软工概述与生命模型

1

1. 软件的定义：

软件是计算机程序、规程，以及运行计算机系统可能需要的相关文档和数据

软件是包括程序、数据及其相关文档的完整集合

程序是按照事先设计的功能和性能要求执行的指令序列；

数据是使程序能正常操纵信息的数据结构；

文档是与程序开发、维护和使用有关的图文材料。

1. 软件的特点：

软件是一种具有抽象的逻辑实体

软件的开发是一种逻辑思维成熟的过程，而无明显的制造过程

软件没有磨损和老化问题，但存在软件退化问题

软件的开发依然很原始

软件的开发和运行常常受到计算机系统的约束和限制，不同程度地依赖计算机硬件

软件是高度复杂的逻辑体

1. 软件的分类：

软件完成功能所处的层次不同：系统软件(指能与计算机硬件紧密配合在一起，使计算机系统各个部件、相关的软件和数据协调、高效地工作的软件)、应用软件(在特定领域内开发，为特定目的服务的一类软件)、中间件软件:(为了解决分布异构系统的集成问题而开发的软件，是处于操作系统软件与用户的应用软件的中间的通用服务，具有标准的接口和协议)

按照服务对象范围划分：通用软件:操作系统、数据库等、定制软件:企业ERP、卫星控制系统等、可配置软件

按照软件使用方式划分：单机软件、服务器软件、客户端软件

按照软件功能划分：办公软件、网络软件、系统工具软件、系统安全软件、多媒体软件、设计与开发软件、游戏软件、家庭应用软件、行业软件

1. 软件危机：由于落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题的现象
2. 软件危机产生的原因：

软件本身存在复杂性，软件开发和维护所使用的方法不合理

1. 软件危机的具体表现：

无法制定合理的开发计划、软件开发费用进度失控、软件成本难以控制、软件质量存在问题、缺乏文档资料、软件功能无法让用户满意、软件的可维护性差。

1. 软件危机解决途径：软件工程学，即采用工程化的方法从事软件系统的研究和维护。
2. 软件工程定义：

软件工程是为了经济地获得能够在实际机器上高效运行的可靠软件而建立和使用的一系列好的工程化原则。

运用现代科学技术知识来设计并构造计算机程序及为开发、运行和维护这些程序所必需的相关文件资料。

软件工程学是为在成本限额以内按时完成开发和修改软件产品所需的系统生产和维护的技术和管理的学科。

IEEE计算机学会将“软件工程”定义为：⑴应用系统化的、规范化的、定量的方法来开发、运行和维护软件，即：将工程应用到软件；⑵对⑴中各种方法的研究。

1. 软件工程三要素：方法、工具、过程

方法：提供了“如何做”的技术；

工具：提供了自动的或半自动的软件支撑环境 ；

过程：将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。

1. 软件工程的目标：

生产具有正确性、可用性以及开销适宜的软件产品。

1. 软件工程的最终目的：

摆脱手工生产软件的状况，逐步实现软件研制和维护的自动化。

1. 软件工程研究的内容：

软件开发技术（软件工程化的技术）：软件开发方法学、开发过程模型、开发工具、软件工程环境。

软件工程管理（工程化设计的技术）：软件管理学、软件工程经济学、软件心理学等内容。

1. 软件工程的基本原则：

选取适宜的开发模型、采用合适的设计方法、提供高质量的工程支持力度、重视开发过程的管理

1. 软件工程管理的基本原理：

用分阶段的生命周期计划严格管理、坚持进行阶段评审、实行严格的产品控制、采用现代程序设计技术、结果应能清楚地审查、开发小组的人员应少而精、承认不断改进软件工程实践的必要性。

1. 软件工程的一般原理：

抽象、信息隐藏、模块化、局部化、确定性、一致性、完备性、可验证性

1. 软件工程知识域：软件需求、软件设计、软件构造、软件测试、软件维护、软件配置管理、软件工程管理、软件工程过程、软件工程工具和方法、软件质量。
2. 工程项目的三个基本目标：合理的进度、有限的经费、一定的质量
3. PDCA环(戴明环)：Plan(规划)、Do(执行)、Check(检查)、Act(处理)
4. 软件工程过程：

为了获得软件产品，在软件工具的支持下由软件工程师完成的一系列软件工程活动。

1. 软件工程过程包含4种基本的过程活动：

软件规格说明：规定软件的功能及其使用限制；

软件开发：产生满足规格说明的软件；

软件确认：通过有效性验证以保证软件能够满足客户的要求；

软件演进：为了满足客户的变更要求，软件必须在使用过程中进行不断地改进。

1. 软件生命周期：指软件产品从考虑其概念开始，到该软件产品不再使用为止的整个时期，一般包括概念阶段、分析与设计阶段、构造阶段、移交和运行阶段等不同时期。
2. 软件生命周期的六个基本步骤：

制定计划：确定要开发软件系统的总目标；给出功能、性能、可靠性以及接口等方面的要求；完成该软件任务的可行性研究；估计可利用的资源 (硬件，软件，人力等)、成本、效益、开发进度；制定出完成开发任务的实施计划，连同可行性研究报告，提交管理部门审查

需求分析和定义：对用户提出的要求进行分析并给出详细的定义；编写软件需求说明书或系统功能说明书及初步的系统用户手册；提交管理机构评审；

软件设计：概要设计 — 把各项需求转换成软件的体系结构。结构中每一组成部分都是意义明确的模块，每个模块都和某些需求相对应；详细设计 — 对每个模块要完成的工作进行具体的描述，为源程序编写打下基础；编写设计说明书，提交评审。

程序编码：把软件设计转换成计算机可以接受的程序代码，即写成以某一种特定程序设计语言表示的“源程序清单”；写出的程序应当是结构良好、清晰易读的，且与设计相一致的；

软件测试：单元测试，查找各模块在功能和结构上存在的问题并加以纠正；组装测试，将已测试过的模块按一定顺序组装起来；按规定的各项需求，逐项进行有效性测试，决定已开发的软件是否合格，能否交付用户使用；

运行维护：改正性维护：运行中发现了软件中的错误需要修正；

适应性维护：为了适应变化了的软件工作环境，需做适当变更；

完善性维护：为了增强软件的功能需做变更。

1. 影响软件生命周期长短的因素：

不考虑硬件环境的快速发展因素时，……为软件的质量、软件的灵活性和适应能力

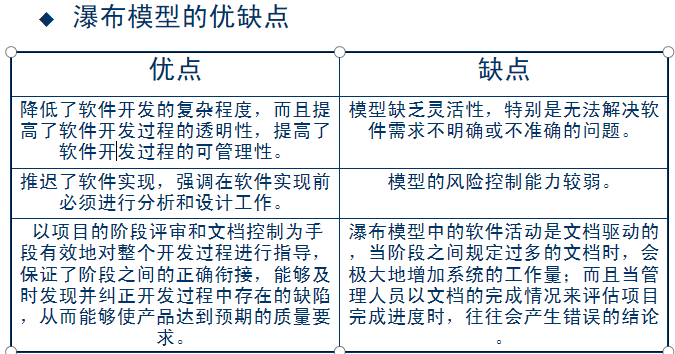
1. 软件过程模型：从一个特定角度提出的对软件过程的概括描述，是对软件开发实际过程的抽象，包括构成软件过程的各种活动（Activities）、软件工件（artifacts）以及参与角色（Actors/Roles）等。
2. 软件生命周期模型：是一个框架，描述从软件需求定义直至软件经使用后废弃为止，跨越整个生存期的软件开发、运行和维护所实施的全部过程、活动和任务，同时描述生命周期不同阶段产生的软件工件，明确活动的执行角色等。是指导软件开发人员按照确定的框架结构和活动进行软件开发的标准。
3. 传统模型种类：瀑布模型、演化模型、增量模型、喷泉模型、V模型和W模型、螺旋模型、构件组装模型、快速应用开发模型、原型方法
4. 瀑布模型：规定了软件生命周期提出的六个基本工程活动，并且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。瀑布模型将软件生命周期划分为定义阶段、开发阶段和维护阶段，在定义阶段部署了计划和需求分析活动；在开发阶段部署了设计、编码和测试活动，维护阶段部署了运行/维护活动。
5. 瀑布模型特征：

本活动的工作对象来自于上一项活动的输出，这些输出一般是代表本阶段活动结束的里程碑式的文档。

根据本阶段的活动规程执行相应的任务。

产生本阶段活动相关产出软件工件，作为下一活动的输入。

对本阶段活动执行情况进行评审。



1. 演化模型：

演化模型提倡两次开发：

第一次是试验开发，得到试验性的原型产品，其目标只是在于探索可行性，弄清软件需求；

第二次在此基础上获得较为满意的软件产品。

演化模型分类：探索式演化模型、抛弃式演化模型

演化模型的特点：

优点：明确用户需求、提高系统质量、降低开发风险；

缺点：难于管理、结构较差、技术不成熟；

演化模型适用范围：需求不清楚；小型或中小型系统；开发周期短。

1. 增量模型：

增量模型的优点：

有利于增加客户对系统的信心；降低系统失败风险；提高系统可靠性；提高了系统的稳定性和可维护性；

增量模型的缺点：增量粒度难以选择；确定所有的基本业务服务比较困难。

1. 螺旋模型：针对大型软件项目开发，最大特点就是引入了明确的风险管理。

四个象限：

制定计划：确定软件项目目标；明确对软件开发过程和软件产品的约束；制定详细的项目管理计划；根据当前的需求和风险因素，制定实施方案，并进行可行性分析，选定一个实施方案，并对其进行规划。

风险分析：明确每一个项目风险，估计风险发生的可能性、频率、损害程度，并制定风险管理措施规避这些风险。

实施工程：针对每一个开发阶段的任务要求执行本开发阶段的活动。

客户评估：客户使用原型，反馈修改意见；根据客户的反馈，对产品及其开发过程进行评审，决定是否进入螺旋线的下一个回路。

优点：设计具有灵活性，可在项目各个阶段进行变更；用户始终参与各阶段开发，保证项目方向正确性与项目可控性。

缺点：需要丰富风险评估经验与专门知识；过多迭代次数会增加开发成本，延迟提交时间。

1. 喷泉模型：面向对象驱动(RUP本质上属于喷泉模型)

喷泉模型也称迭代模型，认为软件开发过程的各个阶段是相互重叠和多次反复的，就象喷泉一样，水喷上去又可以落下来，既可以落在中间，又可以落到底部。各个开发阶段没有特定的次序要求，完全可以并行进行，可以在某个开发阶段中随时补充其他任何开发阶段中遗漏的需求。

优点：提高开发效率、缩短开发周期

缺点：难于管理

1. 构件组装模型：利用模块化思想将整个系统模块化，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，通过组装高效率、高质量地构造软件系统。构件组装模型本质上是演化的，开发过程是迭代的。



优点： 充分利用软件复用，提高了软件开发的效率

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： 缺乏通用的构件组装结构标准，风险较大；

构件可重用性和系统高效性之间不易协调；

由于过分依赖于构件，构件质量影响着最终产品的质量。

1. 快速应用开发（Rapid Application Development，RAD）：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模型的缺点：

并非所有应用都适合采用RAD，如果一个应用不能被模块化，那么构造应用的构件就无法快速获取；

由于时间约束，开发人员和客户必须在较短的时间内完成一系列的需求分析，沟通配合不当都会导致应用RAD模型的失败；

RAD适合于管理信息系统的开发，对于其他类型的应用系统，如技术风险较高、与外围系统的互操作性较高等，不太合适。

1. 原型方法：原型指模拟某种最终产品的原始模型；

原型方法主要用于明确需求，但也可以用于软件开发的其他阶段。

原型的三种作用类型：

（1）探索型：弄清用户对目标系统的要求，确定所期望的特性；探讨多种实现方案的可行性。主要针对需求模糊、用户和开发者对项目开发都缺乏经验的情况。

（2）实验型；用于大规模开发和实现之前，考核技术实现方案是否合适、分析和设计的规格说明是否可靠。

（3）进化型：在构造系统的过程中能够适应需求的变化，通过不断地改进原型，逐步将原型进化成最终的系统。它将原型方法的思想扩展到软件开发的全过程，适用于需求经常变动的软件项目。

原型时可采取以下两种不同的策略：

废弃策略：原型主要用于反馈和评价，据此设计出完整、准确、一致、可靠的最终系统。系统构造完成后，原来的原型系统就被废弃不用。探索型和实验型原型属于这种策略。

追加策略：原型作为最终系统的核心，然后通过不断地扩充修改，逐步追加新要求，最后发展成为最终系统。它对应于进化型原型。

优点：

原型方法有助于增进软件人员和用户对系统服务需求的理解；

原型方法提供了一种有力的学习手段；

使用原型方法，可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果；

软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。

原型法的适用范围和局限性：

对于一个大型系统，如果不经过系统分析得到系统的整体划分，而直接用原型来模拟是很困难的。

对于大量运算的、逻辑性较强的程序模块，原型方法很难构造出该模块的原型来供人评价。

对于原有应用的业务流程、信息流程混乱的情况，原型构造与使用有一定的困难。

对于一个批处理系统，由于大部分活动是内部处理的，因此应用原型方法会有一定的困难。

原型方法存在的问题：

文档容易被忽略。

建立原型的许多工作会被浪费掉。

项目难以规划和管理。

原型方法在生命周期模型的应用：辅助或代替分析阶段、辅助设计阶段、代替分析与设计阶段、代替分析、设计和实现阶段、代替全部开发阶段。

1. RUP（Rational Unified Process）统一过程模型：是一种软件工程过程框架，是一个面向对象的基于web的程序开发方法论。RUP既是一种软件生命周期模型，又是一种支持面向对象软件开发的工具，它将软件开发过程要素和软件工件要素整合在统一的框架中。RUP是融合了喷泉模型和增量模型的一种综合生命周期模型。

RUP中的软件生命周期在上为四个顺序的阶段，每个阶段结束于一个主要的里程碑，并在阶段结尾执行一次评估以确定这个阶段的目标是否已经满足。



RUP的9个核心工作流：6个核心过程工作流、3个核心支持工作流

RUP的最佳实践：

⑴ 短时间分区式的迭代：2～6周，不鼓励时间推迟；

⑵ 适应性开发：小步骤、快速反馈和调整；

⑶ 在早期迭代中解决高技术风险和高业务价值的问题；

⑷ 不断地让用户参与迭代结果的评估，并及时获取反馈信息，以逐步阐明问题并引导项目进展；

⑸ 在早期迭代中建立内聚的核心架构。

⑹ 不断地验证质量；尽早、经常和实际地测试；

⑺ 使用用例驱动软件建模：用例是获取需求、制定计划、进行设计、测试、编写终端用户文档的驱动力量。

⑻ 可视化软件建模：使用UML（Unified Modeling Language，统一建模语言）进行软件建模。

⑼ 仔细地管理需求：不要草率地对待需求，而要有机地进行需求管理。

⑽ 实行变更请求和配置管理。

1. 敏捷模型：敏捷建模（Agile Modeling，AM）是从许多的软件开发过程实践中归纳总结出来的一些敏捷建模价值观、原则和实践等组成的，它只是一种态度，不是一个说明性过程。是对已有生命周期模型的补充，它本身不是一个完整的方法论，在应用传统的生命周期模型时可以借鉴AM的过程指导思想。

敏捷建模的价值观：沟通、简单、反馈、勇气、谦逊

1. 极限编程(eXtreme Programming，XP)：是敏捷模型的一种实现过程，是一种轻量级的软件开发方法，以实践为基础的软件工程过程与思想；它使用快速反馈，最大程度的保证满足用户需求；极限编程相比传统方法学更强调可适应性而非可预测性。
2. 结对编程：这是极限编程最有争议的实践。就是两个程序员合用一台计算机编程，一个编码，一个检查，增加专人审计是为了提供软件编码的质量。两个人的角色经常变换，保持开发者的工作热情。这种编程方式对培养新人或开发难度较大的软件都有非常好的效果。

需求分析 结构化需求分析

1. 系统分析和可行性分析的目的：明确系统是否值得做，避免投资损失
2. 可行性分析的四个方面：经济可行性分析、技术可行分析、法律可行性分析、方案的选择
3. 软件需求分析的对象：用户要求。
4. 软件需求分析的任务：准确地定义新系统的目标，回答系统必须“做什么”的问题并编制需求规格说明书。
5. 需求分析的目标：借助于当前（业务）系统的逻辑模型导出目标系统的逻辑模型，解决目标系统的“做什么”的问题。
6. 需求分析的原则：

需求分析方法的一组操作性原则是：

问题的信息域必须被表示和理解。

软件将完成的功能必须被定义。

软件的行为（作为外部事件的结果）必须被表示。

需求工程的指导性原则：

首先要正确地理解问题，再建立分析模型。

记录每个需求的起源及原因，保证需求的可回溯性。

给需求赋予优先级。

努力删除歧义性。

开发一个能使用户能够了解人机交互过程的原型。

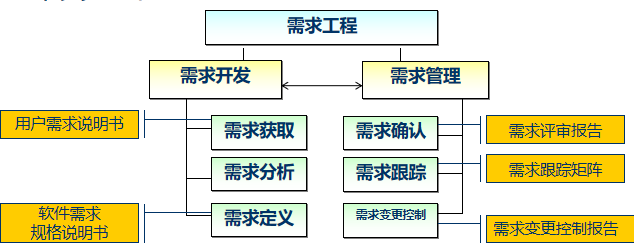
1. 需求分析的三元模型：

数据建模：信息内容和关系：表示了个体数据和控制对象及其与其他的数据和控制对象关联、信息流：表示了数据和控制在系统中流动时变化的方式、信息结构：表示了各种数据和控制项的内部组织。

功能模型：对进入软件的信息和数据进行变换和处理的模块，它必须至少完成三个常见功能：输入、处理和输出。

行为模型：大多数软件对来自外界的事件做出反应，这种刺激／反应特征形成了行为模型的基础。行为模型创建了软件状态的表示，以及导致软件状态变化的事件的表示。

1. 软件需求工程：软件的需求分析是一系列复杂的软件工程活动，为了便于对需求进行更好的管理，人们把所有与需求直接相关的活动通称为需求工程。



1. 需求分析的过程包含哪些方面：

需求沟通、需求获取、需求分析与综合、需求建模、制定需求分析规格说明、需求确认、需求评审。

1. 需求分析的分析模型必须达到三个主要目标：描述客户的需求；建立创建软件设计的基础；定义在软件完成后可以被确认的一组需求。



1. 结构化需求分析模型：数据字典是模型的核心

实体关系图（ER图）：描述数据对象间的关系，用于数据建模。

数据流图（DFD图）：描述了数据流在系统中流动的过程，以及对数据流进行变换的功能，用于功能建模。

状态迁移图（STD图）：描述了对外部事件的响应方式，表示了系统的各种行为模式（称为状态）以及在状态间进行变迁的方式，用于行为建模。

1. 数据模型:一种面向问题的数据模型，是按照用户的观点来对数据和信息建模。它描述了从用户角度看到的数据，反映了用户的现实环境，但与在软件系统中的实现方法无关。

概念模型常用的表示方法是实体-关系图法(ER图)，也称为实体关系模型。

1. 关系的规范化

好处：

消除多义性：使关系中的属性含义清楚、单一；

关系单纯化：让每个数据项只是简单的数或字符串，方便操作。使数据的插入、删除与修改操作可行且方便；

使关系模式更灵活：易于实现接近自然语言的查询方式。

范式级别越高，冗余度越小，会出现的问题：

存储同样数据就需要分解成越多张表，因此“存储自身”的过程也就越复杂。

数据的存储结构与基于问题域的结构间的匹配程度也随之下降，为此在需求发生变化时数据的稳定性随之下降。

需要访问的表增多，因此性能或处理速度将下降。

第一范式（1NF）：关系中所有属性都是“单纯域”，即每个属性值 都是不可再分的最小数据单位 。

第二范式（2NF）：非主属性完全依赖于关键字。

第三范式（3NF）：在满足第二范式的基础上，要求非主属性相互独立，即任何非主属性间不存在函数依赖。

1. 功能模型的工具：数据流图

组成：数据流--是沿箭头方向传送数据的通道，起止于外部实体；加工--对数据流进行处理的功能单元；外部实体--表示图中要处理数据的输入来源或处理结果要送往何处；数据存储--在数据流图中起保存数据的作用，它可以是数据库、文件或任何形式的数据组织。指向文件的数据流可理解为写入文件或查询文件，从文件中引出的数据流可理解为从文件读取数据或得到查询结果。

1. 行为建模工具：状态迁移图可以用来描述系统或对象的状态，以及导致系统或对象的状态改变的事件，从而描述系统的行为。
2. Petri网：简称PNG（Petri Net Graph）主要用于处理并发系统当中的同步问题、资源竞争问题以及死锁问题等。
3. 数据词典（DD，Data Dictionary）：对于数据流图中出现的所有被命名的图形元素加以定义，使得每一个图形元素的名字都有一个确切的解释。

内容组成：数据流词条、数据元素词条、数据文件词条、加工词条、外部实体词条。

1. 加工逻辑说明：

要求：对数据流图的每一个基本加工，必须有一个加工逻辑说明；

加工逻辑说明必须描述基本加工如何把输入数据流变换为输出数据流的加工规则；

加工逻辑说明必须描述实现加工的策略而不是实现加工的细节。

结构化英语：也称为PDL，是一种介于自然语言和形式化语言之间的半形式化语言。它是在自然语言基础上加了一些限制而得到的语言，是使用有限的词汇和有限的语句来描述加工逻辑。

判定表：条件桩、条件项、动作桩、动作项

判定树：也是用来表达加工逻辑的一种工具。

注：在表达一个基本加工逻辑时，结构化英语、判定表和判定树常常交叉使用，互相补充。应当以结构化英语为主，对存在判断问题的加工逻辑，可辅之以判定表和判定树。

软件设计 结构化软件设计

1. 软件设计的最基本目标:回答“概括地描述系统如何实现用户所提出来的功能和性能等方面的需求”

软件设计的最终目标:要取得最佳方案。

1. 软件设计构成：

管理角度

概要设计：将软件需求转化为数据结构和软件的系统结构，并建立接口

详细设计：即过程设计，通过对软件结构进行细化，得到各功能模块的详细数据结构和算法。

技术角度

系统结构设计：定义了软件系统各主要成份之间的关系；

数据设计：将实体关系图中描述的对象和关系，以及数据词典中描述的详细数据内容转化为数据结构的定义；

过程设计：把系统结构设计转换成软件的过程性描述。

1. 软件设计原则:

衡量设计过程的技术原则:

设计必须实现分析模型中描述的所有显式需求，必须满足用户希望的所有隐式需求。

对于开发者和维护者而言，设计必须是可读的、可理解的，使得将来易于编程、易于测试、易于维护。

设计应该给出软件的全貌，包括从实现角度可看到的数据、功能、行为。

设计模型的技术原则:

设计模型应该是一个分层结构。

设计应当模块化。

设计应当包含数据、系统结构、接口和构件（模块）的清晰的视图。

设计应当根据将要实现的对象和数据模式导出合适的数据结构。

设计应当建立具有独立功能特征的构件。

设计应当建立能够降低模块与外部环境之间复杂连接的接口。

1. 软件模块化：

模块(module)：整个软件可被划分成若干个可单独命名且可编址组成部分，这些部分称之为模块。

模块具有如下三个基本属性：功能：实现什么功能，做什么事情；逻辑：描述模块内部怎么做；状态：该模块使用时的环境和条件。

模块设计标准：模块可分解性：可将系统按问题／子问题分解的原则分解成系统的模块层次结构；

模块可组装性：可利用已有的设计构件组装成新系统，不必一切从头开始；

模块可理解性：一个模块可不参考其他模块而被理解；

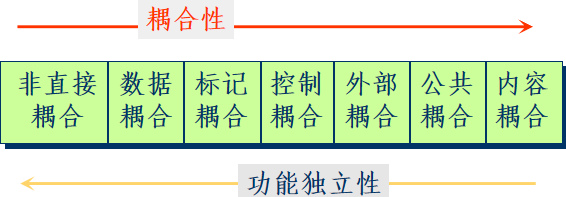
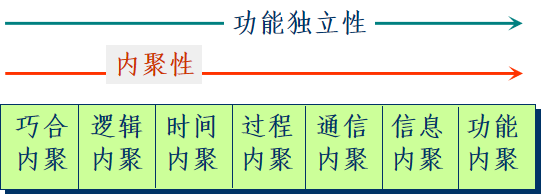
模块连续性：对软件需求的一些微小变更只导致对某个模块的修改而整个系统不用大动；

模块保护：将模块内出现异常情况的影响范围限制在模块。

1. 信息隐藏：每个模块的实现细节对于其它模块来说是隐蔽的。模块中所包含的信息（包括数据和过程）不允许其它不需要这些信息的模块使用。使得模块的修改和软件的维护所造成的影响可以局限在一个或几个模块范围内。
2. 功能独立性：抽象、模块化和信息隐藏的直接产物。如果一个模块能够独立于其他模块被编程、测试和修改，而和软件系统中其它的模块的接口是简单的，则该模块具有功能独立性。
3. 内聚性与耦合性：

内聚是模块功能强度的度量，一个模块内部各个元素之间的联系越紧密，则它的内聚性就越高，相对地，它与其他模块之间的耦合性就会减低，而模块独立性就越强。

耦合是模块之间的相对独立性（互相连接的紧密程度）的度量。模块之间的连接越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。



注：详见PPT

1. 软件设计基础（碎片知识）

程序结构的深度：程序结构的层次数称为结构的深度。

程序结构的宽度：层次结构中同一层模块的最大模块个数称为结构的宽度。

模块的扇入和扇出：扇出表示一个模块直接调用（或控制）的其它模块数目。扇入则定义为调用（或控制）一个给定模块的模块个数。

程序结构可以按水平方向或垂直方向进行划分。

水平划分：水平划分按主要的程序功能来定义模块结构的各个分支。

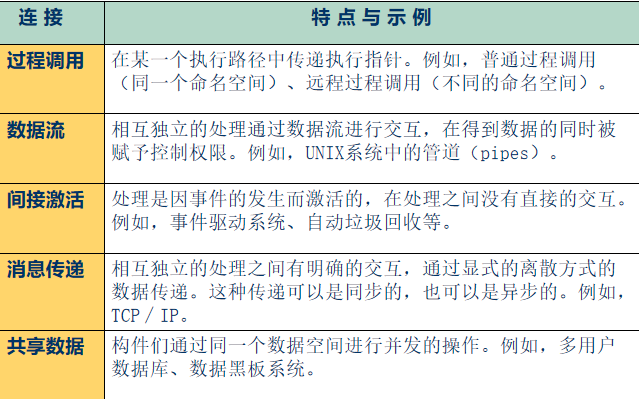
优点：由于主要的功能相互分离，易于修改、易于扩充，且没有副作用。

缺点：需要通过模块接口传递更多的数据，使程序流的整体控制复杂化。

垂直划分：也叫做因子划分，主要用在程序的体系结构中。

优点：对低层模块的修改不太可能引起副作用的传播。

1. 软件体系结构的三要素：程序构件（模块）的层次结构、构件之间交互的方式、数据的结构。



1. 软件体系结构风格:描述某一特定应用领域中系统组织方式的惯用模式。

(详见PPT)

1. 系统结构图模块分类：

传入模块 ：从下属模块取得数据，经过某些处理，再将其传送给上级模块。

传出模块 ：从上级模块获得数据，进行某些处理，再将其传送给下属模块。

变换模块 ：即加工模块。它从上级模块取得数据，进行处理，转换成其它形式，再传送回上级模块。

协调模块 ：对所有下属模块进行协调和管理的模块。

1. 系统功能结构图根据数据流图中加工特性分为以下两种结构：

变换处理型：取得数据、变换数据、给出数据

事务处理型：事务型数据处理问题的工作机理是接受一项事务，根据事务处理的特点和性质，选择分派一个适当的处理单元，然后给出结果。事务型系统结构图中由以下模块：事务处理中心：负责任务的分派；事务处理加工：完成具体的处理。

事务：可以引发一个或多个处理作业数据流。事务是最小的工作单元，不论成功与否都作为一个整体进行工作。

事务特点：

原子性：标识事务是否完全地完成。如果事务完成，则组成事务的所有任务都完成，一旦有一个任务失败，事务就失败，系统反回事务开始前的状态。

一致性：保证系统的任何事务最后都处于有效状态。

隔离性：在隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。

持久性：持久性意味着一旦事务执行成功，在系统中产生的所有变化将是永久的。

1. 映射相关：变换映射是一组设计步骤，将具有变换流特征的数据流图映射为一个预定义的程序结构模版。

事务映射也从分析数据流图开始，自顶向下，逐步分解，建立事务型系统结构图。

一个大型的软件系统通常是变换型结构和事务型结构的混合体。在具体的应用中一般以变换型为主，事务型为辅的方式进行软件结构设计。

1. 改进系统功能结构图的启发式原则：

消除重复功能，改善软件结构：

完全相似：在结构上完全相似，可能只是在数据类型上不一致。

局部相似：需要考虑功能之间的耦合及内聚性

模块的作用范围应在控制范围之内：

将判定所在模块合并到父模块中，使判定处于较高的层次；

将受到判定影响的模块下移到控制范围内；

将判定上移到层次中较高的位置

尽可能减少高扇出结构：

模块的扇出过大，将使得系统的模块结构图的宽度变大，宽度越大结构图越复杂。

扇出过小，使得系统的功能结构图的深度大大增加，不但增加了模块接口的复杂度，而且增加了调用和返回的时间开销，降低系统的工作效率。

模块的扇入太大，而且它又不是公用模块，说明该模块可能具有多个功能。为此应当对其进一步分析并将其功能分解。

避免或减少使用病态联接:

直接病态联接：即模块A直接从模块B内部取出某些数据，或者把某些数据直接送到模块B内部.。

公共数据域病态联接：模块A和模块B通过公共数据域直接传送或接受数据，而不是通过它们的上级模块。

通信模块联接：即模块A和模块B通过通信模块TABLEIT传送数据。通信未经过上级模块。

1. 界面设计黄金规则：置用户于控制之下；减少用户的记忆负担；保持界面一致。

面对对象分析与设计

1. OOA/OOD：是一种逐步进阶的面向对象建模方法，其特点是概念清晰，简单易学。

OOA使用了基本的结构化原则，结合了面向对象的观点，包括下面五个步骤：确定类与对象、标识结构、定义主题、定义属性、定义服务。

确定类与对象主要是描述如何找到类和对象。从应用系统需求出发，以整个应用为基础标识类与对象，然后按这些类与对象分析系统的职责。

标识结构按照两种不同的原则进行：第一种是按照一般化／特殊化结构，确定已标识出的类之间的继承层次关系。第二种是按照整体／部分关系，来确定一个对象怎样由其它对象组成，以及对象怎样组合成更大的复杂对象。

标识主题是通过把类与对象划分成更大的单元来完成的。主题是一组类与对象。主题的大小应合适地选择，使得人们可以从模型很好地理解系统。

标识属性是通过标识与类有关的信息和关联来完成的。对每个类，只需要标识刻画必需的属性就可以了。标识好的属性应放在合适的继承层次上。关联也要通过检查问题域上的关系标识出来。

定义服务就是定义类上的操作。主要是通过定义对象状态，以及定义诸如创建、访问、连接、计算、监控等服务来完成。

OOD负责系统设计，包括四个步骤：

设计问题域部分（细化分析结果）

问题域部分实际上是OOA工作的进一步延伸，在OOA工作基础上进行。

设计人机交互部分(设计用户界面)

突出人如何使用系统，以及系统如何向用户提交信息。

设计任务管理部分(确定系统资源的分配)

任务是进程的别名，任务管理部分用来管理任务的运行、交互等。任务管理部分可设计如下的策略：识别事件驱动任务；识别时钟驱动任务；识别优先任务和关键任务；识别协调者；定义每一个任务。

设计数据管理部分(确定持久对象的存储)

这部分的设计既包括数据存放方法的设计（采用关系型数据库还是面向对象数据库），又包括相应服务的设计（设计哪些类来实现数据的持久化服务，它们需要包含哪些属性和操作）。

1. 统一建模语言UML（Unified Modeling Language）：是在Booch方法、OOSE方法和OMT方法基础上，几经修改而成的一个面向对象分析与设计建模语言。是软件界第一个统一的建模语言。UML是一种标准的图形化建模语言，它是面向对象分析与设计的一种标准表示。
2. UML的组成：

用例视图（use case view），强调从用户的角度看到的或需要的系统功能；

逻辑视图（logical view）， 展现系统的静态或结构组成及特征；

并发视图（concurrent view），描述设计的并发和同步等特性，关注系统非功能性需求，也称过程视图（process view）；

构件视图（component view），关注软件代码的静态组织与管理；

部署视图（deployment view），描述硬件的拓扑结构以及软件和硬件的映射问题，关注系统非功能性需求（性能、可靠性等）；

用例图（Use case diagram）：描述系统的功能；

类图（Class）：描述系统的静态结构（类及其相互关系）；

对象图（Object）：描述系统在某个时刻的静态结构（对象及其相互关系）；

顺序图（Sequence）：按时间顺序描述系统元素间的交互；

协作图（Collaboration）：按照时间和空间的顺序描述系统元素间的交互和它们之间的关系；

状态图（State）：描述了系统元素的状态条件和响应；

活动图（Activity）：描述了系统元素的活动；

构件图（Component）：描述了实现系统的元素的组织；

部署图（Deployment）：描述了环境元素的配置并把实现系统的元素映射到配置上。

用例视图：用例图和活动图；逻辑视图和并发视图：类图、对象图，顺序图，协作图，状态图和活动图；构件视图：构件图；部署视图：部署图。

1. 用例图：组成：参与者(Actor)：是具有行为能力的事物，(人、计算机系统、硬件设备或组织)。用例（Use Case）：描述参与者如何使用系统来达到目标的一组成功场景和失败场景的集合。系统边界：用例图的范围。
2. 类图组成：

类名：表示问题域中的概念，含义清晰准确

属性：可见性 属性名：类型名= 初始值 {性质串}

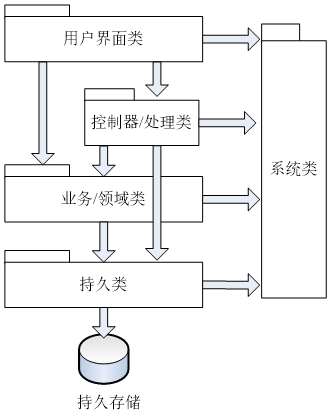
操作：可见性 操作名（参数表）：返回值类型 {性质串}

类图属性的可见性：公有的，私有的，保护的。

类之间的关系：关联-普通、反身、限定、有序、异或、关联类、聚合

继承、依赖。

1. 层次化模型：

 用户界面类实现了系统的主要用户界面元素。

业务/领域类实现与业务领域相关的概念，源于领域模型 。

控制器／处理类作为完成用例任务的责任承担者，用于协调、控制其他类共同完成用例规定的功能或行为。

持久类把永久存储、检索、更新和删除对象的能力封装起来，使底层的存储技术不暴露出来。

系统类为应用提供操作系统相关的功能，通过把特定于操作系统的特性包装起来，使软件与操作系统分离，增加应用的可移植性。

引入持久层目的：当数据存储机制或策略发生变化的时候，能减少维护工作。

1. 面向对象设计原则:

单一职责原则SRP (Single Responsibility Principle)

开闭原则OCP(Open Closed Principle)

里氏替换原则LSP(Liskov Substitution Principle)

依赖倒置原则DIP (Dependency Inversion Principle)

接口隔离原则ISP（Interface Segregation Principle）

组合/聚合复用原则(Composite/Aggregation Reuse Principle, CARP)

迪米特法则(Law of Demeter, LoD)

软件实现与测试

1. 软件实现定义：从宏观上讲，软件实现包括详细设计、程序编码、单元测试和集成测试。从微观上来讲，软件实现指程序编码和单元测试。
2. 软件实现的目标：选择某种程序设计语言，将详细设计结果进行编码实现，并形成可执行的软件系统的过程。
3. 软件实现的任务：程序设计语言的选择、集成开发环境的选择、程序实现算法的设计、程序编码实现。
4. 集成开发环境（IDE：Integrated Development Environment）：通常指运行在Windows操作系统中的图形界面软件系统，其将编辑源程序、调试程序、生成可执行文件等功能集成到一起，极大方便了程序员的编程工作。

IDE基本组成：一个编辑器、一个编译器工具链、一个调试器。

1. 软件测试：是为了发现错误而执行程序的过程。目的在于发现错误，而不是证明软件的正确。
2. 白盒测试：指根据软件产品的内部工作过程，在计算机上进行测试，以证实每种内部操作是否符合设计规格要求，所有内部成分是否已经过检查。又称为结构测试、逻辑驱动测试或基于程序的测试。

逻辑覆盖:以程序内部的逻辑结构为基础的设计测试用例的一种白盒测试技术。语句覆盖、判定覆盖、判定－条件覆盖、条件组合覆盖、路径覆盖。

基本路径测试：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。

1. 黑盒测试：在不考虑程序内部结构和内部特征的情况下，根据软件产品的功能设计规格说明，在计算机上进行测试，以证实每个实现了的功能是否符合要求。又叫做功能测试、数据驱动测试或基于规格说明的测试。

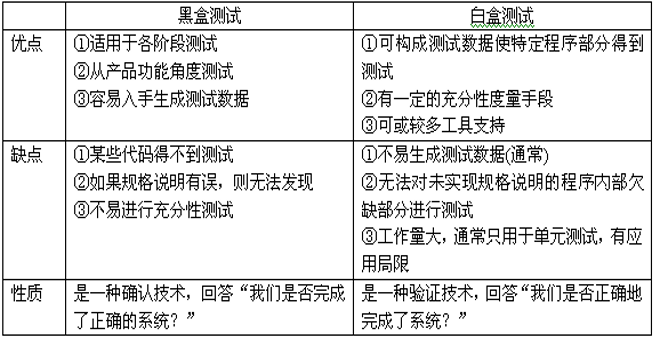
等价类划分：把所有可能的输入数据划分为若干部分，从每一部分中选取少数有代表性的数据作为测试用例。

等价类：指某个输入域的子集合，在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。

边界值分析：选取等于、刚刚大于、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。

因果图法：一种适合于描述对于多种条件的组合，相应产生多个动作的形式来考虑设计测试用例。



1. 单元测试：又称为模块测试，是针对程序模块进行正确性检验的测试。其目的在于发现各模块内部可能存在的各种差错。单元测试主要采用白盒测试为主、黑盒测试为辅的测试方法。
2. 确认测试：又称有效性测试，它的任务是验证软件的有效性，即验证软件的功能和性能及其它特性是否与用户的要求一致。

α测试：由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。这是在受控制的环境下进行的测试 。

β测试：由软件的多个用户在实际使用环境下进行的测试。β测试是在开发者无法控制的环境下进行的软件现场应用。

验收测试：以用户为主的测试。软件开发人员和QA（质量保证）人员也应参加。

1. 系统测试：将通过确认测试的软件，作为基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其它系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试。
2. 简述用例描述的组成内容：用例的目标、用例是如何被启动的、角色和用例之间的消息流、用例的执行的多个分支、用例的前置和后置条件
3. 简述单元测试、集成测试以及确认测试所对应的测试对象：

单元测试：代码和详细设计文档

集成测试：详细设计和概要设计文档

确认测试：概要设计和需求规格说明

1. 简述面向对象开发方法中**OOA**和**OOD**要完成的工作。

OOA：建立用例图、写用例文本描述、创建领域模型、绘制顺序图（可选）、创建操作契约。

OOD：软件体系结构设计、用例实现设计（用协作图寻找参与用例的对象，对对象职责进行分配，并生成设计类图）、用户界面设计。

1. 说明软件测试步骤？简述每个步骤的测试对象和测试依据。

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 测试对象 | 测试依据 |
| 单元测试 | 程序模块 | 详细设计说明书、源代码 |
| 集成测试 | 已测试的模块 | 系统概要设计说明书 |
| 确认测试 | 已集成的软件 | 系统需求规格说明书 |
| 系统测试 | 已确认的软件 | 系统其他元素 |

1. 什么是软件项目管理？制定项目计划时包含的项目管理过程有哪些？

软件项目管理就是为了实现软件项目目标，使软件项目获得成功而对软件开发项目的工作范围、可能遇到的风险、需要的资源（人力、硬件和软件）、要完成的任务、经历的里程碑、花费的工作量（成本）以及进度的安排等进行管理的过程。

制定项目计划是建立项目行动指南的基准，包括对软件项目的估算、风险分析、进度安排、人员的选择与配备等。

1. 阐述面向对象设计中依赖倒置原则的含义。
2. 面向对象分析（OOA）阶段过程中主要包括哪些工作活动？

用例建模、创建领域模型、绘制系统顺序图、创建系统操作契约。

1. ACD图(体系结构语境图)和AFD图(体系结构流程图)都可以基于一个标准的体系结构模板建立，请画出该体系结构模板的组成格式。



1. UML中类图的作用是什么？UML中类之间包含哪些关系？

UML类图用于描述类以及类之间的关系。

UML中类之间的关系可分为：关联、聚合、继承和依赖

1. 软件文档的作用：提高软件开发过程的能见度；提高开发效率；作为开发人员阶段工作成果和结束标志；记录开发过程的有关信息便于使用与维护；提供软件运行、维护和培训有关资料；便于用户了解软件功能、性能。
2. 对软件系统的需求：

功能需求：指定系统必须提供的服务；

性能需求：指定系统必须满足的定时约束或容量约束，通常包括速度、信息量速率、主存容量、磁盘容量、安全性等方面的需求；

可靠性和可用性需求：指定系统的可靠性，可用性量化了用户可以使用系统的程度；

出错处理需求：说明系统对环境错误应该怎么响应；

接口需求：描述应用系统与它的环境通信的格式，如用户接口需求、硬件接口需求、软件接口需求、通信接口需求；

约束：描述在设计或实现应用系统时应遵守的制约条件；

逆向需求：说明软件系统不应该做什么；

将来可能提出的要求：应该明确列出那些虽然不属于当前系统开发范畴，但是据分析将来很可能会提出来的要求。