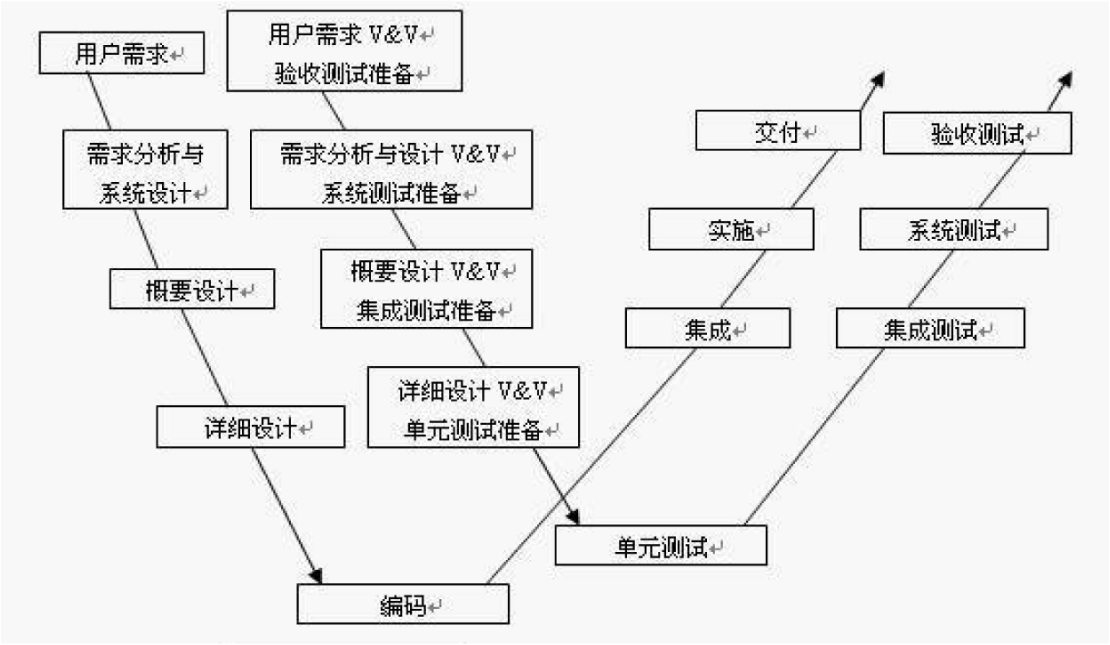
1. V 模型

- **V** 模型是最具有代表意义的测试模型，反映测试活动与分析设计活动的关系。
- **V** 模型指出，单元和集成测试应检测程序的执行是否满足软件设计的要求；系统测试应检测系统功能、性能的质量特性是否达到系统要求的指标；验收测试确定软件的实现是否满足用户需要或合同的要求。



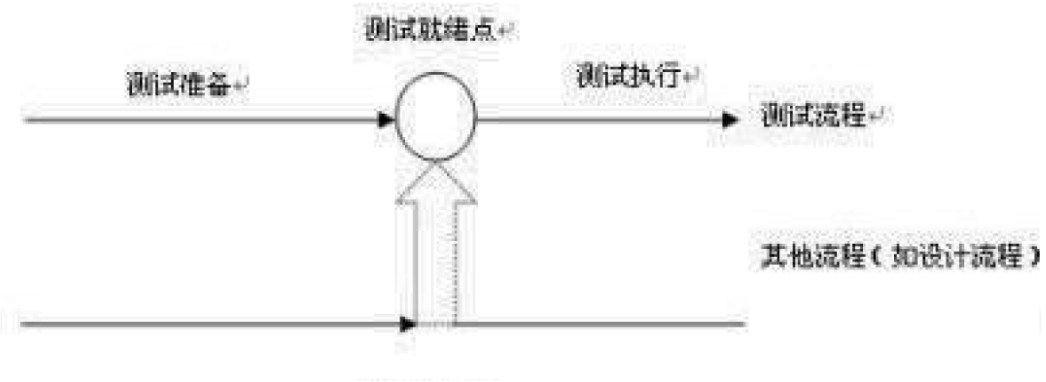
2．W模型

- **W** 模型由两个 **V** 字型模型组成，分别代表测试与开发过程。相比 **V** 模型，增加了同步验证和确认活动。
- **W** 模型强调：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、设计等同样要测试，也就是说，测试与开发是同步进行的。



3．H模型

- **H** 模型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。
- **H** 模型揭示了软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。



4．其他模型

除上述几种常见模型外，还流传着其他几种模型，例如 **X** 模型、前置测试模型等。

- **X** 模型提出针对单独的程序片段进行相互分离的编码和测试，此后通过频繁的交接，通过集成最综合成为可执行的程序。
- 前置测试模型体现了开发与测试的结合，要求对每一个交付内容进行测试。

3. 软件缺陷的定义，种类，分类方法，生命周期

软件缺陷是存在于软件**(**文档、数据、程序**)**之中不希望或不可接受的偏差。其结果是软件运行于某一特定条件时出现软件故障，这时称软件缺陷被激活。

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

按照严重程度划分：系统崩溃；严重；一般；次要；建议（严重；一般；次要）

按照优先级划分：高；中；低

按照测试种类划分：逻辑功能类；性能类；界面类；易用性类；兼容性类

按照功能模块划分：一般软件产品都是分为若干个功能模块

按照 Bug 生命周期划分：新建；确认；解决；关闭；重新打开

4. 静态测试包括代码审查与静态结构分析

白盒测试可分为静态测试和动态测试。

静态测试是一种不通过执行程序而进行测试的技术，其关键功能是检查软件的表示和描述是否一致，没有冲突或者没有歧义。它瞄准的是纠正软件系统在描述、表示和规格上的错误，是任何进一步测试的前提。

动态测试需要软件的执行，当软件系统在模拟或真实环境中执行之前、之中和之后，对软件系统行为的分析是动态测试的主要特点。它显示一个系统在检查状态下是正确还是不正确。

最常见的静态测试是找出源代码的语法错误，这类测试可由编译器来完成，因为编译器可以逐行分析检验程序的语法，找出错误并报告。除此之外，测试人员须采用人工的方法来检验程序，有些地方存在非语法方面的错误，只能通过人工检测的方法来判断。

人工检测的方法主要有代码检查法、静态结构分析法等。

代码检查法主要是通过桌面检查，代码审查和走查方式，对以下内容进行检查：

(1) 检查代码和设计的一致性；

(2) 代码的可读性以及对软件设计标准的遵循情况；

(3) 代码逻辑表达的正确性；

(4) 代码结构的合理性；

(5) 程序中不安全、不明确和模糊的部分；

(6) 编程风格方面的问题等。

代码检查方式：**(1)**桌面检查**(2)**代码审查**(3)**走查

- 桌面检查是指程序设计人员对源程序代码进行分析，检查，并补充相关的文档，发现程序中的错误。主要包括检查变量，标号，子程序，宏，函数，常量，设计标准，风格等。
- 代码审查是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。
  采用讲解，提问并使用编码模板进行的查找错误的活动。一般有正式的计划，流程和结果报告。
  代码审查小组成员：
  组长—能力较强的程序员
  待审程序的设计者或程序员
  测试专家
  代码审查时应注意问题：
  - 提出的建议应针对程序本身，而不应针对程序员。
  - 程序员对整个审查过程采取积极和建设性的态度。
- 走查
  走查是以小组为单元进行代码阅读的，同样也是一系列规程和错误检查技术的集合。采用讲解，讨论，模拟运行的方式进行的查找错误的活动。
  人员组成
  - 一位经验丰富的程序员
  - 一位程序设计语言专家
  - 一位程序员新手
  - 一位其他不同项目的人员
  - 一位该软件编程小组的成员

在静态结构分析中，测试人员通过使用测试工具分析程序源代码的系统结构、数据结构、数据接口、内部控制逻辑等内部结构，生成函数调用关系图、模块控制流图、内部文件调用关系图等各种图形、图表，清晰地标识整个软件的组成结构。

通过分析这些图表，包括控制流分析、数据流分析、接口分析、表达式分析等，使其便于阅读与理解，然后可以通过分析这些图表，检查软件有没有存在缺陷或错误。

静态结构分析法通常采用以下方法进行源程序的静态分析：

- **(1)** 通过各种图表对源程序进行静态分析

常用的各种引用表主要有：

①标号交叉引用表

②变量交叉引用表

③子程序**(**宏、函数**)**引用表

④等价表

⑤常数表

常用的关系图、控制流图：

①函数调用关系图：列出所有函数，用连线表示调用关系，通过应用程序各函数之间的调用关系展示了系统的结构。

②模块控制流图：由许多结点和连接结点的边组成的图形，其中每个结点代表一条或多条

语句，边表示控制流向，可以直观地反映出一个函数的内部结构。

(2)静态错误分析

- 静态错误分析主要用于确定在源程序中是否有某类错误或“危险”结构。

①类型和单位分析

②引用分析

③表达式分析

④接口分析

5. 动态测试包括程序插桩和逻辑覆盖

程序插桩方法是借助往被测程序中插入操作，来实现测试目的的方法，即向源程序中添加一些语句，实现对程序语句的执行、变量的变化等情况进行检查。

最简单的插桩：在程序中插入打印语句 **printf(**“……”**)**语句。

如果我们想要了解一个程序在某次运行中所有可执行语句被覆盖的情况，或是每个语句的实际执行次数，最好的办法是利用插桩技术。

6. 逻辑覆盖包括哪些覆盖以及其强度

逻辑覆盖也是白盒测试主要的动态测试方法之一，是以程序内部的逻辑结构为基础的测试技术，是通过对程序逻辑结构的遍历实现程序的覆盖，这一方法要求测试人员对程序的逻辑结构有清楚的了解。

从覆盖源程序语句的详细程度分析，逻辑覆盖标准有语句覆盖、判定覆盖、条件覆盖、判定**/**条件覆盖、条件组合覆盖。

1 语句覆盖

语句覆盖使程序中每个语句至少都能被执行一次。

2 判定覆盖

比语句覆盖稍强的覆盖标准是判定覆盖。按判定覆盖准则进行测试是指，设计若干测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断的真假值均曾被满足。判定覆盖又称为分支覆盖。

3 条件覆盖

在设计程序中，一个判定语句是由多个条件组合而成的复合判定。

条件覆盖的含义是：构造一组测试用例，使得每一判定语句中每个逻辑条件的可能值至少满足一次。

4 条件判定组合覆盖

条件判定组合覆盖的含义是：设计足够的测试用例，使得判定中每个条件的所有可能**(**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。

5 多条件覆盖

多条件覆盖也称为条件组合覆盖，它的含义是：设计足够的测试用例，使得每个判定中条件的各种可能组合都至少出现一次。显然满足多条件覆盖的测试用例是一定满足判定覆盖、条件覆盖和条件判定组合覆盖的。

7. 黑盒测试概念以及主要方法

黑盒测试也称数据驱动测试，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构盒内部特性的情况下，测试者在程序接口进行测试。只依据程序的需求规格说明书和用户手册，检查程序的功能是否符合其功能说明及性能是否满足用户的要求。

在黑盒测试过程中，只是通过输入数据、进行操作、观察输出结果，来检查软件系统是否按照需求规格说明书的规定正常使用，软件是否能适当地接收输入数据而产生正确的输出信息，并保持外部信息的完整性。

功能测试：等价类划分，边界值分析，错误推测法，因果图法；

场景法，判定表驱动，正交测试法，功能图法等。

非功能测试：强度测试，性能测试，安全测试，等

1. 等价类划分法

- 等价类划分是一种典型的、常用的黑盒测试方法。
- 所谓等价类是指某个输入域的子集，使用这一方法时，是把所有可能的输入数据，即程序的输入域划分成若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。
- 等价类的划分有以下两种不同的情况：
- ①有效等价类：是指对于程序规格说明来说，是合理的、有意义的输入数据构成的集合。利用它，可以检验程序是否实现了规格说明预先规定的功能和性能。
- ②无效等价类：是指对于程序规格说明来说，是不合理的、无意义的输入数据构成的集合。利用它，可以检查程序中功能和性能的实现是否有不符合规格说明要求的地方。
- 划分等价类的方法如下：

①按区间划分（学生成绩）

②按数值划分（学生个数小于 10）

③按数值集合划分

④按限制条件划分

⑤按限制规则划分

⑥按处理方式划分

③按数值集合划分

如果输入条件规定了输入值的集合，或者是规定了“必须如何”的条件，这时可确立一个有效等价类和一个无效等价类。

例如，在 C 语言中对变量标识符规定为“以字母或下划线打头的……串”。那么所有以字母或下划线打头的构成有效等价类，而不在此集合内（不以字母打头）的归于无效等价类。

④按限制条件划分

如果输入条件是一个布尔量（0 或 1），则可以确定一个有效等价类和一个无效等价类。

⑤按限制规则划分

例如，C 语言规定“一个语句必须以分号‘;’结束”。这时，可以确定一个有效等价类“以‘;’结束”，若干个无效等价类“以‘:’结束”、“以‘,’结束”、“以‘’结束”等。

⑥按处理方式划分

其他划分方式：

- 如果规定了输入数据为整型，则可以划分出正整数、零和负整数等三个有效类。
- 如果程序的处理对象是表格，则应该使用空表，以及含一项或多项的表。

在确立了等价类之后，建立等价类表，列出所有划分出的等价类

表 3-1　等价类表示例

输入条件	有效等价类	无效等价类
…	…	…
…	…	…

2.边界值分析法

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。

在测试过程中，边界值分析法是通过选择等价类边界的测试用例进行测试，边界值分析法与等价类划分法的区别是边界值分析不是从某等价类中随便挑一个作为代表，而是使这个等价类的每个边界都要作为测试条件。另外，边界值分析不仅考虑输入条件边界，还要考虑输出域边界产生的测试情况。

使用边界值分析方法设计测试用例，首先应确定边界情况。

通常输入等价类与输出等价类的边界，就是应着重测试的边界情况。应当选取正好等于，刚刚大于，或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

选择测试用例的原则如下。

①如果输入条件规定了值的范围，则应该取刚达到这个范围的边界值，以及刚刚超过这个范围边界的值作为测试输入数据。

②如果输入条件规定了值的个数，则用最大个数、最小个数、比最大个数多 1 个、比最小个数少 1 个的数作为测试数据。

③根据规格说明的每一个输入条件，使用前面两条规则。

④根据规格说明的每一个输出条件，使用前面两条规则。

⑤如果程序的规格说明给出的输入域或输出域是有序集合（如有序表、顺序文件等），则应选取集合的第一个和最后一个元素作为测试用例。

⑥如果程序用了一个内部结构，应该选取这个内部数据结构的边界值作为测试用例。

⑦分析规格说明，找出其他可能的边界条件。

边界值分析测试

采用边界值分析测试的基本思想是：故障往往出现在输入变量的边界值附近。

因此，边界值分析法利用输入变量的最小值(min)、略大于最小值(min+)、输入值域内的任意值(nom)、略小于最大值(max-)和最大值(max)来设计测试用例。

在边界值分析法中获取测试用例的方法是：

(1) 每次保留程序中一个变量，让其余的变量取正常值，被保留的变量依次取 min、min+、nom、max-和 max。

(2) 对程序中的每个变量重复 (1) 。

例 2：有二元函数 f(x,y)，其中 x∈[1,12]，y∈[1,31]。

则采用边界值分析法设计的测试用例是：

{ <1,15>, <2,15>, <11,15>, <12,15>, <6,15>, <6,1>, <6,2>, <6,30>, <6,31> }

3. 错误推测法

基于经验和直觉推测程序中所有可能存在的各种错误，从而有针对性的设计测试用例的方法，这就是错误推测法。

错误推测法的基本想法是：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。

4. 因果图法

因果图法是一种利用图解法分析输入的各种组合情况，从而设计测试用例的方法，它适合于检查程序输入条件的各种组合情况。

利用因果图生成测试用例的基本步骤如下。

①分析软件规格说明的描述中哪些是原因，哪些是结果。原因是输入条件或输入条件的等价类，结果是输出条件。

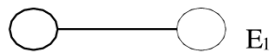
②分析软件规格说明描述中的语义，找出原因与结果之间、原因与原因之间对应的关系，根据这些关系，画出因果图。

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。为表明这些特定的情况，在因果图上使用若干标准的符号标明约束条件。

④把因果图转换成判定表。

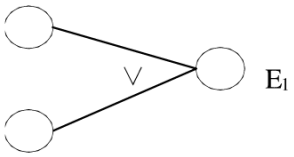
⑤为判定表中的每一列设计测试用例。

通常在因果图中，用 C_i 表示原因，E_i 表示结果，其基本符号如图 3-15 所示。

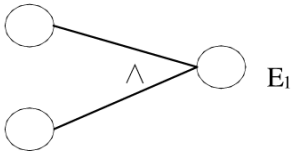


（a）恒等

（b）非



（c）或



（d）与

5. 场景法

现在的软件几乎都是用事件触发来控制流程的，事件触发时的情景便形成了场景，而同一事件不同的触发顺序和处理结果就形成事件流。这种在软件设计方面的思想也可以引入到软件测试中，可以比较生动地描绘出事件触发时的情景，有利于测试设计者设计测试用例，同时使测试用例更容易理解和执行。

用例场景用来描述流经用例的路径，从用例开始到结束遍历这条路径上所有基本流和备选流。

8. 测试用例的定义以及分类

测试用例(test case)是可被独立执行的一个过程，是最小的测试实体，不能再被分解。

测试用例：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，测试次数和预期结果（测试用例=输入+输出+测试环境）

测试用例归为 5 大类：

- 白盒测试用例
- 软件各项功能的测试用例
- 用户界面测试用例
- 软件的各项非功能测试用例
- 对软件缺陷修正所确认的测试用例

9. 单元测试任务，单元测试工具，单元测试环境

单元测试是对软件设计的最小单元——模块进行正确性检验的测试工作，主要测试模块在语法、格式和逻辑上的错误。

单元测试是针对每个程序模块进行测试，单元测试的主要任务是解决以下 5 个方面的测试问题。

1．模块接口测试

针对模块接口测试应进行的检查，主要涉及以下几方面的内容：

(1)模块接受输入的实际参数个数与模块的形式参数个数是否一致。

(2)输入的实际参数与模块的形式参数的类型是否匹配。

(3)输入的实际参数与模块的形式参数所使用单位是否一致。

(4)调用其他模块时，所传送的实际参数个数与被调用模块的形式参数的个数是否相同。

(5)调用其他模块时，所传送的实际参数与被调用模块的形式参数的类型是否匹配。

(6)调用其他模块时，所传送的实际参数与被调用模块的形式参数的单位一致。

(7)调用内部函数时，参数的个数、属性和次序是否正确。

(8)在模块有多个入口的情况下，是否有引用与当前入口无关的参数。

(9)是否会修改了只读型参数。

(10)出现全局变量时，这些变量是否在所有引用它们的模块中都有相同的定义。

(11)有没有把某些约束当做参数来传送。

2．模块局部数据结构测试

3．模块中所有独立执行路径测试

4．各种错误处理测试

5．模块边界条件测试

单元测试环境的建立

一般情况下，在完成了程序编写、复查和语法正确性验证后，就应进行单元测试。测试用例设计应与复审工作相结合，根据设计信息选取数据，将增大发现上述各类错误的可能性。

单元测试时，需设置若干辅助测试模块。辅助模块有两种，一种是驱动模块(Driver)，用以模拟被测试模块的上级模块。另一种是被调用模拟子模块(Sub)，用以模拟被测模块工作过程中所调用的模块。

单元测试工具种类

- 代码规则/风格检查工具
- 内存资源泄漏检查工具
- 代码覆盖率检查工具
- 代码性能检查工具

类别	工具
C 语言	C++ Test、CppUnit、QA C/C++、CodeWzard、Insure++6.0
Java 语言	Jtest、Junit、Jmock、EasyMock、MockRunner
JUnit 扩展框架	TestNG 、JWebUnit 和 HttpUnit
GUI （功能）	JFCUnit、Marathor
通用的	Rexelint、Splint、McCabe QA、CodeCheck、GateKeeper
.NET	.TEST、Nunit
Data Object，DAO	DDTUnit，DBUnit
EJB	MockEJB 或者 MockRunner
Servlet，Struts	Cactu，StrutsUnitTest
XML	XMLUnit
内嵌式系统等	Logiscope、JTestCase

10. 系统测试，功能测试，性能测试，负载测试，压力测试分别等

什么是系统测试？

用户的需求可以分为功能性需求和非功能性需求，而非功能性的需求被归纳为软件产品的各种质量特性，如安全性、兼容性和可靠性等

系统测试就是针对这些非功能特性展开的，就是验证软件产品符合这些质量特性的要求，从而满足用户和软件企业自身的非功能性需求。所以，系统测试分为负载测试、性能系统、容量测试、安全性测试、兼容性测试和可靠性测试等

系统性能的改善是测试、调整、再测试、再调整……一个持续改进的过程——性能调优
性能调优需要借助负载测试方法的帮助
负载测试和性能测试有较多相似之处，例如，测试方法比较接近、都关注系统的性能，而且多数情况下使用相同的测试工具
负载测试可以看作是性能测试所采用的一种技术
压力测试可以被看作是负载测试的一种，即高负载下的负载测试
容量测试也采用负载测试技术来实现

负载测试是通过模拟实际软件系统所承受的负载条件、改变系统负载大小和负载方式来发现系统中所存在的问题

压力测试是在强负载情况下(如大数据量、大量并发用户连接等)稳定性进行测试，查看应用系统在峰值(瞬间使用高峰)使用情况下的行为表现，更有效地发现系统稳定性的隐患和系统在负载峰值的条件下功能隐患等，确认系统是否具有良好的容错能力和可恢复能力。

性能测试是为获取或验证系统性能指标而进行的测试。

11. 系统测试工具

12. 验收测试种类以及通过标准

多采用α 测试和β 测试。

《需求规格说明》验收标准

13. 测试主动化的概念以及特点优势

软件自动化测试的概念

软件测试自动化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的测试，它是软件测试的一个重要组成部分，能够完成许多手工无法完成或者难以实现的一些测试工作。正确、合理地实施自动化测试，能够快速、全面地对软件进行测试，从而提高软件质量、节省经费、缩短产品发布周期。

使用测试工具的目的就是要提高软件测试的效率和软件测试的质量。

通常，自动化测试的好处有：

- 产生可靠的系统；
- 改进测试工作质量；
- 减少测试工作量并加快测试进度。

自动化测试带来的好处：

- 缩短软件开发测试周期，可以让产品更快投放市场。
- 测试效率高，充分利用硬件资源。
- 节省人力资源，降低测试成本。
- 增强测试的稳定性和可靠性。
- 提高软件测试的准确度和精确度，增加软件信任度。
- 软件测试工具使测试工作相对比较容易，但能产生更高质量的测试结果。
- 手工不能做的事情，自动化测试能做，如负载、性能测试。

软件测试实行自动化进程，绝不是因为厌烦了重复的测试工作，而是因为测试工作的需要，更准确地说是回归测试和系统测试的需要。

14. 基本路径测试法包括四步：绘制控制流图，计算环路复杂度，列举基本路径集，设计测试用例

15. 等价类划分与边界值分析综合运用：绘制等价类表和测试用例表