软件工程复习总结

第 1 章　软件工程介绍

1. **软件的定义**　软件是包括程序、数据及其相关文档的完整集合。其中，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作信息的数据结构；文档是与程序开发、维护和使用有关的图文材料。

软件的定义：1、指令的集合，通过执行这些指令可以满足预期的特征、功能和性能需求 2、数据结构，它使得程序可以充分利用信息 3 描述程序操作和使用的文档

2. **软件的特征**

a) 软件是设计开发的，而不是传统意义上的生产制造的

b) 软件不会磨损

c) 虽然整个工业向着基于构件的构造模式发展，然而大多数软件仍是根据实际的顾客需求定制的

3. **软件与硬件的区别**

a) 软件是一种逻辑实体，而不是具体的物理实体

b) 软件的生产与硬件不同，软件开发过程中没有明显的制造过程

c) 软件在运行、使用期间没有磨损、老化问题

d) 软件的开发、运行受到计算机系统的限制，不同程度地依赖于硬件和环境，导致了软件升级和移植地问题

e) 软件复杂性越来越高

f) 软件开发成本相当昂贵

g) 大多数软件是新开发的，而不是通过已有的构件组装而来的

h) 软件工程涉及诸多的社会因素

4. **遗留软件与软件的演化**

系统演化的原因:

a) 系统需要修改其适应性，从而满足新的计算环境或者技术的需求

b) 软件必须根据新的业务需求进行升级

c) 软件必须扩展以具有与更多现代系统和数据库的协作能力

d)　软件架构必须进行改建以适应多样化的网络环境

30 年来软件发展的规律：1、持续变化规律，2、复杂性增长规律，3、自我调控规律，4、组织稳定性守恒规律，5、保证通晓性规律，6、持续增长规律，质量衰减规律，7、反馈系统规律。

5．　**软件神话：**1、管理神话。软件项目经理依赖信条，减轻提高软件进度和质量的压力。如开发宝典、增加人员、软件外包。2、用户神话。开发小组没有和用户进行有效沟通，导致没有达到用户期望。如没有详细了解就开始写程序，认为软件容易适应变更。3、从业者神话：软件开发者深信各种神话，旧的方式根深蒂固。

6. 软件新的挑战：

遍在计算。无线网络的快速发展也许将很快促成真正的分布式计算的实现。电脑、小型设备间通信

网络资源。万维网已经快速发展为一个计算引擎和内容提供平台。全球用户

开源软件。开源软件就是将系统应用程序源代码开放，

新经济。多点通信分布式

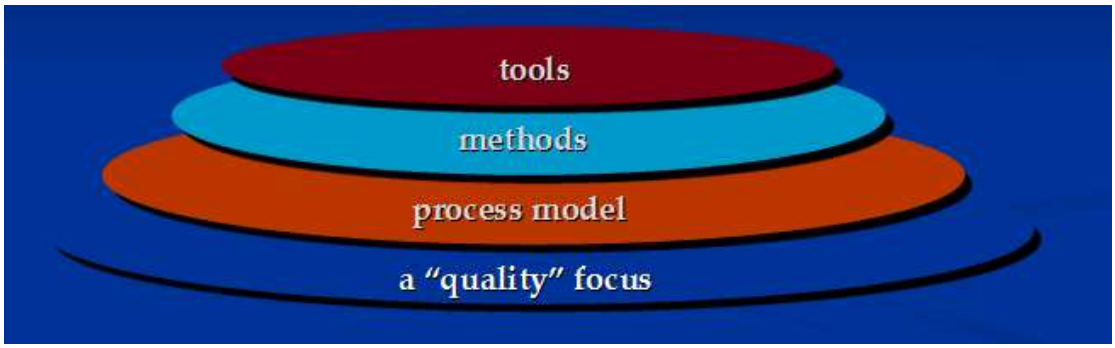
第 2 章　过程综述

1．软件工程定义：

(1) 将系统的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法用于软件开发

(2) 在(1)中所述的方法的研究

2．软件工程的层次：工具　方法　过程　质量关注点（根基）



软件工程的基础是过程（process）层。软件过程是将各个技术层次结合在一起并实施合理地、及时地开发计算机软件。过程定义一个框架，为有效交付软件过程技术，这个框架必须建立。软件过程构成了软件项目管理控制的基础，并且建立了一个环境以便于技

术方法的采用、工作产品的产生、里程碑的建立、质量的保证、正常变更的正确管理。软件工程方法（method）为建造软件提供技术上的解决方法（"如何做"）方法覆盖面很广，包括沟通、需求分析、设计建模、编程、测试和支持。软件工程方法依赖于一组基本原则，这些原则涵盖了软件工程所有技术领域，包括建模和其他描述性技术等。软件工具（tool）为过程和方法提供自动化或半自动化的支持。这些工具可以集成起来，使得一个工具产生的信息可被另外一个工具使用，这样就建立了软件开发的支撑系统，称为计算机辅助软件工程（computer-aided software engineering）

3．**通用过程框架 Generic process framework 的框架活动：**沟通 策划 建模 构建和部署

communication（沟通）这个框架活动包含了与客户（和其他共利益者）之间的大量的交流和协作，还包括需求获取以及其他相关活动。planning（策划）指为后续的软件工程工作制定计划。它描述了需要执行的技术任务，可能的风险、资源需求、工作产品和工作进度计划。modeling （建模）它包括创建模型和设计两方面。创建模型有助于客户和开发人员更好的理解软件需求；设计可以实现需求。Construction（构建）它包括编码（手写的或者自动生成的）和测试（测试是为了发现编码中的错误）deployment（部署）软件（全部或者完成的部分）交付到用户，用户对其进行评估并给出反馈意见。

4．CMMI 的概念和等级（重点）：Capability Maturity Model Integration 能力成熟度模型，SEI 提出的一个全面的过程元模型，当软件组织开发达到不同的过程能力和成熟度水平时，该模型可用来预测其所开发的系统和软件工程能力。

第 0 级：不完全级（Incomplete）。过程域没有实施，或者已经实施但未达到 CMMI 1 级成熟度所规定的所有目标。第 1 级：已执行级（Performed）CMMI 中定义的所有过程域的特定目标都已经实现。产生规定的工作产品所必需的工作任务都已经执行。第 2 级：已管理级（Managed）所有第 1 级规定的要求都已经达到。另外，所有与过程域相关的工作都符合组织的规程；工作人员都有足够的资源完成工作；共利益者都积极参与到要求的过程域；所有的工作任务和工作产品都被"监督、控制和评审；并评估是否与过程描述相一致"。第 3 极：已定义级（Defined）所有第 2 级规定的要求都已经达到。另外，根据组织剪裁准则，对其标准过程进行了裁剪，裁剪过的过程对组织的过程资产增添了新的内容，如工作产品、测量和其他过程改进信息等。第 4 级：已定量管理级（Quantitatively Managed）所有第 3 级规定的要求都已经达到。另外，通过采用测量和定量的估计等手段，对过程域进行控制和不断改进。"已经建立起来对质量和过程性能的定量指标，并作为过程管理的标准"。第 5 级：优化级（Optimized）所有第 4 级规定的要求都已经达到。另外，"采用定量（统计）的方法调整和优化过程域，以满足用户不断变更的需求，并持续地提高过程域的有效性。"

5．**PSP/TSP 模型特点**

PSP（个人软件过程）过程模型定义了 5 个框架活动：策划、高层设计、高层设计评审、开发、后验。

策划：它将需求活动分离出来，并根据需求计算项目的规模和所需资源，并且预测缺陷数目。所有的度量都用工作表或模板记录。最后，识别开发任务，并建立项目进度计划。高层设计：建立每个构件的外部规格说明，并完成构件设计。如果有不确定的需求，则构建原型系统。所有问题都被记录和跟踪。高层设计评审：使用形式化验证方法来发现设计中的错误。对所有的重要任务和工作结果都进行度量。开发：细化和评审构件级设计。完成编码，对代码进行评审，并进行编译和测试。对所有的重要任务和工作结果都进行度量。后验：根据收集到的度量和测量结果，确定过程的有效性。度量和测量结果为提高过程的有效性提供指导。

TSP 的目标（团队软件过程） • 建立自我管理团队来计划和跟踪其工作，确定目标，建立团队自己的过程和计划。 • 只是管理人员如何指导和激励其团队，并保持团队的最佳表现。•使 CMM 第 5 级的行为常规化，并依此约束员工，这样可加速软件过程改进。•为高成熟度的软件组织提供改进指导。 • 协助大学传授工业级团队技能。

第 3 章 过程模型

1．过程模型的作用：使软件开发更加有序

2．传统过程模型

瀑布模型 又被称为经典生命周期，它提出了一个系统的、顺序的软件开发方法，从用户需求规格说明开始，通过策划、建模、构建和部署过程，最终提供一个完整的软件并提供持续的技术支持。要求：需求明确 更改较小的情形

增量过程模型：

增量模型：以迭代的方式运用瀑布模型。随着时间推移，增量模型在每个阶段运用线性序列，每个线性序列生产出一个软件的可交付增量。和原型不同，增量模型每个增量都提交一个可交付的产品。瀑布模型的一个迭代版本，在每个阶段运行瀑布模型生产出一个软件可交付增量。运用增量模型时，第一个增量往往时核心产品。适用范围：在开发过程中开发人员不足

RAD 模型 ：快速应用程序开发是一种侧重于短暂的开发周期的增量软甲过程模型。RAD 是瀑布模型的高速变体，通过基于构建的方法实现快速开发。沟通来理解软件的特征，策划确保多个团队并行工作，建模包括三个阶段业务建模、数据建模和过程建模。构建运用已有的构件技术并用代码自动生成技术，部署为以后的迭代建立基础。不足：1、大量的人员，2、开发者和客户如果没有为短实践内急速完成做好准备，通常为失败，3、需要合理的模块化，否则构建建立会有很多问题，4、不适合高性能，5、高风险不宜采用 RAD。

演化过程模型：

原型模型（重点） 原型模型的基本思想是：软件开发人员在与用户进行需求分析时，以比较小的代价快速建立一个能够反映用户主要需求的原型系统，然后由客户或者用户

进行评价。开发人员根据反馈进一步对原型进行补充和完善，直到用户对开发的原型系统满意为止。使用原型系统时，客户和开发者必须承认原型是为定义需求服务的。然后丢弃原型，实际的软件系统是以质量第一为目标的。适用范围：a) 客户提出了软件的一些基本功能，但是没有详细定义的输入、处理和输出需求。b) 开发人员对算法的效率、操作系统的兼容性和人机交互的形式等情况不确定。——优点：由用户或客户进行评价，能够用来定义需求 缺点：第一个系统通常是不可用的，必须被扔掉

螺旋模型　　一种风险驱动型过程模型，它有两个显著的特点。一是采用循环的方式逐步加深系统定义和实现的深度，同时降低风险（要求在项目的所有阶段始终考虑技术风险）。二是确定一系列里程碑，确保共利益者都支持可行的和令人满意的系统解决方案。适用范围：大型系统开发

协同开发模型。有时候叫协同工程，可以表示为一系列框架活动、软件工程动作和任务以及相应的状态。协同过程模型定义了一系列事件，这些事件将出发软件工程活动、动作或状态转换。协同过程模型可用于所有类型的软件开发，能提供项目当前的状态图。

专用过程模型：

基于构建的开发(CBD)：能够做到软件复用，带来极大收益。

形式化方法模型：的主要活动是生成计算机软件的数学规格说明。使用形式化方法，歧义性问题、不完整问题、不一致问题都容易被发现和改正，不是依靠特定的评审，而是应用分析的方法。

面向方面的软件开发（AOSD）：为定义、说明、设计和构建方面提供过程和方法，是对横切关注点局部表示的一种机制，超越了子程序和继承的方法。

统一过程：

UP　　以用例为驱动、以系统架构为中心的迭代与增量过程。RUP 包括起始、细化、构建、转换和生产 5 个部分。五个 UP 阶段并不是顺序地进行，而是阶段性地并发进行。

UP 模型（概念重点）：一种用 UML 进行面向对象软件工程的框架。敏捷(Agile)的概念

4．了解模型的特点与使用范围

第 6 章 系统工程

1．系统工程的概念（分析、设计要素组成系统）

2．基于计算机系统的要素（软件、硬件、人员、数据库、文档、规程）

3．系统工程的层次 全局/领域/要素/详细视图

4．业务过程工程 需要分析和设计的三种不同架构：数据、应用、和技术基础设施

5．产品工程 需求工程、构建工程、软件工程

需求导出为何困难：范围问题 理解问题 易变问题

产品工程的目的是将用户期望的已定义的一组能力转化成真实产品。为了达到这个目的，产品工程－类似系统工程必须给出架构和基础设施。这个构架包括四个不同的系统构件：软件 硬件 数据（数据库）以及人员

软件构造包括了编码和测试循环，循环过程包括为每个构件生成源码并对其进行测试和纠错。

软件部署发生在向客户展示每个软件增量的时候。交付的关键原则是满足客户期望并且能为客户提供合适的软件信息支持。

6．系统建模方法：HP 方法（输入－处理－输出 ＋ 界面和维护自检）

7．SCD 图

8．UML 系统建模（部署图、活动图和用例图）

第 7 章 需求工程（概念）

1．**需求工程的任务：**起始、导出、精化、协商、规格说明、确认和需求管理－－Inception Elicitation Elaboration Negotiation Specification Validation Requirements Management

2．**质量功能部署（QFD）三类要求：**正常需求、期望需求、令人兴奋的需求。

3．用户场景的概念

用来识别对将要构建的系统的使用线索的描述－－用例。场景通常称为用例。本质上，用例定义了最终用户如何在以特定的环境下与系统交互。

4．UML 用例建模（用例图、活动图、状态图和类图）

系统规格说明的三个目标：功能 性能 约束

用例模版 p127

5. **需求工程概念：**需求工程帮助软件工程师更好的理解他们将要解决的问题。其中所包含的一系列任务有助于理解软件将如何影响业务、客户想要什么以及最终用户将如何与软件交互。通过需求分析可以得到的产品有：用户场景、功能和特征列表、分析模型或功能说明。

需求工程（RE）是一个软件工程动作，开始于沟通并持续到建模。需求工程在设计和构造之间建立联系的桥梁

6. 启动需求工程的过程

a. 确认共利益者

b. 识别多种观点

c. 协同合作

d. 首次提问

7. 导出需求

a. 协同需求收集

b. 质量功能部署

c. 用户场景

d. 导出工作产品

第 8 章 构建分析模型 建模的目的 对象技术 建模原则 分析包

1．分析建模的三个目标 a.描述客户需要什么，b.为软件设计奠定基础，c.定义在软件完成后可以被确认的一组需求。分析模型在系统描述和设计模型之间建立桥梁。

2．分析建模的方法（结构化分析和面向对象）1、一种考虑数据和处理的分析建模方法被称为结构分析。2、第二种方法是面向对象的分析，这种方法关注于定义类和影响客户需求的类之间的协作方式。

3．分析模型的元素：基于场景、面向信息流、基于类、基于行为

4．ERD（实体＋关系＋基数和形态）数据字典 面向对象分析模型

5．基于场景建模（用例模版、活动图/泳道图）

6．状态图

7．基于类的建模：实体、类、类图（CRC 图），行为模型（时序图）

8．基于用例图的分析类的抽象方法

9．分析模型的概念及其组成

分析包：分析建模的一个重要部分就是分类，也就是将分析模型的各种元素（如用例、分析类）分组打包－称作分析包，并为每个包取一个有代表性的名称。

10.创建分析模型遵循的原则：

a. 模型应关注在问题域或业务域内可见的需求，抽象的级别应该相对高一些

b. 分析模型的每个元素都应能增加对软件需求的整体理解，并提供对信息域。功能和系统行为的深入理解

c. 基于基础结构和其他非功能的模型应推延到设计阶段再考虑

d. 最小化整个系统内的关联

e. 确认分析模型为所有共利益者都带来价值

f. 尽可能保持模型简洁

第八章　构建分析模型

1、**分析模型。**分析模型使用文字和图表的综合形式以相对容易理解的方式描绘需求的数据、功能和行为，更重要的是，可以更直接的评审它们的正确性、完整性和一致性。

2、基于场景的建模从用户的角度表现系统，面向流的建模在说明数据对象如何通过处理函数进行转换方面提供了指示，基于类的建模定义了对象、属性和关系，行为建模描述了系统状态、类和事件在这些类上的影响。

3、分析模型必须实现的三个主要目标：a.描述客户需要什么，b.为软件设计奠定基础，c.定义在软件完成后可以被确认的一组需求。分析模型在系统描述和设计模型之间建立桥梁。

4、**分析建模的方法。**1、一种考虑数据和处理的分析建模方法被称为结构分析。2、第二种方法是面向对象的分析，这种方法关注于定义类和影响客户需求的类之间的协作方式。

5、基于场景的建模使用 UML 分析建模，从**开发用例**、**活动图**和**泳道图**形式的场景开始。

6、创建数据流模型，数据流图有助于软件工程师开发信息域的模型，并同时开发功能域的模型。

7、CRC 建模。CRC 提供了一个简单的方法，可以识别和组织与系统或产品需求相关的类。CRC 模型实际上师表示类的标准索引卡片的集合。这些卡片被分为三部分，顶部写类名，下面左侧列出类的职责，右侧部分列出类的协作关系。

8、**生成行为模型**。CRC 索引卡和其他面向对象模型表现了分析模型中的静态元素，行为模型表示系统或产品的动态行为，有状态图、顺序图。

9、**分析模型由 4 种建模元素构成：**基于场景的模型、流模型、基于类的模型和行为模型。

10、基于场景的模型从用户的角度描述软件需求。用例是主要的建模元素，还可以适用活动图说明场景，**泳道图**显示了处理流如何分配给不同的用户。

流模型关注当数据对象通过处理函数转换时的流动。

基于类的建模使用基于场景和面向流的建模元素中提取的信息确定分析类。前面三种分析模型元素提供了软件的静态视图，行为模型描述了动态行为。

行为模型使用基于场景、面向流和基于类的元素作为输入，从整体上表现分析系统和类的状态。要做到这一点，要识别状态，定义导致类做出状态转移的事件，以及确认当转移完成时所发生的动作。状态图和顺序图是用于行为建模的 UML 表达方式。

11.实体/关系图（ERD）图形化的表示对象/关系对。ERD 识别了一组基本元素：数据对象、属性、关系以及各种类型的指示符，使用 ERD 的主要目的是表示数据对象及其关系

第 9 章 设计工程

1. McGlaughlin 指导评价良好设计的 3 个特征

2. 设计的概念

抽象：抽象是人类处理复杂问题的基本方法之一。当我们在不同抽象级间移动时，我们力图创建过程抽象和数据抽象。过程抽象是指具有明确和有限功能的指令序列。数据抽象是描述对象的冠名数据集合

体系结构（概念）

软件体系结构意指"软件的整体结构和这种结构为系统提供概念上完整性的方式"。从最简单的形式来看，体系结构是程序构建（模块）的结构或组织、这些构件交互的形式以及这些构件所用数据的结构。

模式：设计模式描述了在某个特定场景与可能影响模式应用和使用方式的“影响力”中解决某个特定的设计问题的设计结构。

模块化：软件体系结构和设计模式表现为模块化；软件被划分为独立命名的、可寻址的构件，有时被称为模块，把这些构件集到一起可以满足问题的需求

信息隐藏（重点）　每个模块对其它所有模块都隐藏自己的设计决策。就是说，模块应该详细说明且精心设计以求在某个模块中包含的信息（算法和数据）不被不需要这些信息的其他模块访问。隐蔽定义并加强了模块内的过程细节和模块所使用的任何局部数据结构的访问和约束。

功能独立：功能独立的概念是模块化、抽象概念和信息隐蔽的直接结果

求精：逐步求精是一种自顶向下的设计策略，求精实际上是一个细化的过程。

重构：重构是一种重新组织的技术，可以简化构件的设计（或代码）而无需改变其功能或行为。

设计类：组织良好的设计类的 4 个特征 完整性和充分性 原始性 高内聚和低耦合性

4．模式和框架

5．完整设计的 4 个模型和作用：数据、体系结构、接口和构件级设计

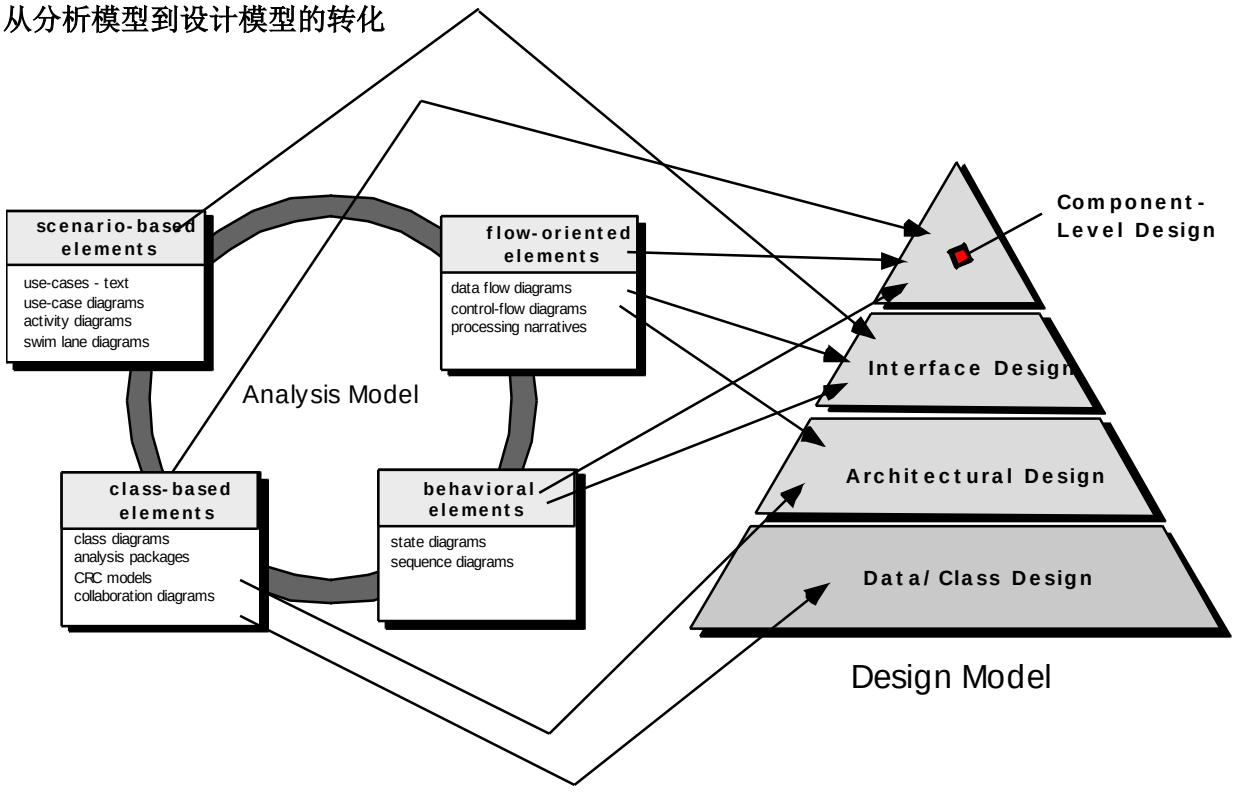
数据设计：创建在高抽象级上（以客户/用户的数据观点）表示的数据模型或信息模型

体系结构设计：揭示了规格分析模型的软件和硬件元素之间的关系和协作，体系结构设计定义了软件的主要结构元素之间的联系，为我们提供了软件的整体视图

接口设计：软件接口设计元素告诉我们信息如何流入和流出系统以及被定义为体系结构一部分的构件之间是如何通信的，描述了一组可以用来描述一个特定类的外部可见的行为的操作以及提供对这些操作的访问包括三个重要元素：用户界面（UI）；和其他系统、设备、网络或其他的信息生产者或使用者的外部接口；各种设计构件之间的内部接口。

构件级设计：描述每一个软件组件的内部细节。为所有本地数据对象定义数据结构，为所有在构件内发生的处理定义算法细节，并定义允许访问所有构件操作（行为）的接口。

6.从分析模型到设计模型的转化



第 10 章 体系结构设计

1． 体系结构设计定义和重要性

定义：一个程序和计算系统软件体系结构是指系统的一个或者多个结构。结构中包括软件的构件，构件的外部可见属性以及它们之间的相互关系。体系结构并非可运行软件，确切的说，它是一种表达，是软件工程师能够（1）分析设计在满足需求方面的有效性，（2）在设计变更相对容易的阶段，考虑体系结构可能的选择方案，（3）降低与软件构造相关联的风险。

重要性：软件体系机构的表示有助于对计算机系统开发感兴趣的各方（共利益者）开展交流；体系结构突出早期设计决策，影响随后的软件工程工作，同时对系统的最后成功有重要作用；体系结构"创建了一个相对小的，易于理解的模型，该模型描述了系统如何构成以及其构件如何一起工作"

2. **数据设计目标** 数据字典－数据字典是用来定义数据流图中的各个成分的具体含义的。它以一种准确的、无二义性的说明方式为系统的分析、设计及维护提供了有关元素的一致的定义和详细的描述。

数据设计是把分析模型定义的数据对象转化成软件构件级的数据结构，并且再必要时转化为应用程序级的数据库体系结构。

3. **体系结构风格**的组成要素 一组构件、一组连接器、约束和语意模型

一种体系风格就是一种加在整个系统设计上面的变换。它的目的就是为系统的所有的构建建立一个结构。对已有体系结构进行再工程时，强制采用一种体系结构风格会导致软件结构的根本性改变，包括对构建功能的再分配。每种风格描述一种系统类别，包括（1）一组构建完成系统需要的某种功能，（2）一组连接器，使构建间实现通信、合作和协调，（3）约束，定义构件成为一个系统，（4）语义模型，使设计者通过分析系统的构成成分的性质来理解系统的整体性质。

4. **体系结构风格分类** 以数据为中心的体系结构 数据流体系结构 调用返回体系结构 面向对象体系结构 层次体系结构

5. **模式 （并发性、持久性、分布性）**

体系结构模式（architecture pattern）软件的体系结构模式定义了处理系统某些行为特征的方法。体系结构模式域：并发性、持久性、分布性。

6. 体系结构设计（体系结构环境 ACD）

在体系结构设计层，软件架构师用体系结构环境图（architectural context diagram）对软件与外部实体交互方式进行建模。

7. 体系结构的复杂性（共享依赖 流依赖 约束依赖）

a. 共享依赖表示在使用相同资源的消费者间或为相同消费者生产的生产者之间的依赖关系

b. 流依赖表示资源的生产者和消费者之间的依赖关系

c. 约束依赖表示在一组活动间相关控制流上的约束

第 11 章 构件级设计建模

1. 构件的定义（面向对象和传统的观点）

构件是：系统中某一定型化的、可配置的和可替换的部件，该部件封装了实现并暴露一系列接口。

从面向对象观点：一个构件就是一个协作类的集合。

传统观点：一个构件就是程序的一个功能要素，程序由处理逻辑及实现处理逻辑所需的内部数据结构以及能够保证构件被调用和实现数据传递的接口构成。承担以下角色：

a. 控制构件

b. 问题域构件

c. 基础设施构件

2. 基于类的构件的基本设计原则

开关原则 替换原则 依赖倒置原则 接口分离原则 发布复用等价性原则 共同封装原则

共同复用原则

开关原则 The Open-Closed Principle (OCP)：模块应该对外延具有开放性，对修改具有封闭性

Liskov 替换原则 The Liskov Substitution Principle (LSP)：子类可以替换它们的基类

依赖倒置原则 Dependency Inversion Principle (DIP)：依赖于抽象，而非具体的实现

接口分离原则 The Interface Segregation Principle (ISP)：多个用户专用接口比一个通用接口要好

发布复用等价性原则 The Release Reuse Equivalency Principle (REP)：复用的粒度就是发布的粒度

共同封装原则 The Common Closure Principle (CCP)：一同变更的类应该合在一起

共同复用原则 The Common Reuse Principle (CRP)：不能一起复用的类不能被分到一组

3． 构件的内聚性（Cohesion）（概念）：

在为面向对象系统进行构件级设计中，内聚性意味着构建或者类只封装那些相互关联密切，以及与构件或类自身有密切关系的属性和操作

按内聚性的级别排序有：功能、分层、通信、顺序、过程、暂时、实用内聚。

4． 构件间的耦合（Coupling）

耦合是彼此联系程度的定性度量。构建级设计中，一个重要的目标就是保持低耦合。

耦合的分类：

内容耦合 共用耦合 控制耦合 印记耦合 数据耦合 例程调用耦合 类型使用耦合 包含或导入耦合 外部耦合

5． 面向对象的构件级设计步骤

步骤一：标识出所有与问题域相对应的设计类。

步骤二：确定所有与基础设施域相对应的设计类。

步骤三：细化所有不能作为复用构件的设计类。

3．1：在类或构件的协作时说明信息的细节

3．2：为每一个构件说明适当的接口。

3．3：细化属性并且定义相应的数据类型和数据结构。

3．4：详细描述每个操作中的处理流。

步骤四：说明持久数据源（数据库和文件）并确定管理数据源所需的类。

步骤五：开发并且细化类或构件的行为表示。

步骤六：细化部署图以提供额外的实现细节。

步骤七：考虑每一个构件设计表示，并且时刻考虑其它选择。

6． 传统构件级设计的图形表示

流程图 决策表 PDL 程序设计语言

第 12 章 实现用户界面设计

1. **黄金原则**

a. 置用户于控制之下

b. 减少用户的记忆负担

c. 保持界面一致

2. **用户界面分析和设计过程包括 4 个不同的框架活动：**

a. 用户、任务和环境分析及建模

b. 界面设计

c. 界面构造（实现）

d. 界面确认。

3. **界面分析步骤：**

1) 用户分析

2) 任务分析和建模

   a. 用例（Use-cases）：描述参与者和系统的交互行为方式

   b. 任务细化（Task elaboration）：进行功能分解，定义和划分任务

   c. 对象细化（Object elaboration）：定义每个类和与其相关的操作

   d. 工作流分析（Workflow analysis）：理解包含多个成员（角色）时，一个工作过程是如何完成的。用 UML 泳道图能够有效的表示工作流

3) 显示内容分析

4) 工作环境分析

4.界面设计步骤：

a. 使用界面分析中获得的信息，定义界面对象和行为（操作)

b. 定义那些导致用户界面状态发生变化的事件（用户动作）。对这个行为建模

c. 描述每一个界面的状态，就像最终用户实际看到的那样

d. 简要说明用户如何从界面提供的界面信息来解释系统状态

第 13 章 测试策略 软件测试（概念）

软件测试是为了发现设计过程中的疏忽所造成的错误的一系列活动。软件测试是一种能够系统地加以计划和说明的活动，可以进行测试用例设计，定义测试策略，根据预期的结果评估测试结果。调试出现在成功的测试之后，也就是说，当测试用例发现错误时，调试是致使错误消除的行为。

1. 测试的不可穷尽性

2. 验证和确认的概念和区别

验证（verification）是指确保软件正确地实现某一特定功能的一系列活动。

确认（validation）则指的是确保开发的软件可追溯到用户需求的另外一系列活动。

验证测试是对产品实现的功能的正确性进行测试，验证实现的功能是否正确。而确认测试是对产品是否实现了需求所定义的功能的测试，确认是否实现了功能。

独立测试组(Independent Test Group)的作用是为了避免开发人员进行测试所引发的固有问题。独立测试可以消除可能存在的认识差异。从分析与设计开始到计划和指定测试规程，ITG 参与到整个项目过程。

3. 测试的过程和策略

单元测试 集成测试 确认测试和系统测试概念

以过程的观点考虑整个测试过程，软件工程环境中的测试实际上就是按照顺序实现四个步骤。单元测试充分利用测试技术，检查构件中每个控制结构的特定路径以确保完全覆盖，并最大可能地发现错误。集成测试组装或集成各个构件以形成完整地额软件包，处理并验证与程序构造相关的问题。确认测试执行一系列的高阶测试，评估确认准则，为软件的功能、行为和性能需求提供最后的保证。系统测试验证所有的成份能够何时地结合在一起，且能满足整个系统地功能/性能需求。

4. 单元测试的策略

测试模块的接口能保证被测程序单元的信息能够正常的流入和流出；检查局部数据结构以确保临时存储的数据在算法的整个执行过程中能维护其完整性；走遍控制结构中的所有独立路径（基本路径）以确保模块中的所有语句至少被执行一次；测试边界条件以确保模块在到达边界置的极限或受限处理的情况下仍能正确地执行。最后对所有的错误处理路径进行测试。

单元测试通常被认为是编码阶段的附属工作。单元测试的设计可以在编码之前或在源代码生成之后完成。

5. 集成测试的策略

集成测试是构造软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试。其目标是利用已通过单元测试的构件建立设计中描述的程序结构。

增量集成以小增量的方式逐步进行构造和测试，这样错误易于奋力和纠正，更易于对接口进行彻底测试，而且可以运用系统化的测试方法。

自顶向下集成：从主控模块开始，沿着控制层次结构逐步向下，利用深度优先或广度优先的方式将从属于（和间接从属于）主控模块的模块集成到结构中去。（概念）

自底向上集成：从原子模块（程序结构的最底层构件）开始进行构造和测试。在处理时所需要的从属于给定层次的模块都是存在的，没有必要使用桩模块。

6. 回归测试

在集成测试策略的环境下，回归测试是重新执行已进行测试的某个子集，以确保变更没有传播不期望的副作用。

冒烟测试 build

7. 系统测试

系统测试实际上是对整个基于计算机的系统进行一系列不同考验的测试

恢复测试 ：通过各种方式强制的让系统发生故障并验证其能适当恢复的一种系统测试

安全测试：验证建立在系统内的保护机制是否能够实际保护系统不受非法入侵

压力测试：以一种要求反常数量、频率或容量的方式执行系统

性能测试：用来测试软件在集成环境中的运行性能，通常与压力测试一起进行，且常需要软件与硬件相配合。

8. 调试策略

三种调试方法：蛮力法 回溯法 原因排除法、induction

9. 确认测试

确认测试始于集成测试的结束，那时已测试完成单个构件，软件已组装成完整的软件包，且接口错误已被发现和改正。确认测试是通过一系列表明已经符合软件需求的测试而获得的。

第 14 章 测试手段

1. 软件可测试性的定义及其特性

软件可测试性就是（计算机程序）能够被测试的容易程度。

可操作性 可观察性 可控制性 可分解性 简单性 稳定性 易理解性

2. **白盒测试&黑盒测试**

白盒测试（玻璃盒测试），利用作为构件层设计的一部分而描述的控制结构来生成测试用例。作用：（1）保证一个模块中的所有独立路径至少被执行一次（2）对所有逻辑值均需测试真和假（3）在上下边界及可操作的范围内执行所有的循环（4）检验内部数据结构以确保其有效性

黑盒测试（行为测试），侧重于软件的功能需求，使软件工程师能设计出将测试程序所有功能需求的输入条件集。可以发现的错误类型：（1）功能不正确或遗漏（2）接口错误（3）数据结构或外部数据库访问错误（4）行为或性能错误（5）初始化和终止错误

3. **独立路径和环复杂度**

独立路径是贯穿程序的、至少引入一组新的处理语句或一个新的条件的路径。

环复杂度：一种软件度量，它为程序的逻辑复杂度提供一个量化的测度。当用在基本路径测试方法的环境下，环复杂性的值是用基本集合定义程序的独立路径数，它提供了保证所有语句被执行一次所需测试数量的上限。

a.域 b.V(G)=E-N+2 c.V(G)=P+1

V(G)：环复杂性，E 为流图的节点数，P 为包含在流图中的判定节点数

4. 等价划分法和边值分析法 bva（黑盒测试）

5. **控制结构的测试（传统）**：条件测试，数据流测试，循环测试

6. 基于场景的测试（00）不正确的规格说明 子系统间的交互

7. **基本路径测试**：是一种白盒测试，使测试用例设计者产生一种过程设计的逻辑复杂性测度，这种测度为执行路径的基本集的定义提供指导。

第 16 章

Web 工程采用"可靠科学的原则、工程化的原则和管理原则，以及规范、系统的手段，以期获得高质量的基于 Web 的系统和应用的成功开发、部署和维护"。

Web 属性：网络密集型 并发性 无法预计的负载量 性能 可得性 数据驱动 内容敏感性 持续演化 即时性 保密性 美学性

Web 应用类型：信息型 下载型 可定制型 交互型 用户输入型 面向事务型 面向服务型 门户型 数据库访问型 数据仓库型

1． Web 应用工程层次

过程：（1）包含变化（2）鼓励创新性、开发团队的独立性以及同 WebApp 的共利益者密切沟通（3）采用小的开发团队构造系统（4）强调使用短开发周期演化或增量开发

方法：（1）沟通方法（2）需求分析方法（3）设计方法（4）测试方法

工具及技术：。。

2． Web 工程过程框架

如果说即时性和持续演化是 webe 的重要特点，选择快速发布 webe 的敏捷过程模型；如果 webe 要开发一段很长的时间，选择增量过程模型

客户沟通－》策划－》建模－》构造－》部署

第 18 章 Web 应用分析

1． Web 应用分析模型

内容、交互、功能和配置

内容分析确定内容类和协作；交互分析描述了用户交互的基本元素、导航及最终发生的系统行为；功能分析定义了为用户提供的 webapp 功能及最终的处理顺序；配置分析确定了 webapp 所处的操作系统

交互模型－－用例 顺序图 状态图 用户界面原型

功能模型－－活动图

配置模型－－部署图

第 19 章 Web 应用设计

1． Web 应用质量

可用性 功能性 可靠性 效率 可维护性 可得性 可伸缩性 面市时间

2． 设计目标：简单性 一致性 相符性 健壮性 导航性 视觉吸引 兼容性

3． Web 界面设计模型 美学设计、内容设计、架构设计、导航设计、构件级设计、超媒体设计

补充材料（了解）：

1．数据流图 DFD 采取了系统的输入-处理-输出观点，也就是说，流入软件的数据对象，经由处理元素转换，最后以结果数据对象的形式流出软件．数据对象由带标记的箭头表

示，转换由圆圈(也称作泡泡)表示, DFD 使用分层的方式表示，即第一个数据流模型(有时也称作第 0 层 DFD 或环境图) 从整体上表现系统，随后的数据流图改进环境图.每个流图都有文字描述，细化，到最后的设计，椭圆代表处理的功能.导出数据流图时有一些简单的指导原则:

1. 第 0 层的数据流图应将软件/系统描述为一个泡泡;

2. 主要的输入和输出应被仔细地标记;

3. 通过把在下一层表示的候选处理过程，数据对象和数据存储分离，开始求精过程;

4. 应使用有意义的名称标记所有的箭头和泡泡;

5. 当从一个层转到另一个层时要保持信息流连续性;

6. 一次精化一个泡泡. 数据流图过于复杂.

2． UML 语言概述（概念）

视图是基于某一个抽象层对系统的一个抽象表示，反映系统的一个特定方面。所有视图一起共同描述一个完整的系统。

在面向对象分析的过程中，（1）基于场景模型 用例图 活动图 泳道图（2）基于类模型 类图（3）基于行为模型 状态图 时序图

UML 中主要有 5 中视图：

a) 用例视图（use case view）：用例视图表达从用户的角度看到的系统应有的外部功能。用例图是其它视图的核心和基础，它的内容将驱动其它视图的构建和发展。通常用例视图用用例图静态地描述系统功能，有时也用时序图、协作图或活动图动态地描述系统功能。

b) 逻辑视图（logic view）：逻辑视图用来描述如何实现用例视图中提出的系统功能，也就是描述系统内部的功能设计，并形成了对问题域解决方案的术语词汇。它关注的是系统的内部，既描述系统的静态结构，也描述系统的内部动态行为。静态结构描述类、对象以及它们之间的关系。动态行为描述对象之间的动态协作关系。系统的静态结构通常在类图和对象图中描述，而动态行为则在状态图、时序图、协作图和活动图中描述。

c) 并发视图（concurrent view）：并发视图用于描述系统的动态行为及其并发性。并发视图的作用是将系统划分为进程和处理器方式，并处理系统向进程和处理器的任务分配。并发视图描述的是系统的非功能属性方面，主要考虑的是资源的有效使用、代码的并发执行和异步事件的处理。并发视图用状态图、时序图、协作图、活动图、组件图和部署图来描述。

d)　组件视图（component view）：组件视图用来显示系统代码组件的组织结构方式，展示系统实现的结构和行为特征，包括实现模块和它们之间的依赖关系。组件视图由组件图组成，组件图通过一定的结构和依赖关系表示系统中的各种组件。组件就是代码模块，不同类型的代码模块构成不同的组件。

e)　部署视图（deployment view）：部署视图显示系统的实现环境和组件被部署到物理结构中的映射。部署视图用部署图来表示。

9 中不同类型的图：

A．静态图　包括用例图、类图、对象图、组件图和部署图。

用例图从系统的外部描述系统所提供的功能。

类图描述系统的静态结构。

对象图描述系统在某个时刻的静态结构。

组件图描述实现系统的元素的组织。

部署图描述系统环境元素的配置。

B．动态图　包括状态图、时序图、协作图和活动图。

状态图描述系统元素的状态和响应。

时序图按时间顺序描述系统元素间的组织。

协作图按照时间和空间的顺序描述系统元素间的交互和相互关系。

活动图描述系统元素的活动。

3．　面向对象的三大特点：

封装（用来实现信息隐藏）　把对象的属性和服务结合成一个独立的系统单位，并尽可能隐藏对象的内部细节。

继承

多态（隐藏在一个接口下的多个实现）　在父类中定义的属性或服务被子类继承后，可以具有不同的数据类型或表现出不同的行为。

耦合和内聚：

耦合（coupling）是类之间彼此联系程度的一种定性度量。内聚（cohesion）意味着构件或者类只封装那些相互关联密切，以及与构件自身有密切关系的属性和操作。