软件工程基础知识

授课：薛大龙

简介：中共党员、北京理工大学博士研究生、多所大学客座教授、北京市评标专家，多次参与软考的**命题与阅卷**，主编出版专著超过60部！

如何判断程序员技术水平

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

一、软件开发模型

薛大龙博士

序号	章节	内容
1	软件开发模型	**开发生命周期模型** 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

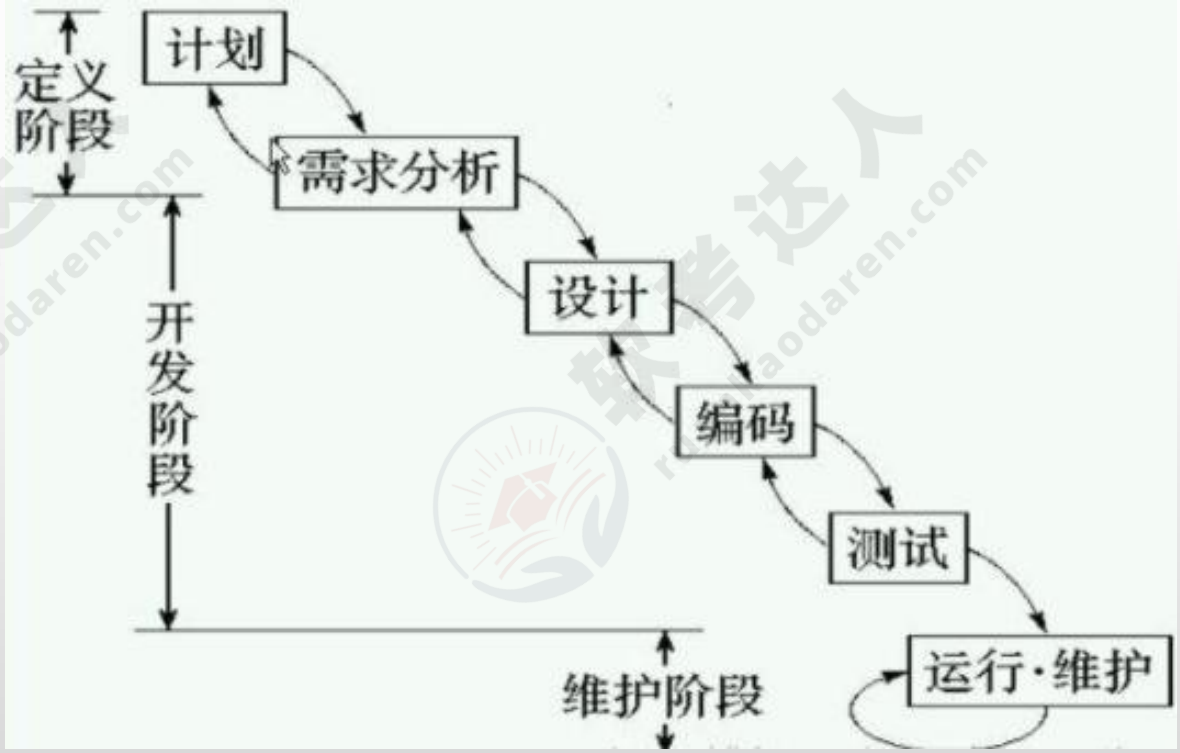
开发生命周期模型

在开发模型知识点中，我们要掌握软件生命周期的概念、各种开发模型的特点和应用场合。主要的开发模型有瀑布模型、增量模型、螺旋模型、喷泉模型、智能模型、V模型、RAD模型、CBSD模型、原型方法、XP方法、RUP方法等。

开发生命周期模型

1.瀑布模型

瀑布模型也称为生命周期法，是生命周期法中最常用的开发模型，它把软件开发的过程分为软件计划、需求分析、软件设计、程序编码、软件测试和运行维护6个阶段，规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。采用瀑布模型的软件过程如图



开发生命周期模型

（1）软件计划（问题的定义及规划）：主要确定软件的开发目标及其可行性。
（2）需求分析：在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析阶段是一个很重要的阶段，这一阶段做得好，将为整个软件开发项目的成功打下良好的基础。
（3）软件设计：主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计、数据库设计等。软件设计一般分为总体设计（概要设计）和详细设计。
（4）程序编码：将软件设计的结果转换成计算机可运行的程序代码。在程序编写中必须制定统一、符合标准的编写规范，以保证程序的可读性，易维护性，提高程序的运行效率。
（5）软件测试：在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性。
（6）软件维护：软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件不能继续适应用户的要求，要延续软件的使用寿命，就必须对软件进行维护。

开发生命周期模型

瀑布模型是最早出现的软件开发模型，在软件工程中占有重要的地位，它提供了软件开发的基本框架。瀑布模型的本质是“一次通过”,即每个活动只做一次，最后得到软件产品，也称做“线性顺序模型”或者“传统生命周期”。

其过程是从上一项活动接收该项活动的工作对象作为输入，利用这一输入实施该项活动应完成的内容，给出该项活动的工作成果，作为输出传给下一项活动；对该项活动实施的工作进行评审，若其工作得到确认，则继续下一项活动，否则返回前项，甚至更前项的活动进行返工。

开发生命周期模型

瀑布模型有利于大型软件开发过程中人员的组织与管理，有利于软件开发方法和工具的研究与使用，从而提高了大型软件项目开发的质量和效率。然而软件开发的实践表明，上述各项活动之间并非完全是自上而下的，而是呈线性图示，因此，瀑布模型存在严重的缺陷。

（1）由于开发模型呈线性，所以当开发成果尚未经过测试时，用户无法看到软件的效果。这样，软件与用户见面的时间间隔较长，也增加了一定的风险。

（2）在软件开发前期未发现的错误传到后面的开发活动中时，可能会扩散，进而可能导致整个软件项目开发失败。

（3）在软件需求分析阶段，完全确定用户的所有需求是比较困难的，甚至可以说是不太可能的。

开发生命周期模型

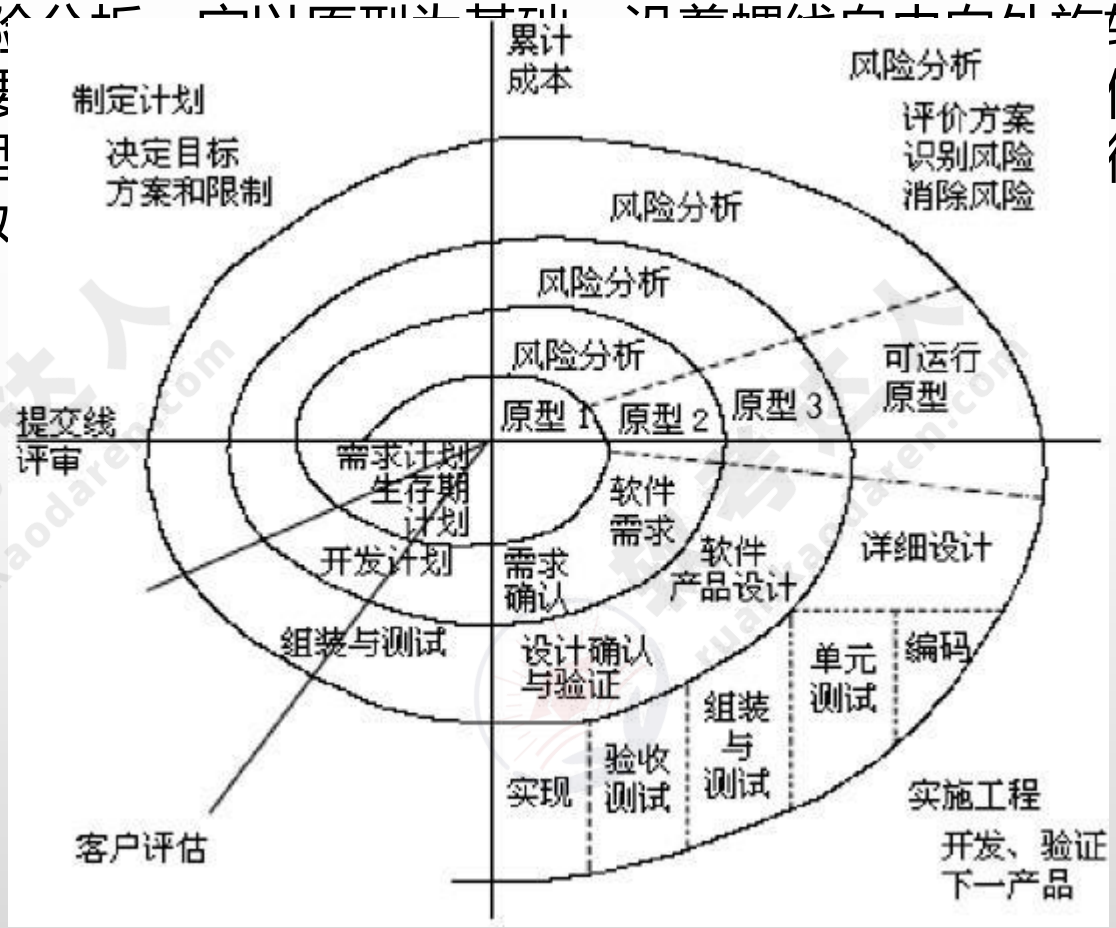
2.变换模型

变换模型（演化模型）是在快速开发一个原型的基础上，根据用户在调用原型的过程中提出的反馈意见和建议，对原型进行改进，获得原型的新版本，重复这一过程，直到演化成最终的软件产品。

开发生命周期模型

3.螺旋模型

螺旋模型将瀑布模型和变换模型相结合，它综合了两者的优点，并增加了风险分析，它以原型为基础，沿着螺线自内向外旋转，每旋转一圈都要进行制定计划、风险分析、实施工程、客户评估等活动，并开发原型的下一个版本，直到得到最终的系统，如图所示。



开发生命周期模型

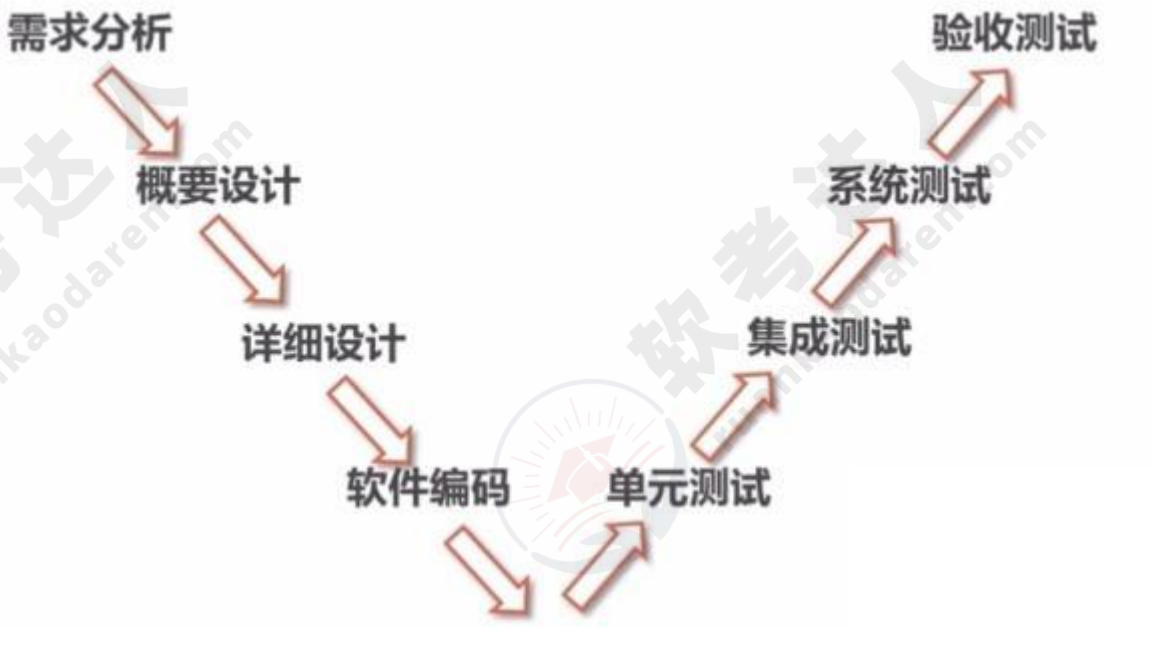
4.喷泉模型

喷泉模型对软件复用和生命周期中多项开发活动的集成提供了支持，主要支持面向对象的开发方法。"喷泉"一词本身体现了迭代和无间隙特性。系统某个部分常常重复工作多次，相关功能在每次迭代中随之加入演进的系统。所谓无间隙是指在开发活动中，分析、设计和编码之间不存在明显的边界，如图

开发生命周期模型

5、V模型

在开发模型中，测试常常作为亡羊补牢的事后行为，但也有以测试为中心的开发模型，那就是V模型。V模型只得到软件业内比较模糊的认可。V模型宣称测试并不是一个事后弥补行为，而是一个同开发过程同样重要的过程，如图



开发生命周期模型

V模型描述了一些不同的测试级别，并说明了这些级别所对应的生命周期中不同的阶段。在图中，左边下降的是开发过程各阶段，与此相对应的是右边上升的部分，即测试过程的各个阶段。请注意在不同的组织中，对测试阶段的命名可能有所不同。

V模型的价值在于它非常明确地表明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系：

（1）单元测试的主要目的是针对编码过程中可能存在的各种错误。例如：用户输入验证过程中的边界值的错误。

（2）集成测试的主要目的是针对详细设计中可能存在的问题，尤其是检查各单元与其他程序部分之间的接口上可能存在的错误。

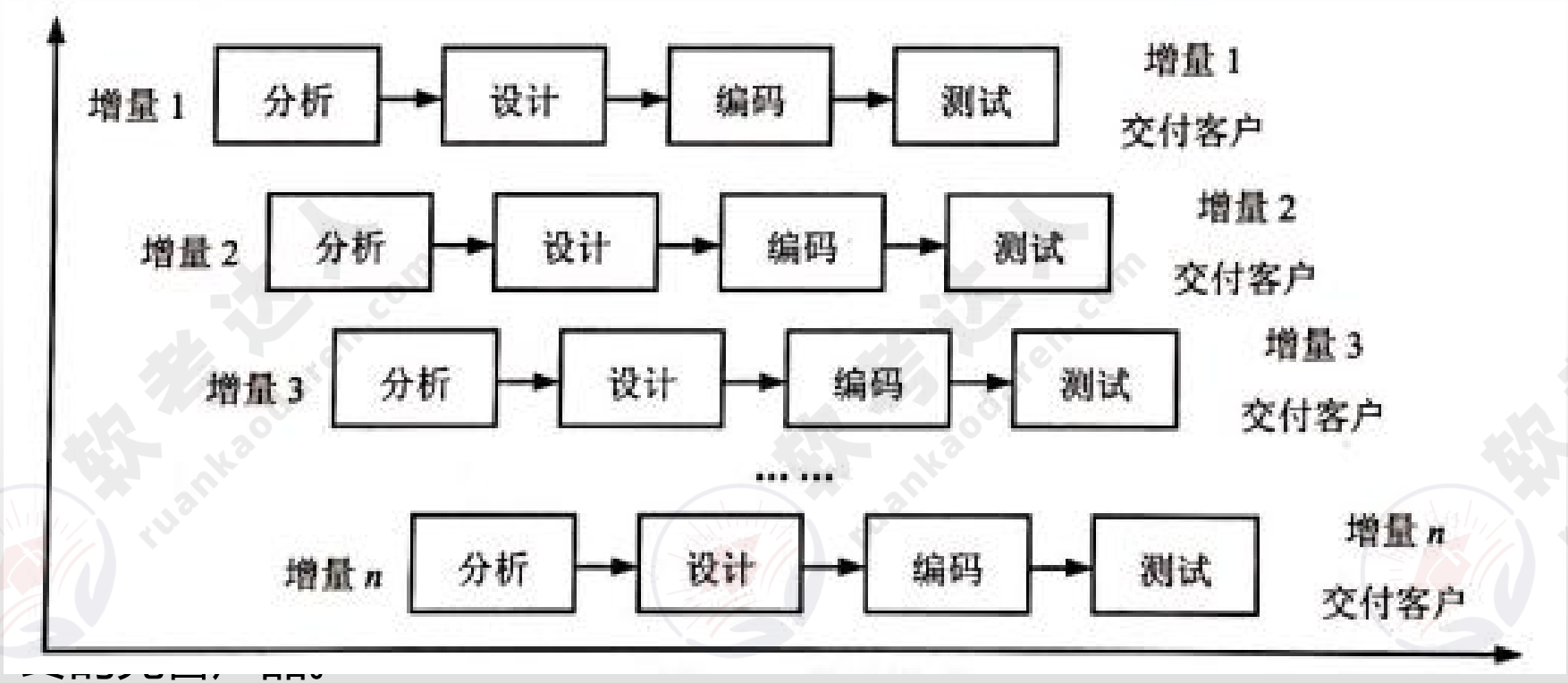
（3）系统测试主要针对概要设计，检查系统作为一个整体是否有效地得到运行。例如：在产品设置中是否达到了预期的高性能。

（4）验收测试通常由业务专家或用户进行，以确认产品能真正符合用户业务上的需要。

开发生命周期模型

6.增量模型

增量模型融合了瀑布模型的基本成分（重复的应用）和原型实



增量模型强调每一个增量均发布一个可操作的产品。

开发生命周期模型

增量模型像原型实现模型和其他演化方法一样，本质上是迭代的。但与原型实现不同的是增量模型强调每一个增量均发布一个可操作产品。早期的增量是最终产品的"可拆卸"版本，但它们确实提供了为用户服务的功能，并且提供了给用户评估的平台。增量模型的特点是引进了增量包的概念，无须等到所有需求都出来，只要某个需求的增量包出来即可进行开发。虽然某个增量包可能还需要进一步适应客户的需求，还需要更改，但只要这个增量包足够小，其影响对整个项目来说是可以承受的。

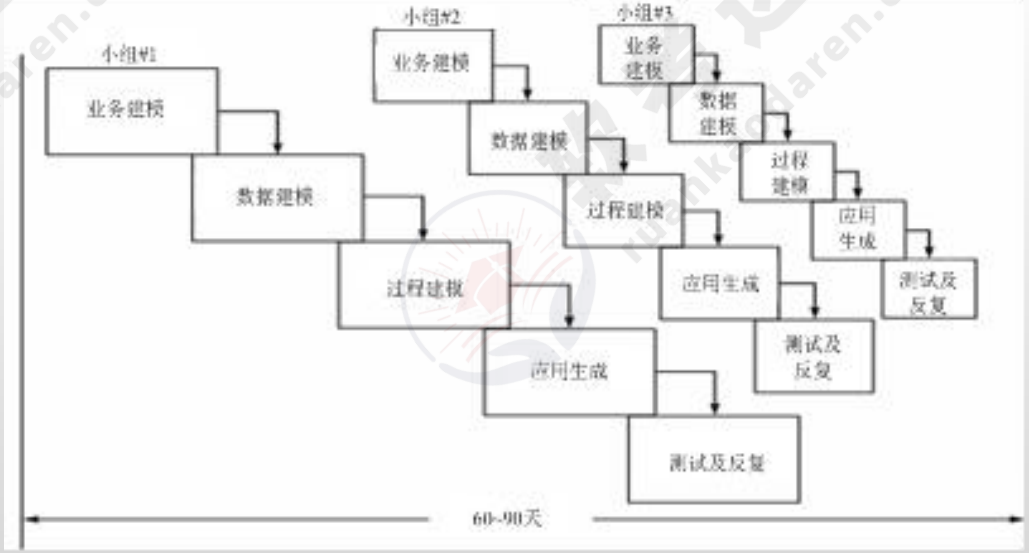
开发生命周期模型

采用增量模型的优点是人员分配灵活，刚开始不用投入大量人力资源，如果核心产品很受欢迎，则可以增加人力实现下一个增量；当配备的人员不能在设定的期限内完成产品时，它提供了一种先推出核心产品的途径，这样就可以先发布部分功能给客户，对客户起到镇静剂的作用。此外，增量能够有计划地管理技术风险。增量模型的缺点是如果增量包之间存在相交的情况且不能很好地处理，就必须做全盘的系统分析。增量模型将功能细化、分别开发的方法适用于需求经常改变的软件开发过程。

开发生命周期模型

7.RAD模型

快速应用开发（Rapid Application Development,RAD）模型是一个增量型的软件开发过程模型，强调极短的开发周期。RAD模型是瀑布模型的一个“高速”变种，通过大量使用可复用构件，采用基于构件的建造方法赢得快速开发。如果需求理解得好且约束了项目的范围，利用这种模型可以很快地创建出功能完善的“信息系统“。其流程从业务建模开始，随后是数据建模、过程建模、应用生成、测试及反复。采用RAD模型的软件过程如图所示。



开发生命周期模型

RAD模型各个活动期所要完成的任务如下。

（1）业务建模：以什么信息驱动业务过程运作？要生成什么信息？谁生成它？信息流的去向是哪里？由谁处理？可以辅之以数据流图。

（2）数据建模：为支持业务过程的数据流，找数据对象集合，定义数据对象属性，与其他数据对象的关系构成数据模型，可辅之以E-R图。

（3）过程建模：使数据对象在信息流中完成各业务功能。创建过程以描述数据对象的增加、修改、删除、查找，即细化数据流图中的处理框。

（4）应用程序生成：利用第四代语言（4GL）写出处理程序，重用已有构件或创建新的可重用构件，利用环境提供的工具自动生成并构造出整个应用系统。

（5）测试与交付，由于大量重用，一般只做总体测试，但新创建的构件还是要测试的。

开发生命周期模型

与瀑布模型相比，RAD模型不采用传统的第三代程序设计语言来创建软件，而是采用基于构件的开发方法，复用已有的程序结构（如果可能的话）或使用可复用构件，或创建可复用的构件（如果需要的话）。在所有情况下，均使用自动化工具辅助软件创造。很显然，加在一个RAD模型项目上的时间约束需要"一个可伸缩的范围".如果一个业务能够被模块化使得其中每一个主要功能均可以在不到三个月的时间内完成，那么它就是RAD的一个候选者。每一个主要功能可由一个单独的RAD组来实现，最后再集成起来形成一个整体。

开发生命周期模型

RAD模型通过大量使用可复用构件加快了开发速度，对信息系统的开发特别有效。但是像所有其他软件过程模型一样，RAD方法也有其缺陷：

（1）并非所有应用都适合RAD.RAD模型对模块化要求比较高，如果有哪一项功能不能被模块化，那么建造RAD所需要的构件就会有问题；如果高性能是一个指标，且该指标必须通过调整接口使其适应系统构件才能赢得，RAD方法也有可能不能奏效。

（2）开发者和客户必须在很短的时间完成一系列的需求分析，任何一方配合不当都会导致RAD项目失败。

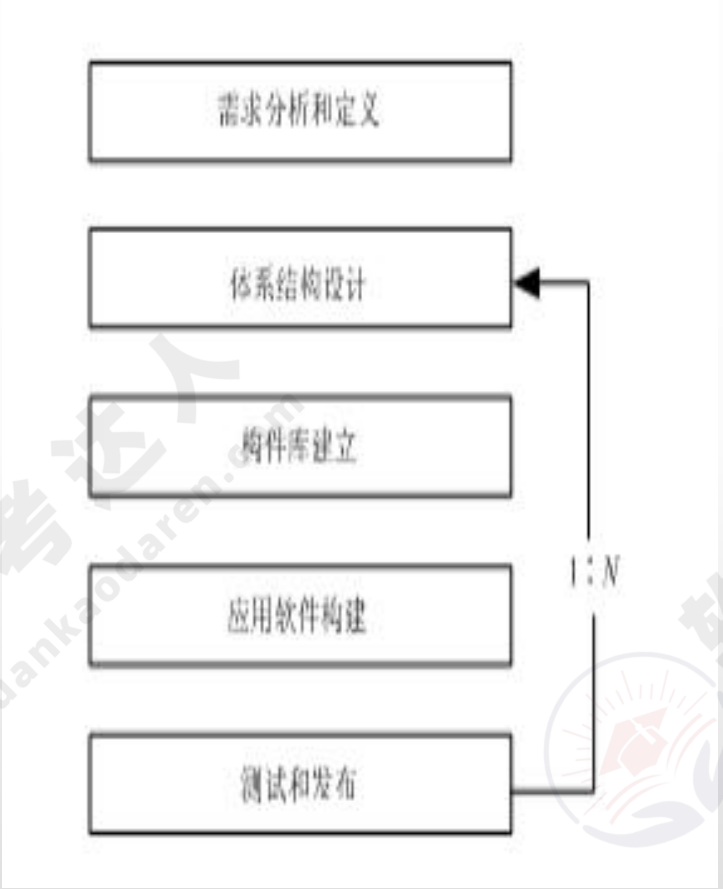
（3）RAD只能用于信息系统开发，不适合技术风险很高的情况。当一个新应用要采用很多新技术或当新软件要求与已有的计算机程序有较高的互操作性时，这种情况就会发生。

开发生命周期模型

8.基于构件的模型

构件（Component,组件）是一个具有可重用价值的、功能相对独立的软件单元。基于构件的软件开发（Component Based Software Development,CBSD）模型是利用模块化方法，将整个系统模块化，并在一定构件模型的支持下，复用构件库中的一个或多个软件构件，通过组合手段高效率、高质量地构造应用软件系统的过程。

基于构件的开发模型融合了螺旋模型的许多特征，本质上是演化型的，开发过程是迭代的。基于构件的开发模型由软件的需求分析和定义、体系结构设计、构件库建立、应用软件构建、测试和发布5个阶段组成。采用基于构件的开发模型的软件过程如图



开发生命周期模型

构件作为重要的软件技术和工具得到了极大的发展，这些新技术和工具有Microsoft的DCOM,Sun的EJB,OMG的CORBA等。基于构件的开发活动从标识候选构件开始，通过搜索已有构件库，确认所需要的构件是否已经存在，如果已经存在，就从构件库中提取出来复用；如果不存在，就采用面向对象方法开发它。在提取出来的构件通过语法和语义检查后，将这些构件通过胶合代码组装到一起实现系统，这个过程是迭代的。

开发生命周期模型

基于构件的开发方法使得软件开发不再一切从头开始，开发的过程就是构件组装的过程，维护的过程就是构件升级、替换和扩充的过程，其优点是构件组装模型导致了软件的复用，提高了软件开发的效率；构件可由一方定义其规格说明，被另一方实现，然后供给第三方使用；构件组装模型允许多个项目同时开发，降低了费用，提高了可维护性，可实现分步提交软件产品。
缺点是由于采用自定义的组装结构标准，缺乏通用的组装结构标准，引入具有较大的风险；可重用性和软件高效性不易协调，需要精干的、有经验的分析人员和开发人员，一般的开发人员插不上手，客户的满意度低；过分依赖于构件，构件库的质量影响着产品质量。

开发生命周期模型

9.原型方法

软件原型是所提出的新产品的部分实现，建立原型的主要目的是为了解决[illegible]问题，其目的是明确并完善[illegible]的产品。

原型有[illegible]来分，软件原型可分为水平[illegible]行为原型，用来探索预期系[illegible]目的。水平原型通常只是功[illegible]原型主要用在界面上。垂直[illegible]功能。垂直原型主要用在复[illegible]

开发生命周期模型

从原型的最终结果来分，软件原型可分为抛弃型原型和演化型原型。抛弃型原型也称为探索型原型，是指达到预期目的后，原型本身被抛弃。抛弃型原型主要用在解决需求不确定性、二义性、不完整性、含糊性等。演化型原型为开发增量式产品提供基础，是螺旋模型的一部分，也是面向对象软件开发过程的一部分。演化型原型主要用在必须易于升级和优化的项目，适用于Web项目。

有些文献把原型分为实验型、探索型和演化型。探索型原型的目的是要弄清对目标系统的要求，确定所希望的特性，并探讨多种方案的可行性。实验型原型用于大规模开发和实现之前，考核方案是否合适，规格说明是否可靠。进化型原型的目的不在于改进规格说明，而是将系统建造得易于变化，在改进原型的过程中，逐步将原型进化成最终系统。

开发生命周期模型

还有些文献也把原型分为抛弃式原型、演化式原型和递增式原型。

原型法适合于用户没有肯定其需求的明确内容的时候。它是先根据已给的和分析的需求，建立一个原始模型，这是一个可以修改的模型（在生命周期法中，需求分析成文档后一般不再多修改）。

在软件开发的各个阶段都把有关信息相互反馈，直至模型的修改，使模型渐趋完善。在这个过程中，用户的参与和决策加强了，最终的结果是更适合用户的要求。

这种原型技术又可分为三类：抛弃式、演化式和递增式。这种原型法成败的关键及效率的高低在于模型的建立及建模的速度。

开发生命周期模型

10.XP方法

XP是一种轻量（敏捷）、高效、低风险、柔性、可预测、科学而且充满乐趣的软件开发方式。与其他方法论相比，其最大的不同在于：

（1）在更短的周期内，更早地提供具体、持续的反馈信息。

（2）迭代地进行计划编制，首先在最开始迅速生成一个总体计划，然后在整个项目开发过程中不断地发展它。

（3）依赖于自动测试程序来监控开发进度，并及早地捕获缺陷。

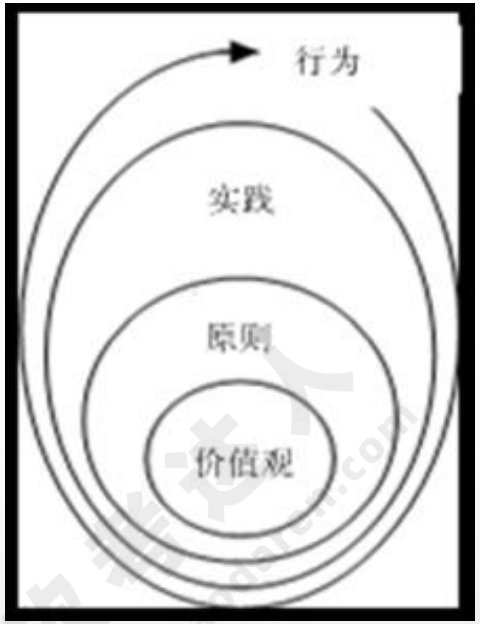
（4）依赖于口头交流、测试和源程序进行沟通。

（5）倡导持续的演化式的设计。

（6）依赖于开发团队内部的紧密协作。

（7）尽可能达到程序员短期利益和项目长期利益的平衡。

如图所示，XP由价值观、原则、实践和行为四个部分组成，它们彼此相互依赖、关联，并通过行为贯穿于整个生命周期。



开发生命周期模型

XP的核心是其总结的4大价值观，即沟通、简单、反馈和勇气。它们是XP的基础，也是XP的灵魂。XP的5个原则是快速反馈、简单性假设、逐步修改、提倡更改和优质工作。而在XP方法论中，贯彻的是"小步快走"的开发原则，因此工作质量决不可打折扣，通常采用测试先行的编码方式来提供支持。

在XP中，集成了12个最佳实践：计划游戏、小型发布、隐喻、简单设计、测试先行、重构、结对编程、集体代码所有制、持续集成、每周工作40小时、现场客户、编码标准。

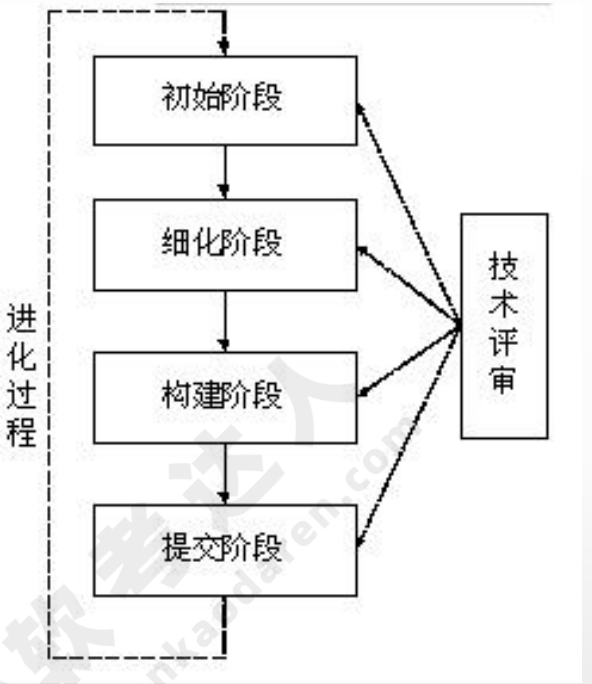
开发生命周期模型

11.RUP方法

RUP（Rational Unified Process）是一个统一的软件开发过程，是一个通用过程框架，可以应付种类广泛的软件系统、不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。RUP是基于构件的，这意味着利用它开发的软件系统是由构件构成的，构件之间通过定义良好的接口相互联系。在准备软件系统所有蓝图的时候，RUP使用的是统一建模语言UML.

与其他软件过程相比，RUP具有三个显著的特点：用例驱动、以基本架构为中心、迭代和增量。

RUP中的软件过程在时间上被分解为四个顺序的阶段，分别是初始阶段、细化阶段、构建阶段和交付阶段。每个阶段结束时都要安排一次技术评审，以确定这个阶段的目标是否已经满足。如果评审结果令人满意，就可以允许项目进入下一个阶段。基于RUP的软件过程模型如图



开发生命周期模型

图中可以看出：基于RUP的软件过程是一个迭代过程。通过初始、细化、构建和提交四个阶段就是一个开发周期，每次经过这四个阶段就会产生一代软件。除非产品退役，否则通过重复同样的四个阶段，产品将演化为下一代产品，但每一次的侧重点都将放在不同的阶段上。这些随后的过程称为演化过程。

开发生命周期模型

在进度和工作量方面，所有阶段都各不相同。尽管不同的项目有很大的不同，但一个中等规模项目的典型初始开发周期应该预先考虑到工作量和进度间的分配，如表

	初始阶段	细化阶段	构建阶段	提交阶段
工作量	5%	20%	65%	10%
进度	10%	30%	50%	10%

对于演进周期，初始和细化阶段就小得多了。能够自动完成某些构建工作的工具将会缓解此现象，并使得构建阶段比初始阶段和细化阶段的总和还要小很多。

*RUP*的工作流程分为两部分：核心工作流程与核心支持工作流程。核心工作流程（在项目中的流程）包括业务需求建模、分析设计、实施、测试、部署；核心支持工作流程（在组织中的流程）包括环境、项目管理、配置与变更管理。

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

系统开发方法论

系统的开发方法主要包括结构化分析与设计、面向数据结构的设计、面向对象分析与设计以及构件化方法四种。

1.结构化分析与设计

这种方法采用结构化技术来完成软件开发的各项任务。该方法把软件生命周期的全过程依次划分为若干阶段，然后顺序地完成每个阶段的任务，与瀑布模型有很好的结合度，是与其最相适应的开发方法。

结构化方法的核心思想是"自顶向下，逐步分解".

系统开发方法论

2.面向数据结构的设计

数据的输入、存储都涉及不同的数据结构，面向数据结构设计方法的基本思想是根据数据结构导出程序结构。典型的面向数据结构的设计方法包括Jackson方法和Warnier方法。

Jackson方法的基本步骤：先建立系统的数据结构；接着以数据结构为基础，对应地建立程序结构；列出程序中要用到的各种基本操作，然后将操作分配到适当的模块中去。

面向数据结构的设计方法并没有明显地使用软件结构的概念，对于模块独立性原则也重视不足，因此并不适合于复杂的软件系统。

系统开发方法论

3.面向对象分析与设计

这种方法引入了"对象"的概念，将数据和方法封装在一起，提高了模块的聚合度，降低了耦合度，更大程度上支持软件复用。面向对象方法是现在最流行和最具有发展前景的软件开发方法

系统开发方法论

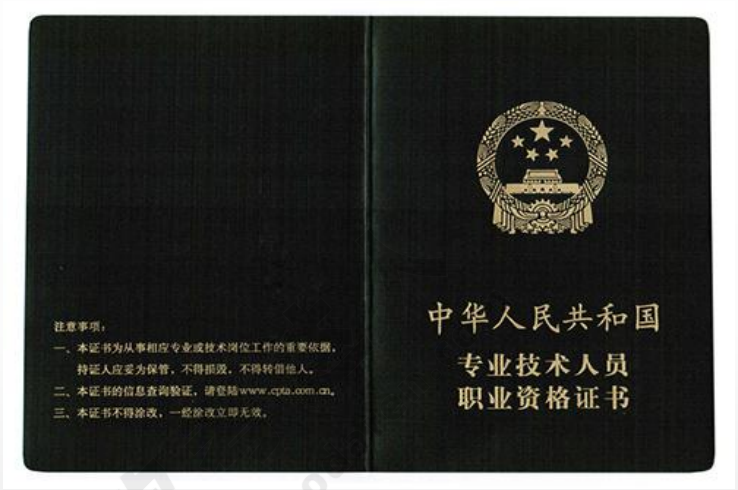
4.构件化开发

为了降低开发费用、提高生产率，以及在快速的技术演化面前提供受控的系统升级的开发方式，就催生了基于构件的软件开发（Component-Based Software Development,CBSD）。它通过有计划地集成现有的软件部分来进行软件开发。它可以有效地遏制复杂性、缩短发布时间、提高一致性，更有效地利用本领域的最佳方法、提高生产率、增加项目进度的可视性、支持并行和分布式的开发、减少维护费用。采用CBSD后，所有的软件解决方案将可以使用预建的构件和模板，像"搭积木"式地建造。

这种"积木"就是构件（组件），构件是一个功能相对独立的具有可重用价值的软件单元。在面向对象方法中，一个构件由一组对象构成，包含了一些协作类的集合，它们共同工作来提供一种系统功能。构件具有5个基本要素：规格说明、一个或多个实现、受约束的构件标准、包装方法和部署方法。

系统开发方法论

可重用性是指系统和（或）其组成部分能在其他系统中重复使用的程度。软件开发的全生命周期都有可重用的价值，包括项目的组织、软件需求、设计、文档、实现、测试方法和测试用例，都是可以被重复利用和借鉴的有效资源。可重用性体现在软件的各个层次，通用的、可复用性高的软件模块往往已经由操作系统或开发工具提供，如通用库、标准组件和标准模板库等，并不需要程序员重新开发。



授 课 人：薛大龙博士

主讲领域：系统分析师、系统架构设计师、信息系统项目管理师、系统规划与管理师

51CTO学院

二、软件测试

薛大龙博士

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

测试的目的

软件测试是软件质量保证的主要手段之一，也是在将软件交付给客户之前所必须完成的步骤。目前，软件的正确性证明尚未得到根本的解决，软件测试仍是发现软件错误和缺陷的主要手段。软件测试的目的就是在软件投入生产性运行之前，尽可能多地发现软件产品（主要是指程序）中的错误和缺陷。

1983年，Bill Hetzel在"Complete Guide of Software Testing"一书中指出："测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量".Grenford J. Myers在"The Art of Software Testing"一书中指出：

（1）软件测试是为了发现错误而执行程序的过程。
（2）测试是为了证明程序有错，而不是证明程序无错误。
（3）一个好的测试用例是在于它能发现至今未发现的错误。
（4）一个成功的测试是发现了至今未发现的错误的测试。

测试的目的

这种观点可以提醒人们测试要以查找错误为中心，而不是为了演示软件的正确功能。但是仅凭字面意思理解这一观点可能会产生误导，认为发现错误是软件测试的唯一目的，查找不出错误的测试就是没有价值的，事实并非如此。

首先，测试并不仅仅是为了要找出错误。通过分析错误产生的原因和错误的分布特征，可以帮助项目管理者发现当前所采用的软件过程的缺陷，以便改进。同时，这种分析也能帮助我们设计出有针对性的检测方法，改善测试的有效性。

其次，没有发现错误的测试也是有价值的，完整的测试是评定测试质量的一种方法。

因此，软件测试可以验证软件是否满足软件需求规格说明和软件设计所规定的功能、性能及其软件质量特性的要求，为软件质量的评价提供依据。我们要注意的是软件测试只是软件质量保证的手段之一，不能单凭测试来保证软件质量。

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

测试的类型

软件测试方法一般分为两大类，分别为动态测试和静态测试。

1.动态测试

动态测试指通过运行程序发现错误，分为黑盒测试法、白盒测试法和灰盒测试法。不管是哪一种测试，都不能做到穷尽测试，只能选取少量最有代表性的输入数据，以期用较少的代价暴露出较多的程序错误。这些被选取出来的数据就是测试用例（一个完整的测试用例应该包括输入数据和期望的输出结果）。

（1）黑盒法

把被测试对象看成一个黑盒子，测试人员完全不考虑程序的内部结构和处理过程，只在软件的接口处进行测试，依据需求规格说明书，检查程序是否满足功能要求。常用的黑盒测试用例的设计方法有等价类划分、边值分析、错误猜测、因果图和功能图等。

测试的类型

等价类划分：将所有可能的输入数据，划分为等价的部分，然后从每个部分中选取少数有代表性的数据作为测试用例。等价类可以分为有效等价类（即合理、有意义的数据集合）、无效等价类（即不合理、无意义的数据集合）两种。而在选取测试用例时，应遵从"设计一个新的测试用例时，应尽可能多地覆盖尚未覆盖的有效等价类；但每次应仅覆盖一个尚未覆盖的无效等价类"的原则。

边界值分析：它是对等价类划分法的一个补充，即选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据。

错误推测法：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。

测试的类型

因果图：等价类划分、边界值分析都只考虑了输入条件，未考虑输入条件间的联系，而因果图则用来描述多种条件组合的测试用例，其最终生成的结果是判定表。它首先基于规格说明书分析原因（等价类）和结果（输出条件）；然后找出原因与结果之间的关系，画出因果图；在因果图上加上约束或限制条件；将其转换为判定表；根据判定表得出测试用例。

功能图：它是由状态迁移图和逻辑功能模型构建的，状态迁移图用于表示输入数据序列以及相应的输出数据；逻辑功能模型用于表示在状态中输入条件与输出条件之间的对应关系。测试用例则是由测试中经过的一系列状态和在每个状态中必须依靠输入/输出数据满足的一对条件组成的。

测试的类型

（２）白盒法

把测试对象看作一个打开的盒子，测试人员须了解程序的内部结构和处理过程，以检查处理过程的细节为基础，对程序中尽可能多的逻辑路径进行测试，检验内部控制结构和数据结构是否有错，实际的运行状态与预期的状态是否一致。 由于白盒测试是结构测试，所以被测对象基本上是源程序，以程序的内部逻辑为基础设计测试用例。常用的白盒测试用例设计方法有基本路径测试、循环覆盖测试、逻辑覆盖测试。

逻辑覆盖：以程序内部逻辑为基础的测试技术，如常用的语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、点覆盖、边覆盖、路径覆盖。

语句覆盖是指选择足够多的测试用例，使得运行这些测试用例时，被测程序的每个语句至少执行一次。很显然，语句覆盖是一种很弱的覆盖标准。

测试的类型

判定覆盖又称分支覆盖，它的含义是不仅每个语句至少执行一次，而且每个判定的每种可能的结果（分支）都至少执行一次。判定覆盖比语句覆盖强，但对程序逻辑的覆盖程度仍然不高。

条件覆盖的含义是不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。条件覆盖不一定包含判定覆盖，判定覆盖也不一定包含条件覆盖。

同时满足判定覆盖和条件覆盖的逻辑覆盖称为判定/条件覆盖。它的含义是选取足够的测试用例，使得判定表达式中每个条件的所有可能结果至少出现一次，而且每个判定本身的所有可能结果也至少出现一次。

条件组合覆盖的含义是选取足够的测试用例，使得每个判定表达式中条件结果的所有可能组合至少出现一次。显然，满足条件组合覆盖的测试用例，也一定满足判定/条件覆盖。因此，条件组合覆盖是上述五种覆盖标准中最强的一种。然而，条件组合覆盖还不能保证程序中所有可能的路径都至少经过一次。

测试的类型

路径覆盖的含义是选取足够的测试用例，使得程序的每条可能执行到的路径都至少经过一次（如果程序中有环路，则要求每条环路径至少经过一次）。路径覆盖实际上考虑了程序中各种判定结果的所有可能组合，因此是一种较强的覆盖标准。但路径覆盖并未考虑判定中的条件结果的组合，并不能代替条件覆盖和条件组合覆盖。

循环覆盖：单循环及嵌套循环。

基本路径法：在程序控制流程图的基础上，通过分析控制构造的环路复杂性导出基本路径集合。然后设计测试用例，保证这些路径至少通过一次。

测试的类型

（3）灰盒法

灰盒测试是一种介于白盒测试与黑盒测试之间的测试，它关注输出对于输入的正确性。同时也关注内部表现，但这种关注不像白盒测试那样详细且完整，而只是通过一些表征性的现象、事件及标志来判断程序内部的运行状态。

灰盒测试结合了白盒测试和黑盒测试的要素，考虑了用户端、特定的系统知识和操作环境，在系统组件的协同性环境中评价应用软件的设计。

测试的类型

2.静态测试

静态测试指被测试程序不在机器上运行，而是采用人工检测和计算机辅助静态分析的手段对程序进行检测。静态分析中进行人工测试的主要方法有桌前检查（Desk Checking）、代码审查和代码走查。经验表明：使用这种方法能够有效地发现30%～70%的逻辑设计和编码错误。

（1）桌前检查

由程序员自己检查自己编写的程序。程序员在程序通过编译之后，进行单元测试设计之前，对源程序代码进行分析、检验，并补充相关的文档，目的是发现程序中的错误。这种桌前检查，由于程序员熟悉自己的程序和自身的程序设计风格，可以节省很多的检查时间，但应避免主观片面性。

测试的类型

（２）代码审查

代码审查是由若干程序员和测试员组成一个会审小组，通过阅读、讨论和争议，对程序进行静态分析的过程。代码审查分两步：

第一步，小组负责人提前把设计规格说明书、控制流程图、程序文本及有关要求、规范等分发给小组成员，作为评审的依据。小组成员在充分阅读这些材料之后，进入审查的第二步。

第二步，召开程序审查会。在会上，首先由程序员逐句讲解程序的逻辑。在此过程中，程序员或其他小组成员可以提出问题，展开讨论，审查错误是否存在。实践表明，程序员在讲解过程中能发现许多原来自己没有发现的错误，而讨论和争议则促进了问题的暴露。

在会前，应当给会审小组每个成员准备一份常见错误的清单，把以往所有可能发生的常见错误罗列出来，供与会者对照检查，以提高会审的实效。这个常见错误清单也叫做检查表，它把程序中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误，然后把它们制成表格，供在会审时使用。这种检查表类似于单元测试中给出的检查表。

测试的类型

（3）代码走查

代码走查与代码审查基本相同，其过程也分为两步。

第一步，把材料先发给走查小组每个成员，让他们认真研究程序，然后再开会。

第二步，开会的程序与代码审查不同，不是简单地读程序和对照错误检查表进行检查，而是让与会者"充当"计算机。即首先由测试组成员为被测程序准备一批有代表性的测试用例，提交给走查小组。走查小组开会，集体扮演计算机角色，让测试用例沿程序的逻辑运行一遍，随时记录程序的踪迹，供分析和讨论用。

值得说明的是使用静态测试的方法也可以实现白盒测试。例如：使用人工检查代码的方法来检查代码的逻辑问题，也属于白盒测试范畴。

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

测试的阶段

为了保证系统的质量和可靠性，应力求在分析、设计等各个开发阶段结束前，对软件进行严格的技术评审。而软件测试是为了发现错误而执行程序的过程。

根据测试的目的、阶段的不同，可以把测试分为单元测试、集成测试、确认测试、系统测试等种类。

1.单元测试

单元测试又称为模块测试，是针对软件设计的最小单位（程序模块）进行正确性检验的测试工作。其目的在于检查每个程序单元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，发现各模块内部可能存在的各种错误。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。

单元测试根据详细设计说明书，包括模块接口测试、局部数据结构测试、路径测试、错误处理测试和边界测试，单元测试通常由开发人员自己负责。而由于通常程序模块不是单独存在的，因此常常要借助驱动模块（相当于用于测试模拟的主程序）和桩模块（子模块）完成。单元测试的计划通常是在软件详细设计阶段完成的。

测试的阶段

2.集成测试

集成测试也称为组装测试、联合测试（对于子系统而言，则称为部件测试）。它主要是将已通过单元测试的模块集成在一起，主要测试模块之间的协作性。集成测试计划通常在软件概要设计阶段完成。

从组装策略而言，可以分为一次性组装和增量式组装，增量式组装又包括自顶向下、自底向上、混合式三种，其中混合式组装又称为三明治测试。

（1）自顶向下集成测试是一种构造程序结构的增量实现方法。模块集成的顺序是首先集成主控模块（主程序），然后按照控制层次结构向下进行集成。隶属于（和间接隶属于）主控模块的模块按照深度优先或者广度优先的方式集成到整个结构中去。

测试的阶段

（２）自底向上集成测试是从原子模块（比如在程序结构的最低层的模块）开始来进行构造和测试的，跟自顶向下集成测试相反。

（３）三明治式测试是一种组合的折中测试策略，从"两头"往"中间"测试，其在程序结构的高层使用自顶向下策略，而在下面的较低层中使用自底向上策略，类似于"两片面包间夹馅的三明治"而得名。

软件集成的过程是一个持续的过程，会形成多个临时版本。在不断的集成过程中，功能集成的稳定性是真正的挑战。在每个版本提交时，都需要进行冒烟测试，即对程序主要功能进行验证。冒烟测试也称为版本验证测试或提交测试。

测试的阶段

3.确认测试

确认测试也称为有效性测试，主要验证软件的功能、性能及其他特性是否与用户要求（需求）一致。确认测试计划通常在需求分析阶段完成。根据用户的参与程度，通常包括4种类型：

（1）内部确认测试：主要由软件开发组织内部按软件需求说明书进行测试。

（2）a测试（Alpha测试）：由用户在开发环境下进行测试。

（3）b测试（Beta测试）：由用户在实际使用环境下进行测试。

（4）验收测试：针对软件需求说明书，在交付前由用户为主进行的测试。

测试的阶段

4.系统测试

如果项目不只包含软件，还有硬件和网络等，则要将软件与外部支持的硬件、外设、支持软件、数据等其他系统元素结合在一起，在实际运行环境下，对计算机系统的一系列集成与确认测试。一般地，系统测试的主要内容包括功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、安装与反安装测试等。系统测试计划通常是在系统分析阶段（需求分析阶段）完成的。

不管是哪个阶段的测试，一旦测试出问题，就要进行修改。修改之后，为了检查这种修改是否会引起其他错误，还要对这个问题进行测试，这种测试称为回归测试或退化测试。

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

性能测试

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行，统称为负载压力测试。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

性能测试

1.性能测试的目的

性能测试的目的是验证软件系统是否能够达到用户提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件，最后起到优化系统的目的。具体来说，包括以下几个方面：

（1）评估系统的能力，测试中得到的负荷和响应时间数据可以被用于验证所计划的模型的能力，并帮助作出决策。

（2）识别体系中的弱点：受控的负荷可以被增加到一个极端的水平，并突破它，从而修复体系的瓶颈或薄弱的地方。

（3）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。

（4）检测软件中的问题：长时间的测试执行可导致程序发生由于内存泄露引起的失败，揭示程序中的隐含的问题或冲突。

（5）验证稳定性和可靠性：在一个生产负荷下执行测试一定的时间是评估系统稳定性和可靠性是否满足要求的唯一方法。

性能测试

2.性能测试的类型

性能测试类型包括负载测试，强度测试，容量测试等。

（1）负载测试：负载测试是一种性能测试，指数据在超负荷环境中运行，程序是否能够承担。

（2）强度测试： 强度测试是一种性能测试，表明在系统资源特别低的情况下软件系统运行情况。

（3）容量测试：确定系统可处理同时在线的最大用户数。

性能测试

3.负载压力测试

系统的负载压力测试（负载测试）是指系统在某种指定软件、硬件及网络环境下承受的流量，例如：并发用户数、持续运行时间、数据量等，其中并发用户数是负载压力的重要体现。系统在应用环境下主要承受并发访问用户数、无故障稳定运行时间、大数据量操作等负载压力。

负载压力测试的目的如下：

（１）在真实环境下检测系统性能，评估系统性能是否可以满足系统的性能设计要求。

（２）预见系统负载压力承受力，对系统的预期性能进行评估。

（３）进行系统瓶颈分析、优化系统。

性能测试

在网络应用系统中，负载压力测试应重点关注客户端、网络、服务器（包括应用服务器和数据库服务器）的性能。应获取的关键测试指标如下：

（1）客户端：并发用户数、响应时间、交易通过率以及吞吐量等。

（2）网络：带宽利用率、网络负载、延迟以及网络传输和应用错误等。

（3）服务器：操作系统的CPU占用率、内存使用、硬盘I/O等；数据库服务器的会话执行情况、SQL执行情况、资源争用以及死锁等；应用服务器的并发连接数、请求响应时间等。

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

第三方测试

第三方测试指独立于软件开发方和用户方的测试，组织的测试也称为"独立测试".软件质量工程强调开展独立验证和确认（IV&V）活动，是由在技术、管理和财务上与开发组织具有规定程序独立的组织执行验证和确认过程。软件第三方测试是相对独立的组织进行的软件测试，一般情况下是在模拟用户真实应用环境下，进行的软件确认测试。

第三方测试机构是一个中介的服务机构，它通过自己专业化的测试手段为客户提供有价值的服务。但是这些服务不同于公司内部的测试，因为第三方测试机构的测试除了发现软件问题之外，还有科学公正地评价软件的职能，这就要求该机构要保持公正、廉洁、客观、科学且独立的态度。

第三方测试

第三方测试机构存在的价值主要是由软件公司、软件用户，以及国家的公正诉求所决定的。对于软件开发商来说，经过第三方测试机构的测试，不仅可以通过专业化的测试手段发现软件错误，帮助开发商提升软件的品质，而且可以对软件有一个客观且科学的评价，有助于开发商认清自己产品的定位。

对于行业主管部门以及软件使用者来说，第三方测试机构可帮助选择合适且优秀的软件产品。而对于一些信息工程项目来说，在验收之前，经过第三方机构的严格测试，可以最大程度地避免信息行业的"豆腐渣"工程。此外，经过国家认可的第三方测试机构，还为国家软件产品的质量监督抽查提供独立公正的测试支持。

在选择第三方测试机构时，主要查看其资质、信息系统工程测评经验、测试环境、测试工具及测试工程师队伍的素质等。

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

面向对象测试基础

面向对象测试是与采用面向对象开发相对应的测试技术，它通常包括4个测试层次，从低到高排列分别是算法层、类层、模板层和系统层。

（1）算法层：测试类中定义的每个方法，基本上相当于传统软件测试中的单元测试。

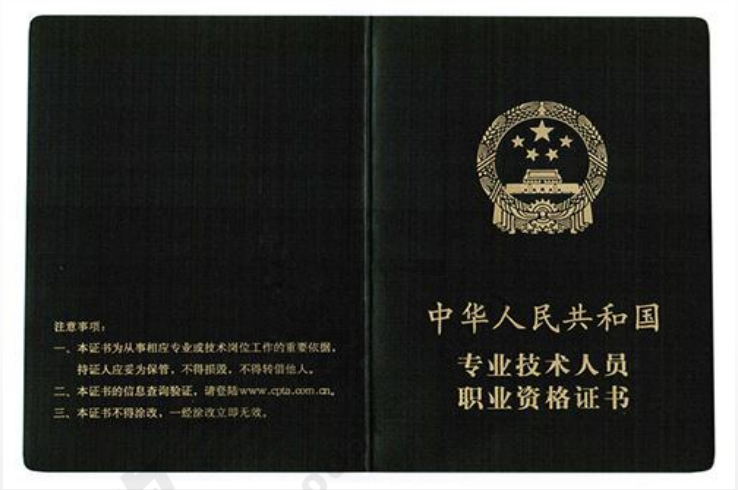
（2）类层：测试封装在同一个类中的所有方法与属性之间的相互作用。在面向对象软件中类是基本模块，因此可以认为这是面向对象测试中所特有的模块（单元）测试。

（3）模板层：也称为主题层，测试一组协同工作的类或对象之间的相互作用。大体上相当于传统软件测试中的子系统测试，但是也有面向对象软件的特点（例如：对象之间通过发送消息相互作用）。

（4）系统层。把各个子系统组装成完整的面向对象软件系统，在组装过程中同时进行测试。

面向对象测试基础

设计测试方案的传统技术，例如：逻辑覆盖、等价划分、边界值分析和错误推测等方法，仍然可以作为测试类中每个方法的主要技术。面向对象测试的主要目标，也是用尽可能低的测试成本和尽可能少的测试方案，发现尽可能多的错误。但是，面向对象程序中特有的封装、继承和多态等机制，也给面向对象测试带来一些新特点，增加了测试和调试的难度。



授 课 人：薛大龙博士

主讲领域：系统分析师、系统架构设计师、信息系统项目管理师、系统规划与管理师

51CTO学院

三、软件维护

薛大龙博士

序号	章节	内容
1	软件开发模型	开发生命周期模型 系统开发方法论
2	软件测试	测试的目的 测试的类型 测试的阶段 性能测试 第三方测试 面向对象测试基础
3	软件维护	软件维护

古月中心相心
10月8日 19:58 来自 HUAWEI Mate 9 Pro
微博搜索的工程师今天结婚你知道嘛！@M鹿M

@丁振凯
服务稳定了，岳父喊我喝酒去了😂😂😂，都是鹿晗干的好事[允悲] 侯马 凤城乡
收起 查看大图 向左旋转 向右旋转

丁振凯
昨天 21:51 来自 iPhone 6s

首先得给我媳妇道歉，媳妇对不起🌹🌹🌹@琪小双 //@来去之间:👀//@橡实:🐼//@郝瑞:// @developerWorks://@StephanieYR:良心难道不会痛嘛！转起来！都来祝这位工程师新婚快乐！（快乐么[笑而不语]//@古月中心相心:微博搜索的工程师今天结婚你知道嘛！@M鹿M

@丁振凯 :服务稳定了，岳父喊我喝酒去了😂😂😂，都是鹿晗干的好事[允悲] 侯马 · 凤城乡

1284 1192 3653

+ 关注 聊天 ≡ 他的热门

软件维护

在软件维护阶段，主要考查软件维护的分类、软件的可维护性等。

软件经过测试，交付给用户后，在使用和运行过程中可能在软件运行/维护阶段对软件产品进行的修改就是维护。

软件可维护性是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充和压缩的容易程度。目前广泛用来衡量程序可维护性的因素包括可理解性、可测试性、可修改性。

软件维护占整个软件生命周期的60%-80%,维护的类型主要有3种：

（1）改正性维护：为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，应当进行的诊断和改正错误的过程就叫做改正性维护。

软件维护

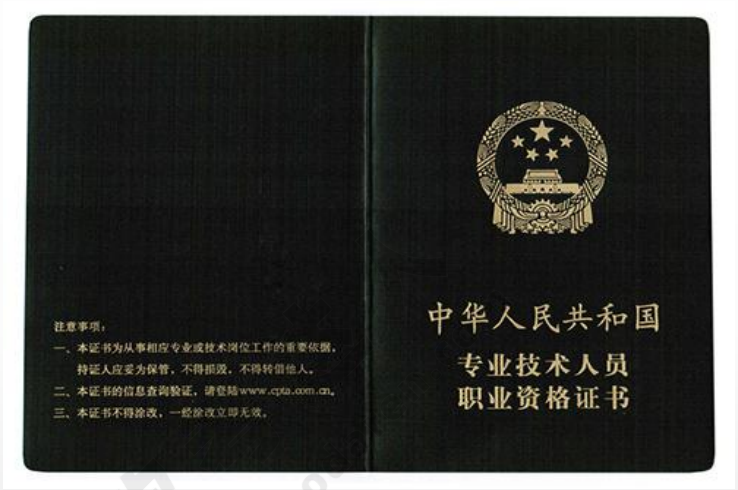
（２）适应性维护：在使用过程中，外部环境（新的硬、软件配置）、数据环境（数据库、数据格式、数据输入/输出方式、数据存储介质）可能发生变化。为使软件适应这种变化，而去修改软件的过程就叫做适应性维护。

（３）完善性维护：在软件的使用过程中，用户往往会对软件提出新的功能与性能要求。为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性。这种情况下进行的维护活动叫做完善性维护。

软件维护

在实际维护工作中，改正性维护大约占20%的工作量，适应性维护大约占25%的工作量，完善性维护大约占50%的工作量。除了这3类维护之外，还有一类维护活动，叫做**预防性维护**。这是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。通常，预防性维护定义为："把今天的方法学用于昨天的系统以满足明天的需要".也就是说，采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试。

在软件开发过程中，错误纠正成本在逐步放大。也就是说，错误发现得越早，纠正错误所花费的成本就会越低，反之则越高。例如：如果在软件设计阶段有个错误未被发现，而待编码阶段时才发现，这时，纠正这个设计错误比纠正源代码错误需要更大的成本。



授 课 人：薛大龙博士

主讲领域：系统分析师、系统架构设计师、信息系统项目管理师、系统规划与管理师

51CTO学院

技术成就梦想

51CTO学院

51CTO学院