**一、判断题**

1．软件测试的目的是尽可能多的找出软件的缺陷。（Y）

2．Beta测试是验收测试的一种。（Y）

3．验收测试是由最终用户来实施的。（N）

4．项目立项前测试人员不需要提交任何工件。（Y）

5．单元测试能发现约80%的软件缺陷。（Y）

6．代码评审是检查源代码是否达到模块设计的要求。（N）

7．自底向上集成需要测试员编写驱动程序。（Y）

8．负载测试是验证要检验的系统的能力最高能达到什么程度。（N）

9．测试人员要坚持原则，缺陷未修复完坚决不予通过。（N）

10．代码评审员一般由测试员担任。（N）

11．我们可以人为的使得软件不存在配置问题。（N）

12．集成测试计划在需求分析阶段末提交。（N）

**二、选择题**

1．软件验收测试的合格通过准则是：（ABCD）

A．软件需求分析说明书中定义的所有功能已全部实现，性能指标全部达到要求。

B．所有测试项没有残余一级、二级和三级错误。

C．立项审批表、需求分析文档、设计文档和编码实现一致。

D．验收测试工件齐全。

2．软件测试计划评审会需要哪些人员参加？（ABCD）

A．项目经理

B．SQA负责人

C．配置负责人

D．测试组

3．下列关于alpha测试的描述中正确的是：（AD）

A．alpha测试需要用户代表参加

B．alpha测试不需要用户代表参加

C．alpha测试是系统测试的一种

D．alpha测试是验收测试的一种

4．测试设计员的职责有：（BC）

A．制定测试计划

B．设计测试用例

C．设计测试过程、脚本

D．评估测试活动

5．软件实施活动的进入准则是：（ABC）

A．需求工件已经被基线化

B．详细设计工件已经被基线化

C．构架工件已经被基线化

D．项目阶段成果已经被基线化

6、为保证测试活动的可控性，必须在软件测试过程中进行软件测试配置管理，一般来说，软件测试配置管理中最基本的活动包括\_A\_\_\_\_\_

A．配置项标识、配置项控制、配置状态报告、配置审计

B．配置基线确立、配置项控制、配置报告、配置审计

C．配置项标识、配置项变更、配置审计、配置跟踪

D．配置项标识、配置项控制、配置状态报告、配置跟踪

7、\_\_B\_\_\_\_方法根据输出对输入的依赖关系设计测试用例。

 A．路径测试　　　　B．等价类　　　　 C．因果图　　　　D．边界值

8、在C++语言中，若类C中定义了一个方法int f(int a，int b)，那么方法\_\_\_A\_\_\_不能与该方法同时存在于类C中

A．int f(int x，int y)　　　　　　　 　B．int f(float a，int b)

C．float f(int x，float y)　　　　　　 D．int f(int x，float y)

9、下列关于软件验收测试的合格通过准则错误的是：\_\_C\_\_\_\_

A． 软件需求分析说明书中定义的所有功能已全部实现，性能指标全部达到要求；

B． 所有测试项没有残余一级、二级和三级错误；

**C**． 立项审批表、需求分析文档、设计文档和编码实现不一致；

D． 验收测试工件齐全

**三、填空题**

1.软件验收测试包括：正式验收测试，alpha测试，beta测试。

2.系统测试的策略有：功能测试，性能测试，可靠性测试，负载测试，易用性测试，强度测试，安全测试，配置测试，安装测试，卸载测试，文挡测试，故障恢复测试，界面测试，容量测试，兼容性测试，分布测试，可用性测试，（有的可以合在一起，分开写只要写出15就满分哦）

3.设计系统测试计划需要参考的项目文挡有：软件测试计划，软件需求工件和迭代计划。

4.对面向过程的系统采用的集成策略有：自顶向下，自底向上两种。

5.通过画因果图来写测试用例的步骤为：

（1）分析软件规格说明描述中，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）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是什么关系?根据这些关系，画出因果图。

（3）由于语法或环境限制，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标明约束或限制条件。

（4）把因果图转换成判定表。

（5）把判定表的每一列拿出来作为依据，设计测试用例。

**四、简答题**

**1.区别阶段评审的与同行评审**

同行评审目的:发现小规模工作产品的错误,只要是找错误;

阶段评审目的:评审模块阶段作品的正确性可行性及完整性

同行评审人数:3-7人人员必须经过同行评审会议的培训,由SQA指导

阶段评审人数:5人左右评审人必须是专家具有系统评审资格

同行评审内容:内容小一般文档< 40页,代码< 500行

阶段评审内容:内容多,主要看重点

同行评审时间:一小部分工作产品完成

阶段评审时间:通常是设置在关键路径的时间点上!

2.什么是软件测试

为了发现程序中的错误而执行程序的过程

**3简述集成测试的过程**

系统集成测试主要包括以下过程：

1.构建的确认过程。

2.补丁的确认过程。

3.系统集成测试测试组提交过程。

4.测试用例设计过程。

5.测试代码编写过程。

6. Bug的报告过程。

7.每周/每两周的构建过程。

8.点对点的测试过程。

9.组内培训过程。

**4怎么做好文档测试**

仔细阅读，跟随每个步骤，检查每个图形，尝试每个示例。P142

检查文档的编写是否满足文档编写的目的

内容是否齐全，正确

内容是否完善

标记是否正确

**5白盒测试有几种方法**

总体上分为静态方法和动态方法两大类。

静态：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

动态：语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖。

**6系统测试计划是否需要同行审批，为什么?**

需要，系统测试计划属于项目阶段性关键文档，因此需要评审。

**7Alpha测试与beta的区别？**

Alpha测试在系统开发接近完成时对应用系统的测试；测试后仍然会有少量的设计变更。这种测试一般由最终用户或其它人员完成，不能由程序或测试员完成。

Beta测试当开发和测试根本完成时所做的测试，最终的错误和问题需要在最终发行前找到。这种测试一般由最终用户或其它人员完成，不能由程序员或测试员完成。

**8比较负载测试，容量测试和强度测试的区别？**

负载测试：在一定的工作负荷下，系统的负荷及响应时间。

强度测试：在一定的负荷条件下，在较长时间跨度内的系统连续运行给系统性能所造成的影响。

容量测试：容量测试目的是通过测试预先分析出反映软件 系统应用特征的某项指标的极限值（如最大并发用户数、数据库记录数等），系统在其极限值状态下没有出现任何软件故障或还能保持主要功能正常运行。容量测试 还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。容量测试的目的是使系统承受超额的数据容量来发现它是否能够正确处理。容量测试是面向数据 的，并且它的目的是显示系统可以处理目标内确定的数据容量。

**9测试结束的标准是什么？**

用例全部测试。

覆盖率达到标准。

缺陷率达到标准。

其他指标达到质量标准

**10描述软件测试活动的生命周期？**

测试周期分为计划、设计、实现、执行、总结。其中：

计划：对整个测试周期中所有活动进行规划，估计工作量、风险，安排人力物力资源，安排进度等；

设计：完成测试方案，从技术层面上对测试进行规划；

实现：进行测试用例和测试规程设计；

执行：根据前期完成的计划、方案、用例、规程等文档，执行测试用例。

总结：记录测试结果，进行测试分析，完成测试报告。

**11软件的缺陷等级应如何划分？**

A类—严重错误，包括以下各种错误： 1． 由于程序所引起的死机,非法退出 2． 死循环 3． 数据库发生死锁 4． 因错误操作导致的程序中断 5． 功能错误 6． 与数据库连接错误 7． 数据通讯错误

B类—较严重错误，包括以下各种错误： 1． 程序错误 2． 程序接口错误 3． 数据库的表、业务规则、缺省值未加完整性等约束条件

C类—一般性错误，包括以下各种错误： 1． 操作界面错误（包括数据窗口内列名定义、含义是否一致） 2． 打印内容、格式错误 3． 简单的输入限制未放在前台进行控制 4． 删除操作未给出提示 5． 数据库表中有过多的空字段

D类—较小错误，包括以下各种错误： 1． 界面不规范 2． 辅助说明描述不清楚 3． 输入输出不规范 4． 长操作未给用户提示 5． 提示窗口文字未采用行业术语 6． 可输入区域和只读区域没有明显的区分标志

E类—测试建议

**五、用例设计**

**随意选取一个简单物品，假定是一个喝水的带广告图案的花纸杯，设计出尽可能多的测试用例。**测试项目：杯子需求测试:查看杯子使用说明书界面测试:查看杯子外观功能度：用水杯装水看漏不漏；水能不能被喝到安全性：杯子有没有毒或细菌可\*性：杯子从不同高度落下的损坏程度可移植性：杯子再不同的地方、温度等环境下是否都可以正常使用兼容性：杯子是否能够容纳果汁、白水、酒精、汽油等易用性：杯子是否烫手、是否有防滑措施、是否方便饮用用户文档：使用手册是否对杯子的用法、限制、使用条件等有详细描述疲劳测试：将杯子盛上水（案例一）放24小时检查泄漏时间和情况；盛上汽油（案例二）放24小时检查泄漏时间和情况等压力测试：用根针并在针上面不断加重量，看压强多大时会穿透跌落测试: ? 杯子加包装(有填充物),在多高的情况摔下不破损震动测试: 杯子加包装(有填充物),六面震动,检查产品是否能应对恶劣的铁路\公路\航空运输测试数据：测试数据具体编写此处略。其中应用到：场景法、等价类划分法、因果图法、错误推测法、边界值法等方法期望输出：该期望输出需查阅国标、行标以及使用用户的需求说明书测试: 检查说明书书写准确性

**六、网络、操作系统、语言知识**

**1请你分别划划OSI的七层网络结构图，和TCP/IP的五层结构图？**

答:七层结构从上到下依次是：7 应用层 ;6 表示层 ;5 会话层 ;4 传输层 ;3 网络层 ;2 数据链路层 ;1 物理层 五层结构是

5 应用层;4 运输层;3 网络层; 2 链路层;1 物理层。

**2请你详细的解释一下IP协议的定义，在哪个层上面，主要有什么作用？ TCP与UDP呢？**

答:UDP，TCP在传输层，IP在网络层， TCP/IP是英文Transmission Control Protocol/Internet Protocol的缩写，意思是"传输控制协议/网际协议"。TCP/IP协议组之所以流行，部分原因是因为它可以用在各种各样的信道和底层协议（例如T1和、以太网以及RS-232串行接口）之上。确切地说，TCP/IP协议是一组包括TCP协议和IP协议，UDP（User Datagram Protocol）协议、ICMP（Internet Control Message Protocol）协议和其他一些协议的协议组。TCP/IP协议并不完全符合OSI的七层参考模型。传统的开放式系统互连参考模型，是一种通信协议的7层抽象的参考模型,其中每一层执行某一特定任务。该模型的目的是使各种硬件在相同的层次上相互通信。这7层是:物理层、数据链路层、网路层、传输层、话路层、表示层和应用层。而TCP/IP通讯协议采用了4层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。这4层分别为：应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等。 传输层：在此层中，它提供了节点间的数据传送服务，如传输控制协议（TCP）、用户数据报协议（UDP）等，TCP和UDP给数据包加入传输数据并把它传输到

**3请问交换机和路由器分别的实现原理是什么？分别在哪个层次上面实现的？**

一般意义上说交换机是工作在数据链路层。但随着科技的发展，现在有了三层交换机，三层交换机已经扩展到了网络层。也就是说：它等于“数据链路层 + 部分网络层”。交换机中传的是帧。通过存储转发来实现的。 路由器是工作在网络层。路由器中传的是IP数据报。主要是选址和路由。

**4请问C++的类和C里面的STRUCT有什么区别？**

答:除关键字不同外(class,struct)的唯一区别是,结构在默认情况下的成员是公共(public)的,而类在默认情况下的成员是私有(private)的。 在C++中,结构是特殊的类。　class是从struct发展而来的。之所以将struct和class都保留，是因为： 1、提出class是为了强调一种概念。 2、保留struct是为了照顾到大多数人的习惯。 struct和class是有区别的。 　struct保证成员按照声明顺序在内存中存储。class不保证等等 而它们都可以继承，实现多态等。但也有少许区别。比如： struct A { }； class B : A{ }; 8086是Inter的16位微处理器 有16根数据线和20根地址线，它既能处理16位数据，也能处理8位数据 内部数据总线都是按16位设计的，单外部数据总线只有8条

**七、其他**

**一、谈谈你了解的软件测试流程及工具**

一般测试流程：

1.需求分析阶段：对业务的学习，分析需求点。

2.测试计划阶段：测试组长根据SOW开始编写《测试计划》，其中包括人员，软件硬件资源，测试点，集成顺序，进度安排和风险识别等内容。

3.测试设计阶段：测试方案一般由对需求很熟的高资深的测试工程师设计，测试方案要求根据《SRS》上的每个需求点设计出包括需求点简介，测试思路和详细测试方法三部分的方案。《测试方案》编写完成后也需要进行评审。

4.测试方案阶段：主要是对测试用例和规程的设计。测试用例是根据《测试方案》来编写的，通过《测试方案》阶段，测试人员对整个系统需求有了详细的理解。这时开始编写用例才能保证用例的可执行和对需求的覆盖。测试用例需要包括测试项，用例级别，预置条件，操作步骤和预期结果。其中操作步骤和预期结果需要编写详细和明确。测试用例应该覆盖测试方案，而测试方案又覆盖了测试需求点，这样才能保证客户需求不遗漏。同样，测试用例也需要评审。

5.测试执行阶段：执行测试用例，及时提交有质量的Bug和测试日报，测试报告等相关文档。

流程：

需求分析→测试计划→测试设计→测试环境搭建→测试执行→测试记录→缺陷管理→软件评估→RTM.

测试工具：

C/S及B/S架构相关的软件产品，那么对不同操作系统，如Windows系列、unix、linux甚至苹果OS等

测试环境都是必须的

常用的软件测试工具分为：

开源测试管理工具：Bugfree、Bugzilla、TestLink、mantis

开源功能自动化测试工具：Watir、Selenium、MaxQ、WebInject

开源性能自动化测试工具：Jmeter、OpenSTA、DBMonster、TPTEST、Web ApplicationLoadSimulator

[TestDirector]：企业级测试管理工具，也是业界第一个基于Web的测试管理系统。

[Quality Center]：基于Web的测试管理工具，可以组织和管理应用程序测试流程的所有阶段，包括指定测试需求、计划测试、执行测试和跟踪缺陷。

[QuickTest Professional]：用于创建功能和回归测试。

[LoadRunner]：预测系统行为和性能的负载测试工具。

**二、如何发现客户端软件中的内存泄露？**

检测内存泄漏的问题应该尽早进行，它绝不应该是系统测试时的主要目标。也就是说，检查是否存在内存泄漏，应该从编码时就要考虑，单元测试和集成测试时要重点检查。如果前期没有考虑，等到了系统测试才想起检查或者才发现泄漏，为时已晚，此时再去定位泄漏的位置，太难太难了，它可能会让你的交付日期delay不确定的时间。 最近看了一些自动错误预防（AEP）的理论，我深受启发。作为测试人员的我们，从“发现错误”转变到“帮助开发人员预防错误”，这将是一个巨大的转变。所以说，下面我的答案中的第一点，我先说如何预防内存泄漏的问题，然后再讲如何发现。

1 如何在开发过程中有效预防内存泄漏？

第一步：遵循“好”的编程规则

“好”的编程规则是各位前辈经验和教训的集合，好的编程规则堪称开发者的“圣经”。遵循统一的编程规则，可以让开发新手少走好多弯路，可以让项目整体的质量维持一个起码的“质量底线”。

有关内存泄漏方面的规则主要是“内存管理”方面的，举几个简单的，如下

×用malloc或new申请内存之后，立即检查指针值是否为NULL（防止使用指针值为NULL的内存）

×动态内存的申请与释放是否配对（防止内存泄漏）

×malloc语句是否正确无误？例如字节数是否正确？类型转换是否正确

×是否出现野指针，例如用free或delete释放了内存之后，忘记将指针设置为NULL

... ...

第二步：积极主动检测“内存泄漏”

严格遵循好的编程规则，可以让程序员在代码中尽量少的引入bug，但一旦不小心引入了，怎么办？这就要求我们在单元测试和集成测试中严格把关。

在这个阶段，单靠程序员或者测试员通过“代码走查”的方式检查内存泄漏，客户的实践和我的经验告诉我，这将是“不切实际”的，无论效率还是时间。如果能够借助于一些专业的工具的话，情况可能就不一样了。

如果你的程序是用Visual C++ 开发，那么Numega的BoundsChecker将是你检测“内存泄漏”最好的选择，如果是Visual C++.NET，可以试一下Compuware的DevPartner。

如果你的程序基于Unix或者Linux平台，使用C或者C++，可以考虑一下开源的工具valgrind，我的朋友跟我说，它在一定程度上比Rational的Purify更出色。

上面的工具都要求程序能够动态运行起来，而且测试用例需要你自己准备。

如果你正处于单元测试或集成测试阶段，程序代码量已经足够大，而且还不能够动态运行，要尽早检测代码中的“内存泄漏”问题，该怎么办？此时你可以试用一下目前最新的静态分析技术：

×它不要求代码能够动态运行

×也不需要你来编写测试用例

×只需要代码能够正常编译，就可以发现代码只有在执行过程中才出现的错误，当然也包括内存泄漏。

这方面的工具有Klocwork的K7，Coverity的SQS，以及C++test中的BugDetective，其中最“物美价廉”的就是c++test的BugDetective。

2 如何发现客户端软件的“内存泄漏”？

如果开发过程中已经按照我上面提到的去做，相信发布后的程序存在“内存泄漏”的可能性几乎为零。

如果开发过程已经到了后期，系统测试已经开始做了，还要发现内存泄漏，这个时候我希望你能够拿到源代码。如果有源代码，你还可以考虑1中的第二步，借助于专业的工具协助，虽然可能效果不一定特别理想，但总比下面我提到的方法更好一些。

当然作为测试人员，我当然也理解事情总没有想像那么完美。我们通常会碰到“需要在系统测试阶段检测是否有内存泄漏，而且没有源代码”的难题。我曾经也遇到过。

记得那还是2002年的事情了。当时我承接的项目是一个电力行业的自动化系统，分为server端和client端，典型的c/s模式，老板要求在测试功能的同时顺便检查内存泄漏的问题，因为这个client端在客户那里可能是长时间不间断运行的，虽然客户很少操作。我当时很为难，因为没有源代码，我甚至无法做“代码走查”。在做功能测试的同时，我一直在琢磨...... 采用什么手段呢？

最后，借助于WinRunner，我出色的完成了任务，起码我的老板相信我的测试是可信的。我的方法是这样的。

×首先咨询开发方，了解到关于内存操作频繁的功能点和模块

×从我的功能测试用例中挑选出和这些功能点和模块相关的测试用例

×找到一个“纯净”的机器，上面除了操作系统和被测的client端外，没有任何其他应用，这样做是为了排除其他应用可能存在的干扰。

×借助于WinRunner，自动化这些用例，形成自动化的脚本；在脚本的最后，添加“切换到Windows任务管理器”“记录该client进程所占用内存数据到文件”的操作脚本。

×连续运行N个小时

×最后我打开这个数据文件，可以发现在该客户端运行过程中，每次执行完特定的测试用例后，记录的内存占用数据。当时我得出的结论是该client程序有“少许”的内存泄漏，因为在连续运行了72小时后，内存使用增加了近百分之十几。我会把这些数据导入到EXCEL中绘成了一个图表，这样更直观一些。经过简单的计算（内存的增量/用例循环次数），得到用例每次执行后增加的内存使用值，即泄漏的内存数量，然后把操作过程和这个结果一起交给开发方，最后开发方根据我的信息，真的找到了一处有内存泄漏的地方，虽然泄漏的数量很少。

以上就是我有过的一个类似的经历，我觉得可以提供给大家参考，同时也可以“举一反三，融会贯通”。如B/S的客户端控件，可以用QTP协助完成。

在测试的最后阶段要去发现甚至定位内存泄漏挺难的，但只要发挥我们测试人员的主观能动性，总是找到一些“旁门左道”的测试手段。

最后，我个人认为，从时间成本和各种风险考虑，要避免内存泄漏的问题，还是要回到前期的预防，即编程过程的规则检查和单元测试阶段主动的检测。

一家之言，欢迎讨论。