- 第 1 讲 软件与软件工程
  - 软件

    软件是计算机系统中与硬件相互依存的部分，它是包括程序、数据及相关文档的完整集合。
  - 软件工程

    软件工程是指导计算机软件开发和维护的工程学科。采用工程的概念、原理、技术和方法来开发和维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它，这就是软件工程。
  - 软件危机

    落后的软件生产方式无法满足快速增长的计算机软件需求，从而出现软件开发和维护过程中出现一系列严重问题的现象。
  - 产品生存周期

    产品生存周期是产品的市场寿命，即一种新产品从开始进入市场到被市场淘汰的整个过程。
  - 软件生存周期

    软件生存周期是软件产品从构思开始到不再使用结束时的时间周期。软件生存周期典型的阶段包括：需求阶段、设计阶段、实现阶段、测试阶段、安装与验收阶段、操作与维护阶段，有时候还包括退役阶段。
  - 软件生存周期过程

    软件生存周期过程分为基本过程、支持过程和组织过程。基本过程是软件开发从事的活动，包括获取过程、供应工程、开发过程、运作过程和维护过程 5 个过程。支持过程是软件需方和开发方各类支持人员从事的活动，包括文档编制过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审核过程、问题解决过程和易用性过程等 9 个过程。组织过程是管理人员从事的活动，包括管理过程、基础设施过程、改进过程、人力资源过程、资产管理过程、重用大纲管理过程和领域工程过程等 7 个过程。
  - 软件生存周期模型（也称软件开发模型）

    软件开发模型是软件开发全部过程、活动和任务的结构框架。
  - 瀑布模型（软件开发模型）

    瀑布模型是将软件生存周期的各项活动规定为按固定顺序而连接的若干阶段工作，形如瀑布流水，最终得到软件产品。瀑布模型核心思想是按工序将问题简化，将功能的实现与设计分开，便于分工协作，即采用结构化的分析与设计方法将逻辑实现与物理实现分开。将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护等六个基本活动，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。（来自百度百科）
  - 演化模型（软件开发模型）

    演化模型主要针对事先不能完整定义需求的软件开发，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。

- 螺旋模型（软件开发模型）

  螺旋模型是瀑布模型和演化模型相结合，并加入两者所忽略的风险分析所建立的一种软件开发模型。螺旋模型是一种演化软件过程模型，它将原型实现的迭代特征与线性顺序模型中控偶之的和系统化的方面结合起来，使软件的增量版本的快速开发成为可能。螺旋模型沿着螺线进行若干次迭代，每次迭代都包括制定计划、风险分析、实施工程和客户评估 4 个方面的工作。

- 快速原型模型（软件开发模型）

  快速原型模型又称原型模型，它是增量模型的另一种形式；它是在开发真实系统之前，构造一个原型，在该原型的基础上，逐渐完成整个系统的开发工作。快速原型模型的第一步是建造一个快速原型，实现客户或未来的用户与系统的交互，用户或客户对原型进行评价，进一步细化待开发软件的需求。通过逐步调整原型使其满足客户的要求，开发人员可以确定客户的真正需求是什么；第二步则在第一步的基础上开发客户满意的软件产品。（来自百度百科）

- 增量模型（软件开发模型）

  增量模型又称为渐增模型，也称为有计划的产品改进模型，它从一组给定的需求开始，通过构造一系列可执行中间版本来实施开发活动。第一个版本纳入一部分需求，下一个版本纳入更多的需求，依此类推，直到系统完成。每个中间版本都要执行必需的过程、活动和任务。如同原型进化模型一样，增量模型逐步地向用户交付软件产品，但不同于原型进化模型的是，增量模型在开发过程中所交付的不是完整的新版软件，而只是新增加的构件。（来自百度百科）

- 统一软件过程（RUP，Rational Unified Process）（软件开发模型）

  RUP 是 Rational 公司开发和维护的过程产品，是由 Objectory 过程演化而来。RUP 将项目管理、业务建模、分析与设计等统一起来，贯穿整个开发过程。RUP 过程为软件开发提供了规范性的指南，模板和范例，可用来开发所有类型的应用。

- 第 2 讲 软件项目管理
  - 项目

    项目就是在既定的资源和要求的限制下，为实现某种目标而相互联系的一次性的工作任务。PMI 对项目的定义为：项目是为创造独特的产品、服务或成果而进行的临时性工作。

  - 项目管理

    所谓管理，就是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。

  - 项目整体管理

    项目整体管理知识领域包括识别、确定、结合、统一和协调各项目管理过程组内不同过程与项目管理活动所需进行的各种过程和活动。从项目管理的角度来看，整体管理兼有统一、合并、结合各方面特征，包括为完成项目和满足顾客与其他利害关系者的要求，管理他们的期望而必须采取的贯穿项目整体的至关重要的行动。

  - 项目范围管理

项目范围管理是对项目应该包括什么和不应该包括什么进行定义和控制。它包括用以保证项目能按要求的范围完成所要涉及的所有过程，包括：确定项目的需求、定义规划项目的范围、范围管理的实施、范围的变更控制管理和范围核实。

- 软件项目管理

  软件项目管理就是为了使软件项目能够按照预定的成本、进度、质量顺利完成，而对人员、产品、过程和项目进行分析和管理的活动。

- 工作分解结构（WBS）（项目范围管理）

  工作分解结构（Work Breakdown Structure，简称 WBS），以可交付成果为导向对项目要素进行的分组，它归纳和定义了项目的整个工作范围，每下降一层，代表对项目工作的更详细的定义。

- COCOMO 模型（项目成本管理）

  COCOMO（COnstructive COst MOde，构成性成本模型）是一种结构型估算模型，是一种精确、易于使用估算方法。该模型按其详细程度分为基本 COCOMO 模型、中级 COCOMO 模型和详细 COCOMO 模型。

- 度量

  根据一定的规则，将数字或符号赋予系统、构件、过程等实体的特定属性，从而能清楚地理解该实体及其属性。简而言之，度量就是对事物属性的量化。

- 项目风险（项目风险管理）

  项目风险是一种不确定的事件或条件，一旦发生，会对项目目标产生某种正面或者负面的影响。

- RMMM 计划（项目风险管理）

  风险管理计划（RMMM，Risk Mitigation，Monitoring and Management Plan），是将所有的风险分析工作文档化，并作为整个项目中的一部分来使用。

- 软件质量（项目质量管理）

  软件质量是与软件产品满足明确或隐含需求的能力有关的特征和特性的总和。

- 软件可靠性（软件质量）

  软件可靠性是系统在规定的时间内及规定的环境条件下，完成规定功能的能力，也就是系统无故障运行的概率。

- 配置管理（软件配置管理）

  配置管理是标识和确定系统中配置项的过程，在系统整个生存期内控制这些配置项的投放和变动，记录并报告配置的状态和变动要求，验证配置项的完整性和正确性。

- 配置项（软件配置管理）

  软件开发中的文档和软件在其开发、运行和维护的过程中会得到许多阶段性的成果，在开发和运行过程中还需要用到多种工具软件和配置。这些信息项是配置管理的对象，称为配置项。

- 基线（软件配置管理）

  基线是项目生存期各开发阶段末尾的特定点，也称为里程碑，在这些特定点上，阶段工作已结束，

并且已经形成了正式的阶段产品。

- 第 3 讲 软件过程
  - 过程

    过程是做事的流程，是规章制度。
  - 软件过程

    软件过程（process）是在软件生存周期中所实施的一系列活动的集合，且每个活动可由一些任务组成。
  - 过程模型

    所谓软件过程模型就是一种开发策略，这种策略针对软件工程的各个阶段提供了一套范形，使工程的进展达到预期的目的。对一个软件的开发无论其大小，我们都需要选择一个合适的软件过程模型，这种选择基于项目和应用的性质、采用的方法、需要的控制，以及要交付的产品的特点。一个错误模型的选择，将迷失我们的开发方向。
  - ISO9000（过程模型）

    ISO9000 是一个关于质量管理体系的标准族，用于帮助组织确保满足客户和利益相关者的需求。
  - SPICE（过程模型）

    软件过程改进和能力测定（Software Process Improvement and Capability dEtermination）。
  - 6 西格玛（过程模型）

    6σ 管理的定义： 6σ 管理是通过减少波动，不断创新，达到缺陷逼近百万分之三点四的质量水平，以实现顾客满意和最大收益的系统科学。
  - 软件能力成熟度模型（CMM）（过程模型）

    它是对于软件组织在定义、实施、度量、控制和改善其软件过程的实践中各个发展阶段的描述。CMM 的核心是把软件开发视为一个过程，并根据这一原则对软件开发和维护进行过程监控和研究，以使其更加科学化、标准化、使企业能够更好地实现商业目标。
- 第 4 讲 软件开发
  - 需求工程

    需求工程是包括创建和维护软件需求文档所必需的的一切活动的过程，可分为需求开发和需求管理两大工作。
  - 非功能性需求（软件需求工程）

    非功能性需求是指系统必须具备的属性或品质，又可细分为软件质量属性（如易用性、可维护性、效率等）和其他非功能性需求。
  - 结构化分析方法（软件需求工程）

    结构化分析方法（SA）的基本思想是自顶向下，逐层分解，把一个大问题分解为若干个小问题，每个小问题再分解成若干个更小的问题。

- 数据流图（软件需求工程，结构化分析方法）

  数据流图（DFD）是 SA 方法中的重要工具，是表达系统内数据的流动并通过数据流描述系统功能的一种方法。DFD 还被认为是一个系统模型，在信息系统开发中，如果采用结构化方法，则一般将 DFD 作为需求规格说明书的一个组成部分。

- 数据字典（软件需求工程，结构化分析方法）

  数据字典是在 DFD 的基础上，对 DFD 中出现的所有命名元素都加以定义，使每个图形元素的名字都有一个确切的解释。

- 业务流程图（系统分析）

  TFD（Transaction Flow Diagram）是分析和描述现有系统的传统工具，是业务流程调查结果的图形化表示。TFD 是一种用尽可能少、尽可能简单地方法，描述业务处理过程的方法。

- 耦合（系统设计，结构化设计，模块结构）

  耦合表示模块之间联系的程度。紧密耦合表示模块之间联系非常强，松散耦合表示模块之间联系比较弱，非耦合则表示模块之间无任何联系，是完全独立的。

- 数据耦合（系统设计，结构化设计，模块结构，耦合）

  是一种模块的耦合类型，即一组模块借助参数表传递简单数据。

- 标记耦合（系统设计，结构化设计，模块结构，耦合）

  是一种模块的耦合类型，即一组模块通过参数表传递记录信息（数据结构）。

- 控制耦合（系统设计，结构化设计，模块结构，耦合）

  是一种模块的耦合类型，即模块之间传递的信息中包含用于控制模块内部逻辑的信息。

- 内聚（系统设计，结构化设计，模块结构）

  内聚表示模块内部各成分之间的联系程度，是从功能角度来度量模块内的联系，一个好的内聚模块应当恰好做目标单一的事情。

- 通信内聚（系统设计，结构化设计，模块结构，内聚）

  是一种模块的内聚类型，即所有处理元素集中在一个数据结构的区域上。

- 信息隐蔽（系统设计，结构化设计，模块结构）

  信息隐藏原则要求采用封装技术，将程序模块的实现细节（过程或数据）隐藏起来，对于不需要这些信息的其他模块来说是不能访问的，使模块接口尽量简单。

- 结构化程序设计（系统实现）

  结构化程序设计（SP）是一种程序设计方法，它采用自顶向下、逐步求精的设计方法和单入口、单出口的控制结构。

- 第 5 讲 面向对象方法
  - 对象

    对象是指一组属性以及这组属性上的专用操作的封装。属性可以是一些数据，也可以是另一个对象。

- 对象类

  将具有相同结构、操作，并遵守相同约束规则的对象聚合为一组，这组对象集合就称为对象类，简称类。

- 继承

  继承是在某个类的层次关联中不同的类共享属性和方法的一种机制。父类与子类的关系是一般与特殊的关系，一个父类可以有多个子类，这些子类都是父类的特例。

- 封装

  面向对象系统中的封装单位是对象，对象之间只能通过接口进行信息交互，对象外部不能对对象中的数据随意地进行访问。封装的目的是使对象的定义和实现分离。

- 多态

  多态是指同一个操作作用于不同的对象时可以有不同的解释，并产生不同的执行结果。

- 消息

  对象之间的进行通信的结构叫做消息，它包含接收对象去执行某种操作的信息。消息是对象与对象之间相互联系和通信的唯一途径。

- 领域分析

  领域分析指对特定问题空间内的知识进行提取、整理和建模，它为以后涉及该问题空间的系统开发提供复用基础。领域分析的目标是为领域产品族中相似应用系统建立核心资产的过程，用于应用过程的复用。

- 统一建模语言（UML）

  它是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持，包括由需求分析到规格，到构造和配置。是一种面向对象的建模语言，它是运用统一的、标准化的标记和定义实现对软件系统进行面向对象的描述和建模。

- 用例图（UML 图）

  描述角色以及角色与用例之间的连接关系。说明是谁要使用系统，以及他们使用该系统可以做些什么。一个用例图包含了多个模型元素，如系统、参与者和用例，并且显示了这些元素之间的各种关系，如泛化、关联和依赖。

- 类图（UML 图）

  类图是描述系统中的类，以及各个类之间的关系的静态视图。能够让我们在正确编写代码以前对系统有一个全面的认识。类图是一种模型类型，确切地说，是一种静态模型类型。类图表示类、接口和它们之间的协作关系。

- 对象图（UML 图）

  与类图极为相似，它是类图的实例，对象图显示类的多个对象实例，而不是实际的类。它描述的不是类之间的关系，而是对象之间的关系。

- 状态图（UML 图）

状态图通常是对类描述的补充，它说明该类的对象所有可能的状态以及哪些事件将导致的改变。

- 时序图（序列图）（UML 图）

  时序图表示几个对象之间的动态协作关系，它主要用来分析对象之间发送消息的顺序。

- 协作图（UML 图）

  与时序图一样，协作图表示对象间的动态协作关系。它除了说明消息的交互外，还显示对象及其间的关系。

- 活动图（UML 图）

  表示连续的活动流，通常用来描述一个操作所需要的活动。

- 构件图（组件图）（UML 图）

  以代码构件为单位表示代码的物理结构，以及构件之间的依赖关系。

- 部署图（配置图）（UML 图）

  表示系统中硬件和软件的物理结构，以及各部分之间的连接方式。

- 动态视图

  就是用图形化的方法表示各种状态随时间变化的状态。动态视图包括：顺序图、交互图、状态图、活动图。

- 第 6 讲 软件测试
  - 测试

    测试是一种旨在评估一个程序或系统的属性或能力，确定它是否符合其所需结果的活动。

  - 静态测试

    静态测试是指被测试程序不在机器上运行，而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。静态测试包括对文档的静态测试和对代码的静态测试。

  - 动态测试

    所谓软件的动态测试，就是通过运行软件来检验软件的动态行为和运行结果的正确性。目前，动态测试也是公司的测试工作的主要方式。

  - 白盒测试

    白盒测试又称为结构测试，它把测试对象看做是一个透明的盒子，它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。

  - 控制流测试（白盒测试）

    控制流测试根据程序的内部逻辑结构设计测试用例，常用的技术是逻辑覆盖，即使用测试数据运行被测程序，考察对程序逻辑的覆盖程度。

  - 语句覆盖（白盒测试，控制流测试）

    语句覆盖是指选择足够多的测试用例，似的运行这些测试用例时，被测程序的每个语句至少执行一次。

- 判定覆盖（白盒测试，控制流测试）

  判定覆盖也称为分支覆盖，它是指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。

- 条件覆盖（白盒测试，控制流测试）

  设计若干用例，使程序中每个判断的每个条件的可能取值至少执行一次。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。

- 条件/判定覆盖（白盒测试，控制流测试）

  设计足够测试用例，使判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有结果至少执行一次。

- 条件组合覆盖（白盒测试，控制流测试）

  设计足够的测试用例，使每个判断的所有可能的条件取值组合至少执行一次。

- 路径覆盖（白盒测试，控制流测试）

  设计足够的测试用例，覆盖程序中所有可能的路径。

- 基本路径测试

  实际程序中路径是个很庞大的数字，所有路径覆盖是不现实的，测试中把路径数压缩在一定范围内，称为基本路径测试。

- 黑盒测试

  黑盒测试也称为功能测试，主要用于集成测试、确认测试和系统测试阶段。黑盒测试将软件看作是一个不透明的黑盒，完全不考虑程序的内部结构和处理算法，而只检查软件功能是否能按照需求规格说明书的要求正常使用，软件是否能适当地接受输入数据并产生正确的输出信息。

- 等价类划分（黑盒测试）

  在设计测试用例时，等价类划分是用得最多的一种黑盒测试方法。所谓等价类就是某个输入域的集合，对于一个等价类中的输入值来说，它们揭示程序的作用是等效的。如果一个等价类内的数据是符合要求的、合理的数据，则称这个等价类是有效等价类；如果一个等价类内的数据是不符合要求的、不合理或非法的数据，则称这个等价类为无效等价类。

- 边界值分析（黑盒测试）

  经验表明，软件在处理边界情况时最容易出错。设计一些测试用例，使软件恰好运行在边界附近，暴露出软件错误的可能性会更大一些。

- 因果图（黑盒测试）

  因果图法根据输入条件与输出结果之间的因果关系来设计测试用例，它首先检查输入条件的各种组合情况，并找出输出结果对输入条件的依赖关系，然后，为每种输出条件的组合设计测试用例。

- 单元测试（测试的类型）

  单元测试是指对软件中的最小可测试单元进行检查和验证。

- 集成测试（测试的类型）

集成测试也叫组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求组装成子系统或系统，进行集成测试。

- 系统测试（测试的类型）

  系统测试是对整个系统的测试，将硬件、软件、操作人员看作一个整体，检验它是否有不符合系统说明书的地方。

- 确认测试（测试的类型）

  确认测试又称有效性测试，是对通过集成测试的软件在模拟的环境下进行的有效性测试，目的是要表明软件是可以工作的，并且符合“软件需求说明书”中规定的全部功能和性能要求。

- 功能测试（系统测试）

  功能测试是系统测试中最基本的测试，它不管软件内部的实现逻辑，主要根据软件需求规格说明书和需求列表，验证产品的功能实现是否符合需求规格。

- 压力测试（系统测试）

  压力测试是检查系统在资源超负荷情况下的表现，特别是对系统的处理时间有什么影响。

- 配置项测试

  配置项测试的对象是软件配置项，配置项测试的目的是检验软件配置项与软件需求规格说明书的一致性。

- 测试配置

  配置测试用于测试和验证软件，在不同的软件和硬件配置中进行运行。配置测试就是测试软件是否和系统的其他与之交互的元素之间兼容，如浏览器、操作系统、硬件等，验证被测软件在不同的软件和硬件配置中的运行情况。

- 第 7 讲 敏捷方法

  - 极限编程（XP）

    极限编程是一个轻量级的、灵巧的软件开发方法；同时它也是一个非常严谨和周密的方法。它的基础和价值观是交流、朴素、反馈和勇气；即任何一个软件项目都可以从四个方面入手进行改善：加强交流、从简单做起、寻求反馈、勇于实事求是。XP 是一种近螺旋式的开发方法，它将复杂的开发过程分解为一个个相对比较简单的小周期；通过积极的交流、反馈以及其它一系列的方法，开发人员和客户可以非常清楚开发进度、变化、待解决的问题和潜在的困难等，并根据实际情况及时地调整开发过程。

- 第 8 讲 软件进化

  - 软件再工程

    软件再工程关心的是通过对遗留系统改造使他们的可维护性得到提提高。一般软件功能不改变，系统的体系结构保持不变。软件再工程包括：对系统重新建立文档、组织并改造系统、用一种更先进的程序语言转换系统、修改或更新系统地体系结构、修改或更新系统地数据。

  - 软件维护

在软件运行/维护阶段对软件产品所进行的修改就是软件维护。

- 适应性维护

  在使用过程中，外部环境（新的软硬件配置）、数据环境（数据库、数据格式等）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成为适应性维护。

- 预防性维护

  采用先进的软件工程方法对需要维护的软件或软件中的一部分进行设计、编码和测试，预先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。

注：以上数据来自老师讲义、系统分析师教程和之前同学提供的题库答案。如有不正确或不完善之处，欢迎大家进行修改和补充。