芝士架构系统架构设计师红宝书

2024 年 11 月终稿 - 芝士架构凯恩编辑整理 v1.0.7

目录

前言

各位会员同学好，这是凯恩自己结合大量资料（官方教材讲义、历年真题、专业教材）和自己对于考试的理解，研究编制的系统架构设计师内部讲义。在这里，凯恩首先对考点优先级和重难点进行了调整。红字标出的内容非常有可能在选择题中考到，必须熟悉起来。另外每个章节开头，我都会提到此类章节的题型分布，若有出论文概率会给出论文预测。预测的论文题目我建议你自己做好练习，至少要把论点论据准备好。

使用说明

红宝书中对各个知识点进行了标注，其中标注重点的，说明在案例论文选择都会出现，属于重要知识点，标注次重点的一般只会在选择题中进行考察，并且频次不低。标注非重点的，考察概率很低。

系统架构设计师考试理论较多，联系实际更加好理解，其中红宝书第一章系统设计、第二章软件工程、第四章需求工程、第六章数据库系统、第十七章系统架构设计是最重要的内容，可以说不看必挂！次重点章节可以不看。一定要结合红宝书好好复习，有问题微信联系我。这边可以私我进行答疑，任何问题都可以！

红宝书用法是这样，不要试图一次逐字看完，先快速扫完，非重点章节，真题刷到在回过头来看，重复 3-4 回就可以上考场了。一定要配合刷题，否则效果很差。标题中标明（选择｜案例｜论文）的章节，表示在选择，案例，论文中都会进行考察，标明（选择）的章节表明只会在选择题中考察。

在线刷题

想要检测你的学习效果推荐你去使用我们的 PC 刷题网站 www.cheko.cc，或者 APP 刷题应用市场搜索芝士架构（蓝白图标的就是）。或者 http://oss.cheko.cc/app-release.apk 这里下载。

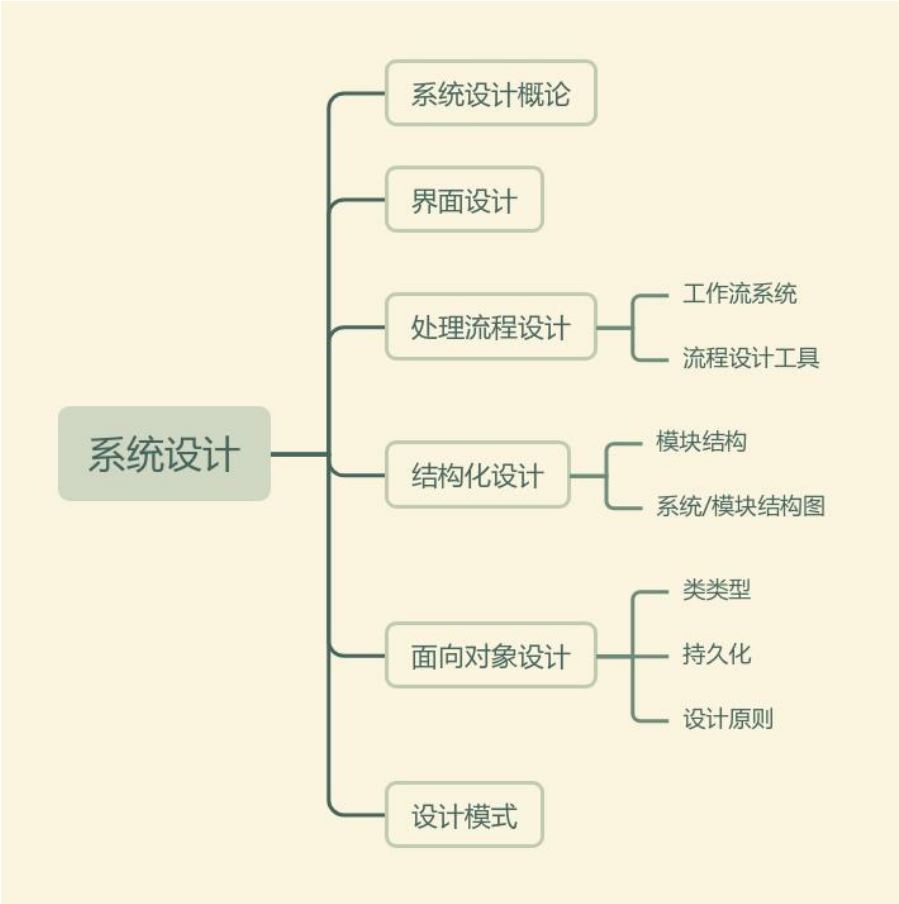
其他导学内容参考 docs.cheko.cc

1. 系统设计（案例丨论文）（重点★★★★★）

系统设计这里的内容会出现在案例和论文部分。

（1）其中案例部分多数考察面向对象的相关设计（必须非常熟悉，红字标出都要背出），结构化设计考察的相对偏少，但是一些基本的设计方法也建议了解。

（2）论文部分 2023 年已经考察过《论面向对象分析方法及其应用》，2019 年已经考察过《论软件设计方法及其应用》，2016 年已经考察过《论软件设计模式及其应用》。这一块一定要做好理论准备，否则非常容易无话可说。



1.1.主要内容（次重点★★★☆☆）

系统设计是系统分析的延伸与拓展。系统分析阶段解决“做什么”的问题，而系统设计阶段解决“怎么做”的问题。

1.1.1.概要设计（次重点★★★☆☆）

系统设计的主要内容包括概要设计和详细设计。概要设计又称为系统总体结构设计，它是系统开发过程中很关键的一步，其主要任务是将系统的功能需求分配给软件模块，确定每个模块的功能和调用关系，形成软件的模块结构图，即系统结构图。在概要设计中，将系统开发的总任务分解成许多个基本的、具体的任务。

1.1.2.详细设计（次重点★★★☆☆）

为每个具体任务选择适当的技术手段和处理方法的过程称为详细设计。根据任务的不同，详细设计又可分为多种，例如，网络设计、代码设计、输入输出设计、处理流程设计、数据存储设计、用户界面设计、安全性和可靠性设计等。

1.2.用户界面设计（次重点★★★☆☆）

通常情况下，良好的用户界面设计需要遵循如下一些基本原则：

（1）置于用户控制之下。在定义人机交互方式时，不强迫用户采用不是必需的或者不情愿的方式来进行操作，允许交互的中断和撤销。

（2）减轻用户的记忆负担。尽量减轻对用户记忆的要求，创建有意义的默认设置，定义一些符合用户直觉的访问途径，适当定义一些快捷方式，视觉布局

应该尽量与真实世界保持一致，并能够以不断扩展的方式呈现信息。

（3）保持界面一致性。用户应以一致的方式提供或获取信息，所有可视信息的组织需要按照统一的设计标准，在系列化的应用软件中需要保持一致性，用户已经很熟悉的一些界面交互模型不到万不得已时，不要随意进行修改。

以上三条原则由著名用户界面设计专家 Theo Mandel 博士所创造,通常称之为人机交互的“黄金三原则”。另外，在设计用户界面时，还需要保证界面的合理性和独特性，有效进行组合，注重美观与协调；恰到好处地提供快捷方式，注意资源协调等。

1.2.处理流程设计概述（次重点★★★☆☆）

处理流程设计的任务是设计出系统所有模块以及它们之间的相互关系,并具体设计出每个模块内部的功能和处理过程，为开发人员提供详细的技术资料。

1.2.1 流程设计概述（次重点★★★☆☆）

ISO 9000 定义业务流程（Business Process） 为一组将输入转化为输出的相互关联或相互作用的活动。流程具有目标性、内在性、整体性、动态性、层次性和结构性等特点。一般来说，流程包括 6 个基本要素，分别是输入资源、活动、活动的相互作用（结构）、输出结果、用户和价值。例如，在线教育平台系统中的“开通课程”流程，其 6 个要素。如表所示。

要素名称	要素含义
输入资源	需要开通课程的注册用户名
活动	开通课程的业务逻辑（例如，用户名合法性判断、时间判断和费用计算等）
活动的相互作用	开通课程与其他活动（例如，在线测试等）流程的相互关系
输出结果	开通课程成功后获取的短消息通知和电子邮件通知
用户	已交纳课程学习费用的注册用户
价值	用户可通过该流程实现学习课程的功能

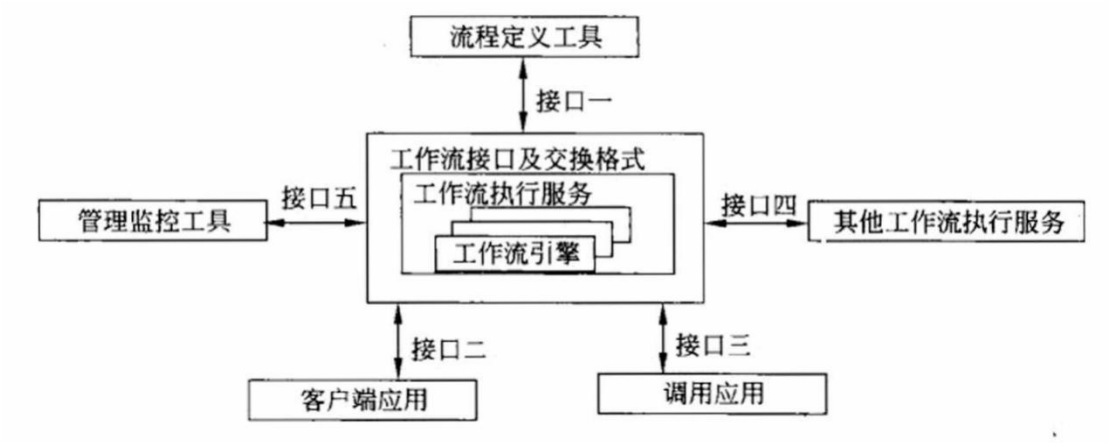
1.2.2.工作流管理系统（次重点★★★☆☆）

工作流管理系统（WFMS）通过软件定义、创建工作流并管理其执行。它运行在-个或多个工作流引擎上，这些引擎解释对过程的定义与工作流的参与者相互作用，并根据需要调用其他 IT 工具或应用。

WFMS 基本功能包括对工作流建模（定义活动和规则，对应现实业务处理过程）、工作流执行（创建和执行实际工作流）、业务过程管理和分析（监控管理执行中的业务情况）。

工作流参考模型（WRM）包含 6 个基本模块，分别是工作流执行服务、工作流引擎、流程定义工具、客户端应用、调用应用和管理监控工具。

工作流参考模型（WRM）为工作流管理系统（WFMS）关键模块提供了功能描述，并描述了关键模块之间的交互，而且这个描述是独立于特定产品或技术的实现的。



1.2.3.流程设计工具（次重点★★★☆☆）

在处理流程设计过程中，为了清晰表达过程规则说明，常用的流程设计工具分为三类：图形工具、表格工具和语言工具。

大类	工具名称	描述
图形工具	程序流程图 (PFD)	用图框表示各种操作，独立于任何程序设计语言，直观且易于掌握，但符号不规范、随意转移控制的问题需要改进。
	N-S 图 (盒图)	以方框代替传统的 PFD，包括顺序型、选择型、WHILE 循环型、UNTIL 循环型和多分支选择型控制结构，具有强烈的结构化特征。
	IPO 图	描述模块的输入、输出和数据加工，结构清晰，常用于系统设计文档。
	问题分析图 (PAD)	由日立公司提出，支持结构化程序设计，包含五种基本控制结构，执行顺序明确，适合嵌套和层次关系的表示。
	判定树	用树形结构表示逻辑判断，条件和处理流程直观，适合复杂条件判断。
表格工具	判定表	结构清晰、简明，用表格形式表示逻辑判断问题，条件和行动一目了然，适合多个互相联系的条件和多种结果的描述。
语言工具	过程设计语言 (PDL)	也称伪代码，混合自然语言词汇和结构化程序设计语言语

		法，用于描述处理过程，灵活且有助于逐步求精和详细设计。

1.3.面向对象设计（重点★★★★★）

面向对象设计（面向对象设计）是面向对象分析（OOA） 方法的延续，其基本思想包括抽象、封装和可扩展性，其中可扩展性主要通过继承和多态来实现。在面向对象设计 中，数据结构和在数据结构上定义的操作算法封装在一个对象之中。

1.3.1.设计类分类（重点★★★★★）

在系统设计过程中，类可以分为三种类型：实体类、边界类和控制类。

实体类映射需求中的每个实体，实体类保存需要存储在永久存储体中的信息，例如， 在线教育平台系统可以提取出学员类和课程类，它们都属于实体类。实体类通常都是永久性的，它们所具有的属性和关系是长期需要的，有时甚至在系统的整个生存期都需要。

实体类是对用户来说最有意义的类，通常采用业务领域术语命名，一般来说是一个名词，在用例模型向领域模型的转化中，一个参与者一般对应于实体类。通常可以从软件需求规格说明书（SRS）中的那些与数据库表（需要持久存储）对应的名词着手来找寻实体类。通常情况下，实体类一定有属性，但不一定有操作。

控制类是用于控制用例工作的类，一般是由动宾结构的短语（“动词+名词”或“名 词+动词”）转化来的名词，例如，用例 “身份验证”可以对应于一个控制

类“身份验证器”，它提供了与身份验证相关的所有操作。控制类用于对一个或几个用例所特有的控制行为进行建模，控制对象（控制类的实例）通常控制其他对象，因此它们的行为具有协调性。

控制类将用例的特有行为进行封装，控制对象的行为与特定用例的实现密切相关，系统执行用例的时候，就产生了一个控制对象，控制对象经常在其对应的用例执行完毕后消亡。通常情况下，控制类没有属性，但一定有方法。

边界类用于封装在用例内、外流动的信息或数据流。边界类位于系统与外界的交界处，包括所有窗体、报表、打印机和扫描仪等硬件的接口，以及与其他系统的接口。要寻找和定义边界类，可以检查用例模型，每个参与者和用例交互至少要有一个边界类，边界类使参与者能与系统交互。边界类是一种用于对系统外部环境与其内部运作之间的交互进行建模的类。常见的边界类有窗口、通信协议、打印机接口、传感器和终端等。实际上，在系统设计时，产生的报表都可以作为边界类来处理。边界类用于系统接口与系统外部进行交互，边界对象将系统与其外部环境的变更（例如，与其他系统的接口的变更、用户需求的变更等）分隔开，使这些变更不会对系统的其他部分造成影响。通常情况下，边界类可以既有属性也有方法。

1.3.2.面向对象设计原则（重点★★★★★）

在面向对象设计中，可维护性的复用是以设计原则为基础的。常用的面向对象设计原则包括单一职责原则（S）、开闭原则、里氏替换原则、依赖倒置原则、组合/聚合复用原则、接口隔离原则和最少知识原则等。

设计原则	详细描述

单一职责原则（S）	单一职责原则要求类应该只有一个设计目的
开闭原则（O）	可扩展系统并提供新行为，通过添加抽象层实现，是其他原则的基础。
里氏替换原则（L）	软件实体使用基类对象时适用于子类对象，反之不一定，设计时应将变化类设计为抽象类或接口。
接口隔离原则（I）	分逻辑和狭义两种理解，前者将接口视为角色，后者要求接口提供客户端需要的行为，将大接口方法分至小接口。
依赖倒置原则（D）	抽象不依赖于细节，针对接口编程，是实现开闭原则的主要机制，与各种技术和框架相关。
组合/聚合复用原则	通过组合或聚合关系复用已有对象，比继承更灵活，耦合度低。
最少知识原则（迪米特法则）	软件实体应尽量少与其他实体相互作用，分狭义和广义原则，狭义降低类耦合但可能影响通信效率，广义主要控制信息相关方面，利于子系统解耦和复用。

单一职责原则（重点★★★★★）

单一职责原则指一个类应该仅有一个引起它变化的原因。比如有一个类负责用户信息的处理，包括用户数据的存储、用户权限的管理等。这就违反了单一职责原则，应该将存储功能和权限管理功能分别提取到不同的类中。比如专门有一个用户数据存储类负责数据存储相关操作，有一个用户权限管理类负责权限相关的操作，这样每个类的职责就更单一明确。

开闭原则（重点★★★★★）

开闭原则是指软件实体应对扩展开放，而对修改关闭，即尽量在不修改原有

代码的情况下进行扩展。此处的“实体”可以指一个软件模块、一个由多个类组成的局部结构或一个独立的类。在 面向对象设计 中，开闭原则一般通过在原有模块中添加抽象层 （例如接口或抽象类）来实现，它也是其他面向对象设计 原则的基础，而其他原则是实现开闭原则的具体措施。

里氏替换原则（重点★★★★★）

里氏替换原则由 Barbara Liskov 提出，其基本思想是，一个软件实体如果使用的是一个基类对象，那么一定适用于其子类对象，而且觉察不出基类对象和子类对象的区别，即把基类都替换成它的子类，程序的行为没有变化。反过来则不一定成立，如果一个软件实体使用的是一个子类对象，那么它不一定适用于基类对象。

依赖倒置原则（重点★★★★★）

依赖倒置原则是指抽象不应该依赖于细节，细节应当依赖于抽象。换言之，要针对接口编程，而不是针对实现编程。在程序代码中传递参数时或在组合（或聚合）关系中，尽量引用层次高的抽象层类，即使用接口和抽象类进行变量类型声明、参数类型声明和方法返回类型声明，以及数据类型的转换等，而不要用具体类来做这些事情。为了确保该原则的应用，一个具体类应当只实现接口和抽象类中声明过的方法，而不要给出多余的方法，否则，将无法调用到在子类中增加的新方法。

实现开闭原则的关键是抽象化，并且从抽象化导出具体化实现，如果说开闭原则是面向对象设计 的目标的话，那么依赖倒置原则就是面向对象设计的主要

机制。有了抽象层，可以使得系统具有很好的灵活性，在程序中尽量使用抽象层进行编程，而将具体类写在配置文件中， 这样，如果系统行为发生变化，则只需要扩展抽象层，并修改配置文件，而无需修改原有系统的源代码，在不修改的情况下来扩展系统功能，满足开闭原则的要求。

组合/聚合复用原则（重点★★★★★）

组合/聚合复用原则又称为合成复用原则，是在一个新的对象中通过组合关系或聚合关系来使用一些已有的对象，使之成为新对象的一部分，新对象通过委派调用已有对象的方法达到复用其已有功能的目的。简单地说，就是要尽量使用组合/聚合关系，少用继承。

通过继承来进行复用的主要问题在于继承复用会破坏系统的封装性，因为继承会将基类的实现细节暴露给子类，由于基类的内部细节通常对子类来说是透明的，所以这种复用是透明的复用，又称为白盒复用。

由于组合或聚合关系可以将已有的对象(也可称为成员对象)纳入新对象中，使之成为新对象的一部分，新对象可以调用已有对象的功能，这样做可以使得成员对象的内部实现细节对于新对象是不可见的，因此，这种复用又称为黑盒复用。

接口隔离原则（重点★★★★★）

接口隔离原则是指使用多个专门的接口 ， 而不使用单一的总接口。每个接口应该承担一种相对独立的角色，不多不少，不干不该干的事，该干的事都要干。

接口隔离原则表达的意思则是指接口仅仅提供客户端需要的行为，客户端不需要的行为则隐藏起来，应当为客户端提供尽可能小的单独的接口，而不要提供

大的总接口。在面向对象编程语言中，如果需要实现一个接口，就需要实现该接口中定义的所有方法，因此，大的总接口使用起来不一定很方便，为了使接口的职责单一，需要将大接口中的方法根据其职责不同，分别放在不同的小接口中，以确保每个接口使用起来都较为方便，并都承担单一角色。

最少知识原则（重点★★★★★）

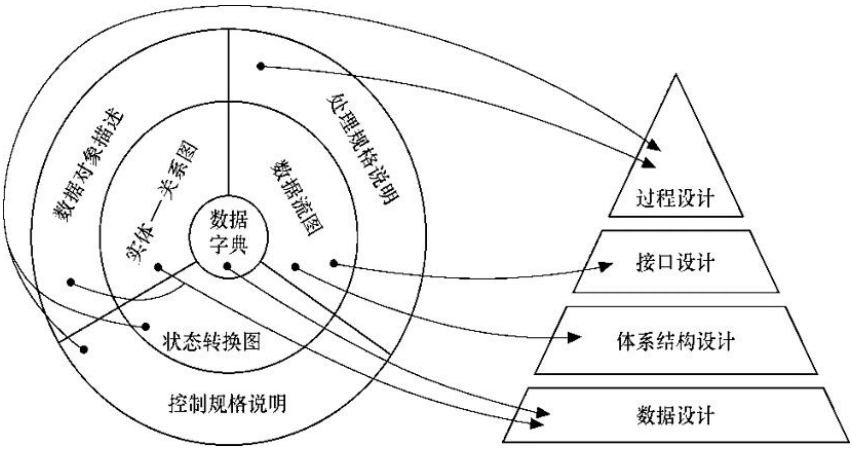
最少知识原则也称为迪米特法则，是指一个软件实体应当尽可能少地与其他实体发生相互作用。这样，当一个模块修改时，就会尽量少地影响其他的模块，扩展会相对容易。这是对软件实体之间通信的限制，它要求限制软件实体之间通信的宽度和深度。

1.4.结构化设计（重点★★★★★）

结构化设计（Structured Design ， 以下简称 SD） 是一种面向数据流的方法，它以 SRS 代表的软件需求规格说明（Software Requirements Specification），SA 代表的系统分析（System Analysis）阶段所产生的数据流图和数据字典等文档为基础，是一个自顶向下、逐步求精和模块化的过程。SD 方法的基本思想是将软件设计成由相对独立且具有单一功能的模块组成的结构，分为概要设计和详细设计两个阶段，其中概要设计的主要任务是确定软件系统的结构，对系统进行模块划分，确定每个模块的功能、接口和模块之间的调用关系。详细设计的主要任务是为每个模块设计实现的细节。

软件设计模型/结构化设计——包括四个既独立又相互联系的活动——数据设计、 软件结构设计、人机界面设计（接口设计）和过程设计。

高质量的数据设计将改善程序结构和模块划分，降低过程复杂性；软件结构设计的主要目标是开发一个模块化的程序结构，并表示出模块间的控制关系；人机界面设计描述了软件与用户之间的交互关系。



1.4.1.模块结构（重点★★★★★）

在 SD 中表现为将系统划分为可自由组合、分解和变换的模块，模块是系统基本单位，任何处理功能都可视为模块。在 SD 方法中，系统由多个逻辑上相对独立的模块组成，在模块划分时需要遵循如下原则：

（1）模块的大小要适中。系统分解时需要考虑模块的规模，过大的模块可能导致系统分解不充分，其内部可能包括不同类型的功能，若要进一步划分，尽量使得各个模块的功能单一；过小的模块将导致系统的复杂度增加，模块之间的调用过于频繁，反而降低了模块的独立性。一般来说，一个模块的大小使其实现代码在 1-2 页纸之内，或者其实现代码行数在 50-200 行之间，这种规模的模块易于实现和维护。

（2）模块的扇入和扇出要合理。一个模块的扇出是指该模块直接调用的下级模块的个数：扇出大表示模块的复杂度高，需要控制和协调过多的下级模块。扇出过大一般是因为缺乏中间层次，应该适当增加中间层次的控制模块；扇出太

小时可以把下级模块进一步分解成若干个子功能模块，或者合并到它的上级模块中去。一个模块的扇入是指直接调用该模块的上级模块的个数；扇入大表示模块的复用程度高。设计良好的软件结构通常顶层扇出比较大，中间扇出较少，底层模块则有大扇入。一般来说，系统的平均扇入和扇出系数为 3 或 4 ，不应该超过 7 ， 否则会增大出错的概率。

（3）深度和宽度适当。深度表示软件结构中模块的层数，如果层数过多，则应考虑是否有些模块设计过于简单，看能否适当合并。宽度是软件结构中同一个层次上的模块总数的最大值，一般来说，宽度越大系统越复杂，对宽度影响最大的因素是模块的产出。

（4）信息隐蔽。信息隐蔽采用封装技术隐藏程序模块实现细节，使模块接口简单，系统模块应设计成“黑盒”，只通过接口信息交互，模块间相对独立易实现、理解和维护；抽象原则抽取事物基本特性和行为，忽略细节，分层次抽象可控制开发复杂性，包括过程抽象、数据抽象和控制抽象。

1.4.1.3.耦合（重点★★★★★）

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。模块的耦合类型通常分为 7 种，根据耦合度从低到高排序如表所示。

耦合类型	描 述
非直接耦合（低）	两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的
数据耦合	一组模块借助参数表传递简单数据
标记耦合	一组模块通过参数表传递记录信息（数据结构）
控制耦合	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

外部耦合	一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息
公共耦合	多个模块都访问同一个公共数据环境，公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等
内容耦合（高）	一个模块直接访问另一个模块的内部数据；一个模块不通过正常入口转到另一个模块的内部；两个模块有一部分程序代码重叠；一个模块有多个入口

对于模块之间耦合的强度，主要依赖于一个模块对另一个模块的调用、一个模块向另一个模块传递的数据量、一个模块施加到另一个模块的控制的多少，以及模块之间接口的复杂程度。

1.4.1.4.内聚（重点★★★★★）

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一件事情。模块的内聚类型通常也可以分为 7 种，根据内聚度从高到低的排序如表所示 （特别注意功能内聚和偶然内聚）。

内聚类型	描 述
功能内聚（高）	完成一个单一功能，各个部分协同工作，缺一不可
顺序内聚	处理元素相关，而且必须顺序执行
通信内聚	所有处理元素集中在一个数据结构的区域上
过程内聚	处理元素相关，而且必须按特定的次序执行
瞬时内聚（时间内聚）	所包含的任务必须在同一时间间隔内执行
逻辑内聚	完成逻辑上相关的一组任务
偶然内聚（巧合内聚）（低）	完成一组没有关系或松散关系的任务

一般说来，系统中各模块的内聚越高，则模块间的耦合就越低，但这种关系并不是绝对的。耦合低使得模块间尽可能相对独立，从而各模块可以单独开发和维护；内聚高使得模块的可理解性和维护性极大增强。因此，在模块的分解中应尽量减少模块的耦合，力求增加模块的内聚，遵循 “高内聚、低耦合”的设计原则。

1.5.设计模式（次重点★★★☆☆）

设计模式是前人经验的总结，它使人们可以方便地复用成功的设计和架构。当人们在特定的环境下遇到特定类型的问题，采用他人已使用过的一些成功的解决方案，一方面可以降低分析、设计和实现的难度，另一方面可以使系统具有更好的可复用性和灵活性。历年真题中直接考察过的设计模式如下，17 年之后考察不多，但是做好准备以防杀回马枪，选择题，案例都有可能出现。设计模式这块展开能写一本书，凯恩也很为难，为此我把 《深入理解设计模式》这本书放在会员网盘里，想要进一步了解的同学可以看看。

大类	方法名称	描述
创建模式	工厂方法	定义一个接口用于创建对象，但由子类决定实例化哪个类。
	抽象工厂	提供一个接口以创建相关或依赖对象的家族，而无需明确指定具体类。
	生成器	将一个复杂对象的构建与其表示分离，使同样的构建过程可以创建不同的表示。
	原型	通过复制现有对象来创建新对象，而不是从头开始实例化。
	单例	确保一个类只有一个实例，并提供全局访问点
结构模式	适配器	将一个类的接口转换为客户希望的另一个接口，使原本接

		口不兼容的类可以一起工作。
	桥接	将抽象部分与它的实现部分分离，使它们都可以独立变化
	组合	将对象组合成树形结构以表示“部分-整体”的层次结构，使客户可以统一处理单个对象和组合对象
	装饰	动态地给对象添加职责，而不改变其接口
	外观	为子系统中的一组接口提供一个统一的高层接口，使子系统更容易使用。
	享元	通过共享技术来有效支持大量细粒度对象的复用。
	代理	为另一个对象提供一个代理或占位符以控制对它的访问。
行为模式	责任链	使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合。
	命令	将请求封装成对象，以便使用不同的请求、队列或日志来参数化其他对象。
	迭代器	提供一种方法顺序访问一个聚合对象中的各个元素，而不暴露其内部表示。
	中介者	用一个中介对象来封装一系列对象交互，使对象不需要显式地相互引用，降低它们的耦合度。
	备忘录	在不破坏封装的前提下，捕获并外部化对象的内部状态，以便以后恢复到该状态。
	观察者	定义对象间一种一对多的依赖关系，使得每当一个对象改变状态，所有依赖它的对象都会得到通知并自动更新。
	状态	允许对象在内部状态改变时改变其行为，对象看起来好像修改了其类。
	策略	定义一系列算法，将每一个算法封装起来，并使它们可以互换。
	模板方法	在一个方法中定义一个算法的骨架，而将一些步骤延迟到子类中，使子类可以不改变算法结构即可重定义该算法的某些步骤。
	访问者	表示一个作用于某对象结构中的各元素的操作，使你可以

		在不改变各元素类的前提下定义作用于这些元素的新操作。
	解释器	给定一个语言，定义其文法的一种表示，并定义一个解释器，用于处理该语言中的句子。

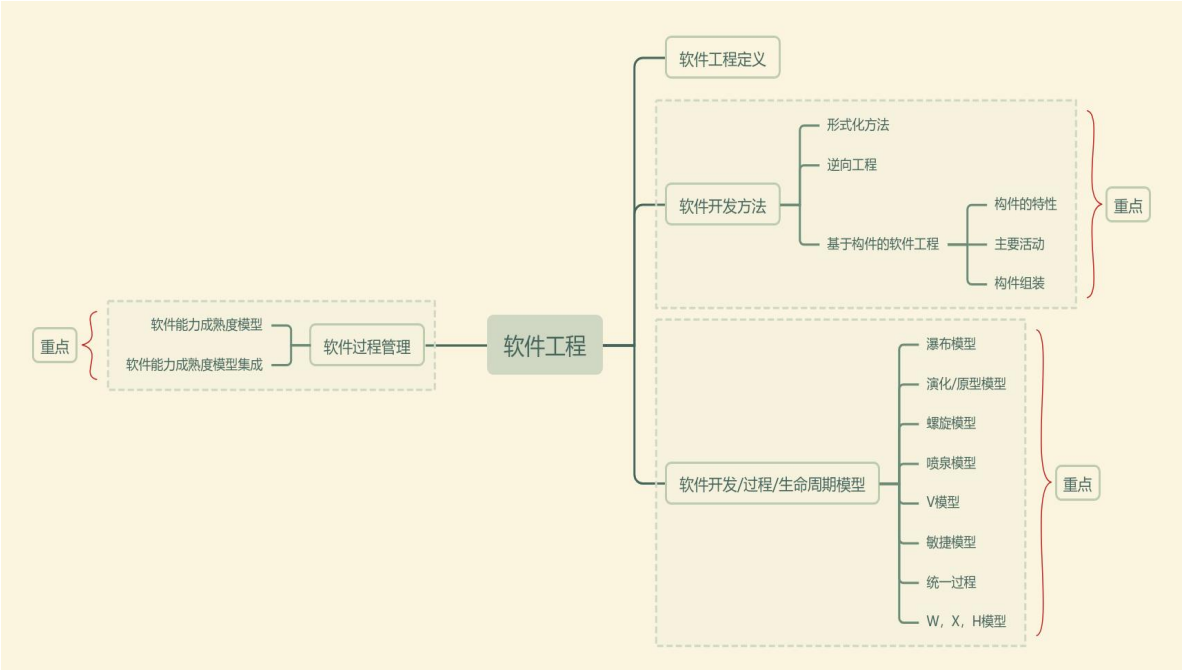
2.软件工程（选择丨案例丨论文）（重点★★★★★）

软件工程章节主要在选择题、论文题中都有考察。这一章非常枯燥，直接听我视频课，过一遍就行，通过我的思维导图去回忆，配合真题加深记忆。具体复习办法如下：

（1）对于综合题来说，本章所有红字标出的内容都要熟悉，包括生命周期，开发方法，开发模型，开发环境和工具，软件过程管理等。

（2）案例题考察的比较少，可以不关注。

（3）论文部分重点考察点就是软件开发模型，2022 年已经考察过《论基于构件的软件开发方法及其应用》，2018 年已经考察过《论软件开发过程 RUP 及其应用》，2024 年凯恩建议重点《敏捷开发》相内容，重点关注敏捷开发方法论里面的一个具体实施框架 Scrum。软件工程这一章内容非常枯燥，凯恩主要通过梳理考点和讲解的形式来辅助大家记忆。重点章节的内容你必须要好好记忆，选择题会重点考查



2.1.软件工程相关定义

软件工程由方法、工具和过程三个部分组成。软件工程过程是指为获得软件产品，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，包括以下 4 个方面 PDCA。（选择题考察）

（1）P（Plan）——软件规格说明。规定软件的功能及其运行时的限制。

（2）D（Do）——软件开发。开发出满足规格说明的软件。

（3）C（Check）——软件确认。确认开发的软件能够满足用户的需求。

（4）A（Action）——软件演进。软件在运行过程中不断改进以满足客户新的需求。

2.2.软件开发方法（次重点★★★☆☆）

软件开发方法是指软件开发过程所遵循的办法和步骤，从不同的角度可以对软件开发方法进行不同的分类。

分类方法	具体方法	描述
从开发风格上看	自顶向下方法	系统的整体结构首先被定义和设计，然后逐步细化为更具体的模块和功能
	自底向上方法	是一种从具体模块开始逐步构建整个系统的方法
从性质上看	形式化方法	形式化方法是一种具有坚实数学基础的方法
	非形式化方法	非形式化方法则不把严格性作为其主要着眼点
从适应范围来看	整体性方法	适用于软件开发全过程的方法称为整体性方法
	局部性方法	适用于开发过程某个具体阶段的软件方法称为局部性方法

2.2.1.形式化方法（次重点★★★☆☆）

1.形式化方法概述

形式化方法的主要优越性在于它能够数学地表述和研究应用问题及其软件实现。但是，它要求开发人员具备良好的数学基础。用形式化语言书写的大型应用问题的软件规格说明往往过于细节化，并且难于为用户和软件设计人员所理解。由于这些缺陷，形式化方法在目前的软件开发实践中并未得到普遍应用。

2.净室软件工程

净室软件工程（Clean room Software Engineering，CSE） 是软件开发的一种

形式化方法，可以开发较高质量的软件。它使用盒结构归约进行分析和建模，并且将正确性验证作为发现和排除错误的主要机制，使用统计测试来获取认证软件可靠性所需要的信息。 CSE 强调在规约和设计上的严格性，以及使用基于数学的正确性证明来对设计模型的每个元素进行形式化验证。作为对形式化方法中的扩展，CSE 还强调统计质量控制技术，包括基于客户对软件的预期使用的测试。CSE 的主要缺点体现在以下三个方面：

（1）对开发人员的要求比较高。CSE 要求采用增量式开发、盒结构和统计测试方法，开发人员必须经过强化训练才能掌握。

（2）正确性验证的步骤比较困难，且比较耗时。

（3）开发小组不进行传统的模块测试，这是不现实的。

2.2.2.逆向工程（次重点★★★☆☆）

软件的逆向工程是分析程序，力图在比源代码更高抽象层次上建立程序的表示过程，逆向工程是设计的恢复过程。

1.相关概念

与逆向工程相关的概念有重构、设计恢复、再工程和正向工程。

（1）重构（restructuring）。重构是指在同一抽象级别上转换系统描述形式。

（2）设计恢复（design recovery）。设计恢复是指借助工具从已有程序中抽象出有关数据设计、总体结构设计和过程设计等方面的信息。

（3）再工程（re-engineering）。再工程是指在逆向工程所获得信息的基础上，修改或重构已有的系统，产生系统的一个新版本。再工程是对现有系统的重新开发过程，包括逆向工程、新需求的考虑过程和正向工程三个步骤。它不仅能从已

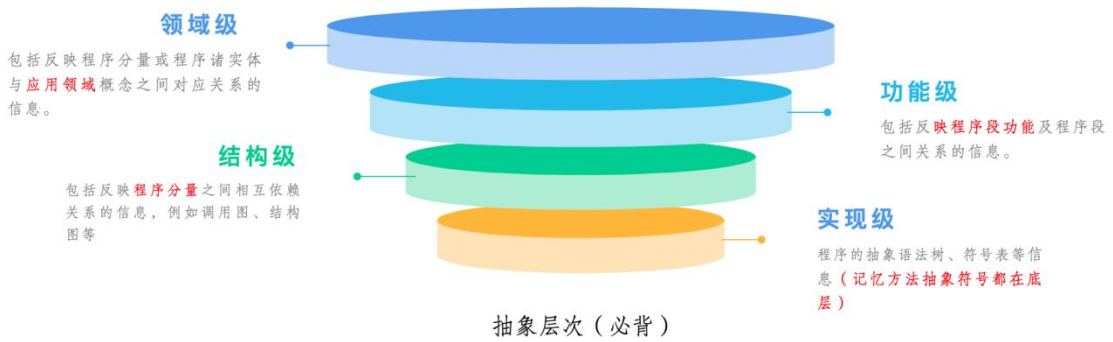
存在的程序中重新获得设计信息，而且还能使用这些信息来重构现有系统，以改进它的综合质量。在利用再工程重构现有系统的同时，一般会增加新的需求，包括增加新的功能和改善系统的性能。

（4）正向工程（Forward Engineering）。正向工程是指不仅从现有系统中恢复设计信息，而且使用该信息去改变或重构现有系统，以改善其整体质量。

2.抽象层次

逆向工程导出的信息可分为如下 4 个抽象层次。（1）实现级：包括程序的抽象语法树、符号表等信息。（2）结构级：包括反映程序分量之间相互依赖关系的信息，例如调用图、结构图等。（3）功能级：包括反映程序段功能及程序段之间关系的信息。（4）领域级：包括反映程序分量或程序诸实体与应用领域概念之间对应关系的信息。

抽象级别越高，它与代码的距离就越远，通过逆向工程恢复的难度亦越大而自动工具支持的可能性相对变小，要求人参与判断和推理的工作增多。

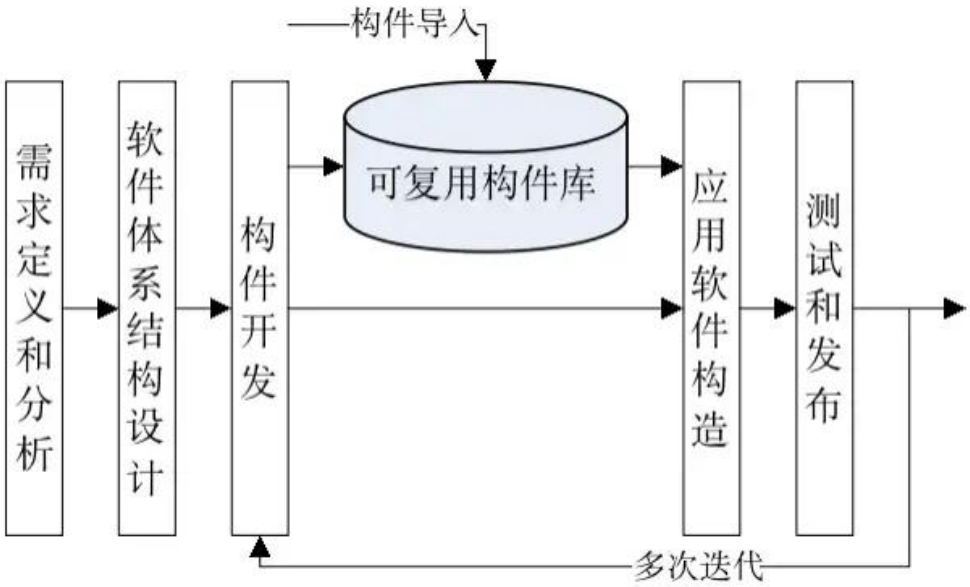


抽象层次（必背）

2.2.3.基于构件的软件工程（重点★★★★★）

构件这块 23 年架构出了至少 5 分选择题。需要重点关注。

基于构件的软件工程（Component-Based Software Engineering，CBSE）是一种新型的软件开发模式，旨在通过复用可重用的软件构件来构造高质量、高效率的应用软件系统。这种模式强调将软件系统分解为独立的构件，每个构件都具有明确定义的功能和接口，可以独立开发、测试和部署。



2.2.3.1.构件的特性（重点★★★★★）

记忆口诀——竹（组）鼠（属）档（档）毒（独）镖（标）（应付选择题）

（1）可组装性。构件的所有外部交互必须通过公开定义的接口进行，同时还必须对自身信息提供外部访问。

（2）可部署性。构件必须是自包含的，能够作为独立实体在构件平台上运行，以二进制形式部署而无需编译。

（3）文档化。构件必须完全文档化，用户可以根据文档判断构件是否满足需求。

（4）独立性。构件应该是独立的，可以在无需其他特殊构件的情况下进行组装和部署，只有在确实需要其他构件提供服务时才应声明依赖。

（5）标准化。在 CBSE 过程中使用的构件必须符合某种标准化的构件模型。

（6）没有（外部的）可见状态。（补充）

2.2.3.2.CBSE 过程的主要活动（重点★★★★★）

记忆口诀——虚（需求）竹师（识别）傅羞（修改）涩（设计）的定（定制）住（组装）了。

（1）系统需求概览（2）识别候选构件（3）根据发现的构件修改需求（4）体系结构设计（5）构件定制与适配（6）组装构件，创建系统。

2.2.3.3.构件组装（重点★★★★★）

（1）顺序组装。按顺序调用已有构件，需要保证上下游构件接口兼容。（2）层次组装。一个构件直接调用另一个构件提供的服务，需要接口匹配。（3）叠加组装。将多个构件的功能合并到一个新构件中，提供统一的接口。

在组装构件时可能会遇到接口不兼容的问题，主要包括:（1）参数不兼容。接口操作名称相同但参数类型或个数不同。（2）操作不兼容。提供接口和请求接口的操作名不同。（3）操作不完备。一个构件接口是另一个构件接口的子集。

2.3.软件开发/过程/生命周期模型（重点★★★★★）

软件开发模型给出了软件开发活动各阶段之间的关系，它是软件开发过程的概括，是软件工程的重要内容。

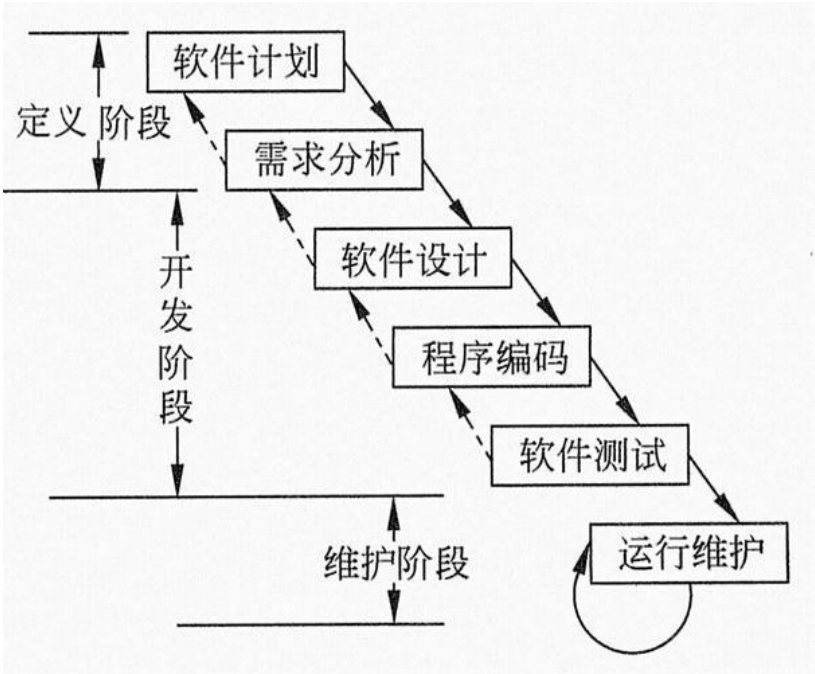
软件要经历从软件定义、软件开发、软件运行、软件维护（软件生命周期方法学），直至被淘汰这样的全过程，这个全过程称为软件的生命周期。软件生命周期描述了软件从生到死的全过程。

为了使软件生命周期中的各项任务能够有序地按照规程进行，需要一定的工作模型对各项任务给予规程约束，这样的工作模型被称为软件过程（开发）模型，有时也称之为软件生命周期模型。

软件开发模型大体上可分为三种类型。第一种是以软件需求完全确定为前提的瀑布模型；第二种是在软件开发初始阶段只能提供基本需求时采用的迭代式或渐进式开发模型，例如，喷泉模型、螺旋模型、统一开发过程和敏捷方法等；第三种是以形式化开发方法为基础的变换模型。

2.3.1.瀑布模型（重点★★★★★）

瀑布模型最早由 Winston W. Royce 在 1970 年的一篇论文中提出，它将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，形如瀑布流水，最终得到软件产品，如下图所示



瀑布模型是一个线性顺序模型，支持直线开发。其优点是强调开发的阶段性、早期计划及需求调查和产品测试，以这样严格的方式构造软件，开发人员很清楚

每一步应该做什么，有利于项目管理。

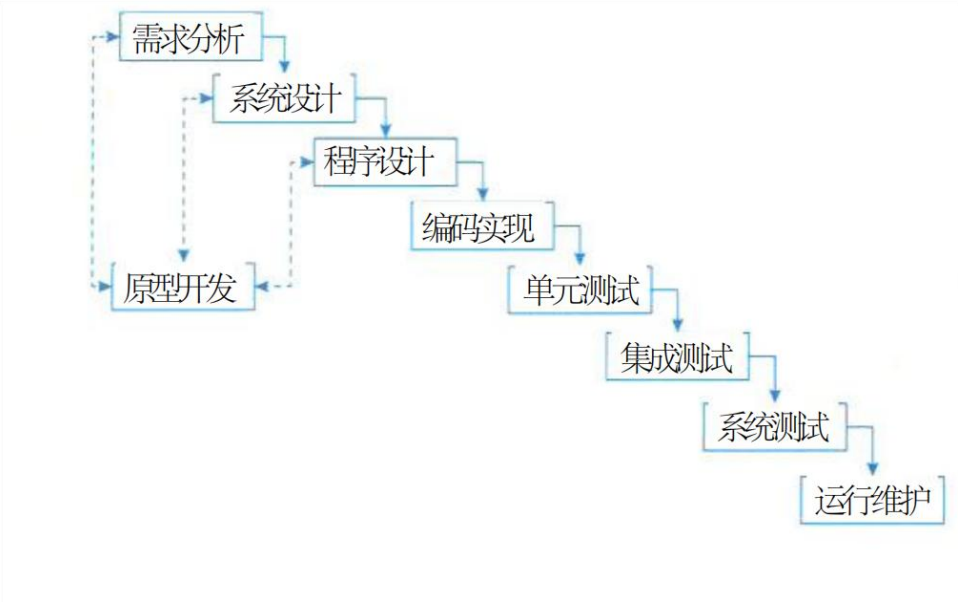
缺点（必备）：

（1）依赖于早期进行的需求调查，不能适应需求的变化；

（2）软件需求的完整性、正确性等很难确定，甚至是不可能和不现实的。

2.3.2.演化模型/原型模型（重点★★★★★）

由于瀑布型的缺点，人们提出了原型模型。该模型如下图所示。



原型模型主要有以下两个阶段。

（1）原型开发阶段。软件开发人员根据用户提出的软件系统的定义，快速地开发一个原型。

（2）目标软件开发阶段。在征求用户对原型的意见后对原型进行修改完善，确认软件系统的需求并达到一致的理解，进一步开发实际系统。

原型模型后续也发生了一些演变，按照原型的作用不同，出现了抛弃型原型和演化型原型。抛弃型原型（快速模型）是将原型作为需求确认的手段，在需求确认结束后，原型就被抛弃不用，重新采用一个完整的瀑布模型进行开发。演化

性原型（变化模型）是在需求确认结束后，不断补充和完善原型，直至形成一个完整的产品。原型的概念也被后续出现的过程模型采纳，如螺旋模型和敏捷方法。这里架构新教材把演化模型归类在演化原型模型中。凯恩提醒大家注意分辨。

2.3.3.螺旋模型（重点★★★★★）

螺旋模型是瀑布模型与演化模型相结合，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。在螺旋模型中，软件开发是一系列的增量发布。

螺旋模型 （Spiral Model）是在快速原型的基础上扩展而成。也有人把螺旋模型归到快速原型，实际上，它是生命周期模型与原型模型的结合，如下图所示。

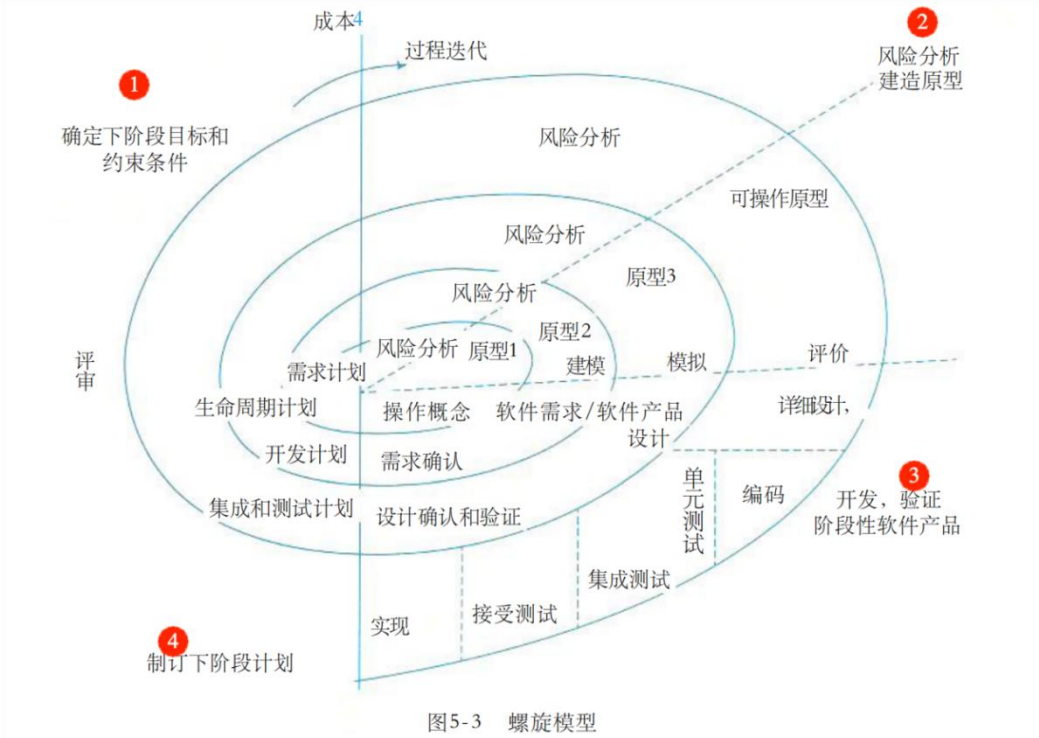


图5-3 螺旋模型

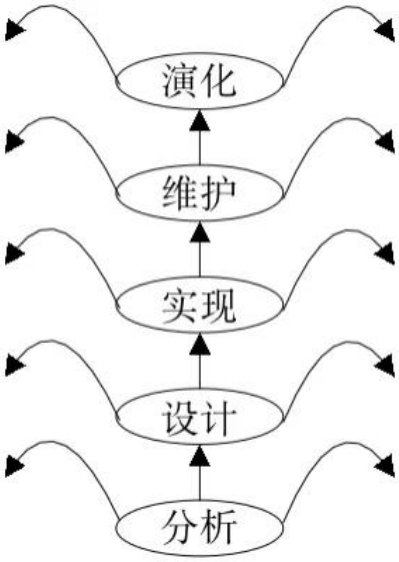
螺旋模型沿着螺线进行若干次迭代，每次迭代都包括制订计划（目标设定）、风险分析、实施过程和客户评估 4 个方面的工作。特别适用于庞大、复杂并具有高风险的系统。

该模型支持大型软件开发，适用于面向规格说明、面向过程和面向对象的软件开发方法，也适用于几种开发方法的组合。与瀑布模型相比，螺旋模型支持用户需求的动态变化，另外，过多的迭代次数会增加开发成本，延迟提交时间。

2.3.4.喷泉模型（次重点★★★☆☆）

喷泉模型是一种以用户需求为动力，以对象为驱动的模型，主要用于描述面向对象的软件开发过程。该模型认为软件开发过程自下而上的各阶段是相互重叠和多次反复的，就像水喷上去又可以落下来，类似一个喷泉。各个开发阶段没有特定的次序要求，并且可以交互进行，可以在某个开发阶段中随时补充其他任何开发阶段中的遗漏。

在喷泉模型中，各活动之间无明显边界，从而可以较容易地实现活动的迭代和无间隙，提高软件项目开发效率，节省开发时间。



2.3.5.变换模型（次重点★★★☆☆）

变换模型是基于形式化规格说明语言和程序变换的软件开发模型，它对形式

化的软件规格说明进行一系列自动或半自动的程序变换，最后映射为计算机能够接受的软件系统。

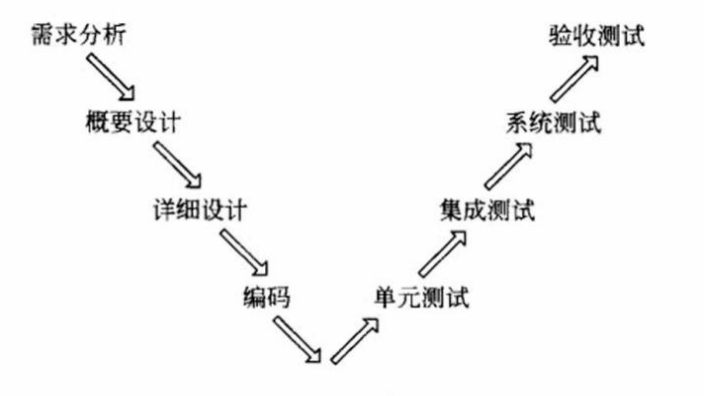
变换模型的优点是解决了代码结构性多次修改而变坏的问题，减少了许多中间步骤（例如，设计、编码和测试等）。但是，变换模型仍有较大局限，以形式化开发方法为基础的变换模型需要严格的数学理论和一整套开发环境的支持。

2.3.6.智能模型（次重点★★★☆☆）

智能模型也称为基于知识的软件开发模型，它综合了上述若干模型，并把专家系统结合在一起。该模型应用基于规则的系统，采用规约和推理机制，帮助开发人员完成开发工作，并使维护在系统规格说明一级进行。为此，需要建立知识库，将模型本身、软件工程知识与特定领域的知识分别存入知识库。

2.3.7.V 模型（重点★★★★★）

V 模型是在快速应用开发模型基础上演变而来，由于将整个开发过程构造成一个 V 字形而得名。V 模型应用在软件测试方面，和瀑布模型有着一些共同的特征。V 模型中的过程从左到右，描述了基本的开发过程和测试行为，其价值在于它非常明确地标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各阶段的对应关系，如下图所示。



在 V 模型中，单元测试是基于代码的测试，最初由开发人员执行，以验证程序代码的各个部分是否已达到预期的功能要求；集成测试验证了两个以上单元之间的集成是否正确，并有针对性地对详细设计中所定义的各单元之间的接口进行检查；在所有单元测试和集成测试完成后，系统测试开始以客户环境模拟系统的运行，以验证系统是否达到了在概要设计中所定义的功能和性能；最后，当技术部门完成了所有测试工作后，由业务专家或用户进行验收测试，以确保产品能真正符合用户业务上的需要。

V 模型强调软件开发的协作和速度，将软件实现和验证有机地结合起来，在保证较高的软件质量情况下缩短开发周期。V 模型适合企业级的软件开发。

2.3.9.统一过程（重点★★★★★）

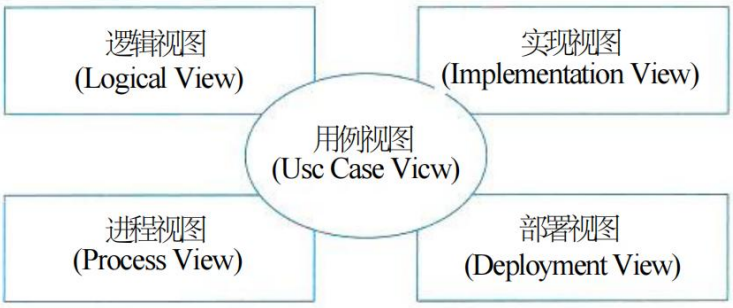
统一过程（Unified Process， UP）是一个通用过程框架，可以用于种类广泛的软件系统、不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。UP 是基于构件的，在为软件系统建模时，UP 使用的是 UML。与其他软件过程相比，UP 具有三个显著的特点，即用例驱动、以架构为中心、迭代和增量。

RUP 概述（重点★★★★★）

Rational Unified Process 是对 Rational Software（现在归 IBM 所有）创建的 Unified Process 的改进。RUP （Rational Unified Process）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 是用例驱动的、以体系结构为中心的、迭代和增量的软件开发过程。下面对这些特点做进一步的分析。

（1）用例驱动。RUP 中的开发活动是用例驱动的，即需求分析、设计、实现和测试等活动都是用例驱动的。

（2）以体系结构为中心。RUP 中的开发活动是围绕体系结构展开的。软件体系结构的设计和代码设计无关，也不依赖于具体的程序设计语言。RUP 采用“4+1”视图模型来描述软件系统的体系结构(这里的和第 17 章提到的 4+1 视图不同注意区别）。



（3）迭代与增量 RUP 强调采用迭代和增量的方式来开发软件，把整个项目开发分为多个迭代过程。在每次迭代中，只考虑系统的一部分需求，进行分析、设计、实现、测试和部署等过程；每次迭代是在已完成部分的基础上进行的，每次增加一些新的功能实现，以此进行下去，直至最后项目的完成。

生命周期（重点★★★★★）

RUP 软件开发生命周期是一个二维的软件开发模型， RUP 中有 9 个核心工作流，这 9 个核心工作流如下。（23 年系分考试直接问你这九大核心工作流是什么）

（1）业务建模 (Business Modeling)： 理解待开发系统所在的机构及其商业运作，确保所有参与人员对待开发系统所在的机构有共同的认识，评估待开发系统对所在机构的影响。

（2）需求 (Requirements)：定义系统功能及用户界面，使客户知道系统的功能，使开发人员理解系统的需求，为项目预算及计划提供基础。

（3）分析与设计 （Analysis & Dcsign）：把需求分析的结果转化为分析与设计模型。

（4）实现 (Implementation)： 把设计模型转换为实现结果，对开发的代码做单元测试，将不同实现人员开发的模块集成为可执行系统。

（5）测试（Test）：检查各子系统之间的交互、集成，验证所有需求是否均被正确实现，对发现的软件质量上的缺陷进行归档，对软件质量提出改进建议。

（6）部署 (Deployment): 打包、分发、安装软件，升级旧系统；培训用户及销售人员，并提供技术支持。

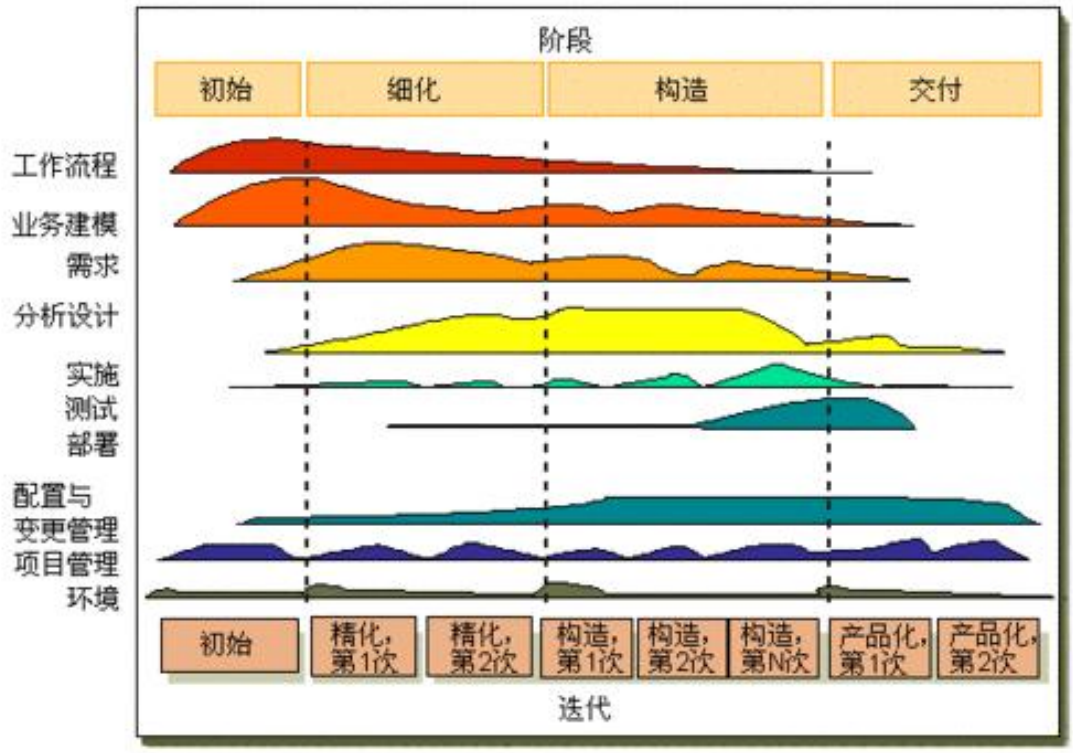
（7）配置与变更管理 (Configuration & Change Management): 跟踪并维护系统开发过程中产生的所有制品的完整性和一致性。

（8）项目管理 (Project Management): 为软件开发项目提供计划、人员分配、执行、监控等方面的指导，为风险管理提供框架。

（9）环境 (Environment): 为软件开发机构提供软件开发环境，即提供过程

管理和工具的支持。

RUP 把软件开发生命周期划分为多个循环，每个循环生成产品的一个新的版本，每个循环依次由 4 个连续的阶段组成，每个阶段完成确定的任务。这 4 个阶段如下。初始阶段：定义最终产品视图和业务模型，并确定系统范围。细化阶段：设计及确定系统的体系结构，制订工作计划及资源要求。构造阶段：构造产品并继续演进需求、体系结构、计划直至产品提交。移交阶段：把产品提交给用户使用。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。



核心概念（次重点★★★☆☆）

RUP 中定义了如下一些核心概念，（可能会出选择题）。

（1）角色 (Role)：Who 的问题。角色描述某个人或一个小组的行为与职责。

（2）活动（Activity）：How 的问题。活动是一个有明确目的的独立工作单元。制品（Artifact）：What 的问题。视频是活动生成、创建或修改的一段信息。也有些书把 Artifact 翻译为产品、工件等，和制品的意思差不多。

（3）工作流（Workflow）：When 的问题。工作流描述了一个有意义的连续的活动序列，每个工作流产生一些有价值的产品，并显示了角色之间的关系。

2.3.10.敏捷方法（重点★★★★★）

敏捷方法相对于“非敏捷”而言，它们更强调开发团队与用户之间的紧密协作、面对面的沟通、频繁交付新的软件版本、紧凑而自我组织型的团队等，也更注重人的作用。

敏捷宣言（重点★★★★★）

敏捷宣言认为，个体和交互胜过过程和工具；可工作的软件胜过大量的文档；客户合作胜过合同谈判；响应变化胜过遵循计划。

目前，主要的敏捷方法有极限编程（XP）、自适应软件开发（ASD）、水晶方法（Crystal）、特性驱动开发（FDD）、动态系统开发方法（DSDM）、测试驱动开发（TDD）、敏捷数据库技术（AD） 和精益软件开发等。虽然这些过程模型在实践上有差异，但都是遵循了敏捷宣言或者是敏捷联盟所定义的基本原则。这些原则包括客户参与、增量式移交、简单性、接受变更、强调开发人员的作用和及时反馈等。

特点（重点★★★★★）

敏捷方法是一种以人为核心、迭代、循序渐进的开发方法。在敏捷方法中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。

在敏捷方法中，从开发者的角度来看，主要的关注点有短平快的会议、小版本发布、较少的文档、合作为重、客户直接参与、自动化测试、适应性计划调整和结对编程；从管理者的角度来看，主要的关注点有测试驱动开发、持续集成和重构。敏捷方法主要适用于以下场合：

（1）项目团队的人数不能太多，适合于规模较小的项目。

（2）项目经常发生变更。敏捷方法适用于需求萌动并且快速改变的情况，如果系统有比较高的关键性、可靠性、安全性方面的要求，则可能不完全适合。

（3）高风险项目的实施。

（4）从组织结构的角度看，组织结构的文化、人员、沟通性决定了敏捷方法是否适用。

敏捷型方法是“适应性”（adaptive）而非“预设性”（predictive）的。敏捷的目的就是成为适应变化的过程，甚至能允许改变自身来适应变化。

敏捷型方法是“面向人的”（People-oriented）而非“面向过程的”（Process-oriented）。它们试图使软件开发工作能够充分发挥人的创造能力。它们强调软件开发应当是一项愉快的活动。

核心思想（重点★★★★★）

敏捷方法的核心思想主要有下面 3 点。

（1）敏捷方法是适应型，而非可预测型。（2）敏捷方法是以人为本，而非以过程为本。（3）迭代增量式的开发过程。

主要敏捷方法（重点★★★★★）

这里简单介绍 4 种影响比较大的敏捷方法。

（1）极限编程（Extreme Programming，XP）。在所有的敏捷型方法中， XP 是最引人注目的。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 4 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。

（2）水晶系列方法。水晶系列方法是由 Alistair Cockburn 提出的敏捷方法系列。它与 XP 方法一样，都有以人为中心的理念，但在实践上有所不同。其目的是发展一种提倡“机动性的”方法，包含具有共性的核心元素，每个都含有独特的角色、过程模式、工作产品和实践。

（3）Scrum。该方法侧重于项目管理。 Scrum 是迭代式增量软件开发过程，通常用于敏捷软件开发。 Scrum 包括了一系列实践和预定义角色的过程骨架（是一种流程、计划、模式，用于有效率地开发软件）。在 Scrum 中，使用产品 Backlog 来管理产品的需求，产品 Backlog 是一个按照商业价值排序的需求列表。根据 Backlog 的内容，将整个开发过程被分为若干个短的迭代周期 (Sprint)。

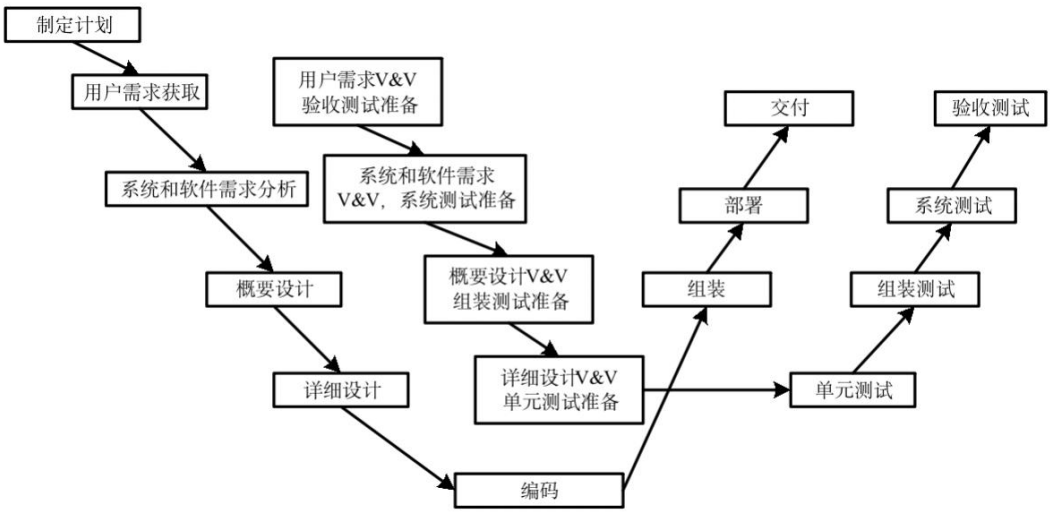
在 Sprint 中，Scrum 团队从产品 Backlog 中挑选最高优先级的需求组成 Sprint backlog。在每个迭代结束时， Scrum 团队将递交潜在可交付的产品增量。当所有 Sprint 结束时，团队提交最终的软件产品。

（4）特征驱动开发方法（FDD）。FDD 是由 Jeff De Luca 和大师 Peter Coad 提出来的。 FDD 是一个迭代的开发模型。 FDD 认为有效的软件开发需要 3 个要素：人、过程和技术。FDD 定义了 6 种关键的项目角色：项目经理、首席架构设计师、开发经理、主程序员、程序员和领域专家。根据项目大小，部分角色可以重复。FDD 有 5 个核心过程：开发整体对象模型、构造特征列表、计划特征开发、特征设计和特征构建。

2.3.11.W/H/X 模型（重点★★★★★）

23 年系统架构设计师已经考到过，凯恩建议大家重点关注。

W 模型由 Evolutif 公司公司提出，相对于 V 模型，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动。W 模型由两个 V 字型模型组成，分别代表测试与开发过程。W 模型中测试与开发对应关系如下：



W 模型强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发是同步进行的。

H 模型。H 模型指出软件测试要尽早准备，尽早执行，只要某个测试达到准

备就绪点，测试执行活动就可以开展。

X 模型。X 模型是一种基于用户故事的测试模型，可以提高测试效率和质量，但需要有专业的测试人员和用户参与。

2.5.软件过程管理（次重点★★★☆☆）

软件过程是软件生命周期中的一系列相关活动，即用于开发和维护软件及相关产品的一系列活动。软件产品的质量取决于软件过程，具有良好软件过程的组织能够开发出高质量的软件产品。

2.5.1.软件能力成熟度模型（次重点★★★☆☆）

软件能力成熟度模型（Capability Maturity Model for Software，简称 CMM）是由美国卡内基梅隆大学软件工程研究所（SEI）于 1987 年提出的一种用于评估和改进软件开发过程的框架。CMM 的核心思想是将软件开发视为一个过程，并通过定义、实施、度量、控制和改善各个阶段来提升软件开发的质量和效率。

CMM 的目的是帮助组织对软件过程进行管理和改进，增强开发与改进能力，从而能按时地、不超预算地开发出高质量的软件。CMM 的 5 个成熟度等级分别为初始级、可重复级、已定义级、已管理级和优化级。

CMM 分为五个成熟级别	
初始级	在这一级别，软件开发过程缺乏结构化，项目通常依赖于个人经验和口头指令。

可重复级	在这一级别，组织能够定义和文档化其软件过程，确保项目的成功。
已定义级	在这一级别，组织不仅定义了软件过程，还对这些过程进行了标准化和文档化。
已管理级	在这一级别，组织能够监控和量化其软件过程，以便进行持续改进。
优化级	在这一级别，组织能够动态地调整其软件过程以适应不同的项目需求。

2.5.2.能力成熟度模型集成 CMMI 量化等级（重点★★★★★）

CMMI 是 CMM 的升级版本，强调系统工程和软件工程的整合，适合于信息系统集成企业。CMMI 是一个更广泛的模型，不仅适用于软件工程，还包括系统工程、供应链管理和信息技术服务管理等多个领域。CMMI（能力成熟度模型集成）通过支持迭代开发过程和经济动机推动的组织，采用基于结果的方法来促进其发展。

CMMI 分为五个成熟级别	
初始级	过程通常是随意且混乱的。
已管理级	组织要确保策划、文档化、执行、监督和控制项目级的过程。
已定义级	能够根据自身情况定义己企业和项目的标准流程，将这套管理体系与流程予以制度化。
量化管理级	组织建立了产品质量、服务质量以及过程性能的定量目标。
优化级	在这一级别，组织能够动态地调整其软件过程以适应不同的项目需求。

2.7.软件项目管理（次重点★★★☆☆）

软件项目管理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（People）、产品（Product）、过程（Process）和项目（Project）进行分析和管理的活动。

2.7.1.软件进度管理（次重点★★★☆☆）

在软件进度管理过程中，一般包括活动定义、活动排序、活动资源估计、活动历时估计、制定进度计划和进度控制。

（1）工作分解结构 WBS。软件项目往往是比较大而复杂的，往往需要进行层层分解，将大的任务分解成一个个的单一小任务进行处理。 WBS 总是处于计划过程的中心，也是制订进度计划、资源需求、成本预算、风险管理计划和采购计划等的重要基础。

任务分解的 5 大基本要求。（1）WBS 的工作包是可控和可管理的，不能过于复杂。（2）任务分解也不能过细，一般原则 WBS 的树形结构不超过 6 层。（3）每个工作包要有一个交付成果。（4）每个任务必须有明确定义的完成标准。（5）WBS 必须有利于责任分配。

（2）任务活动图。活动定义是指确定完成项目的各个交付成果所必须进行的各项具体活动，需要明确每个活动的前驱、持续时间、必须完成日期、里程碑或交付成果。前在项目管理中，目前通常采用甘特图等方式来展示和管理项目活动。

2.7.2.3 软件配置管理（次重点★★★☆☆）

软件配置管理 SCM 是一种标识、组织和控制修改的技术。 SCM 活动的目标就是为了标识变更、控制变更、确保变更正确实现并向其他有关人员报告变更。软件配置管理核心内容包括版本控制和变更控制。

3.系统可靠性（选择丨案例丨论文）（重点★★★★★）

本章内容属于特别重点内容，凯恩总结如下。

（1）从选择角度来看，主要考察 4 个可靠性指标，以及串并联可靠性分析和可靠性评价。

（2）案例考察的很少，可以忽略。

（3）论文部分重点考察点就是可靠性评价和设计方法，2023 年已经考察《论软件可靠性评价的设计与实现》，2014 年已经考察过《论软件的可靠性设计》，2010 年已经考察过《论软件可靠性评价》，2009 年已经考察过《论软件可靠性设计与应用》。

3.1.软件可靠性设计（次重点★★★☆☆）

3.1.2.容错设计技术（次重点★★★☆☆）

（1）恢复块设计。将程序划分为一系列操作构成的恢复块。每个恢复块包含多个功能相同但设计不同的程序块。当检测到某个程序块出现故障时，可以切换到备用程序块。

（2）N 版本程序设计。设计多个不同的软件版本，对同样的输入采用多数表决的方式。要求各版本使用不同的算法、编程语言、测试方法等确保相互独立。可以防止单个版本故障导致整体失效。

（3）冗余设计。在主系统之外设计不同实现方式的备用软件模块。当主模块出现故障时，可以切换到备用模块。可以有效提高软件可靠性，但会增加额外开销。

3.1.3.检错技术（次重点★★★☆☆）

在软件中设置检测点和检测机制，检测软件运行状态。一旦发现故障，及时报警以便人工干预。实现方式包括监测返回结果、运行时间等。处理方式一般是停止软件并报警

3.1.4.降低复杂度设计（次重点★★★☆☆）

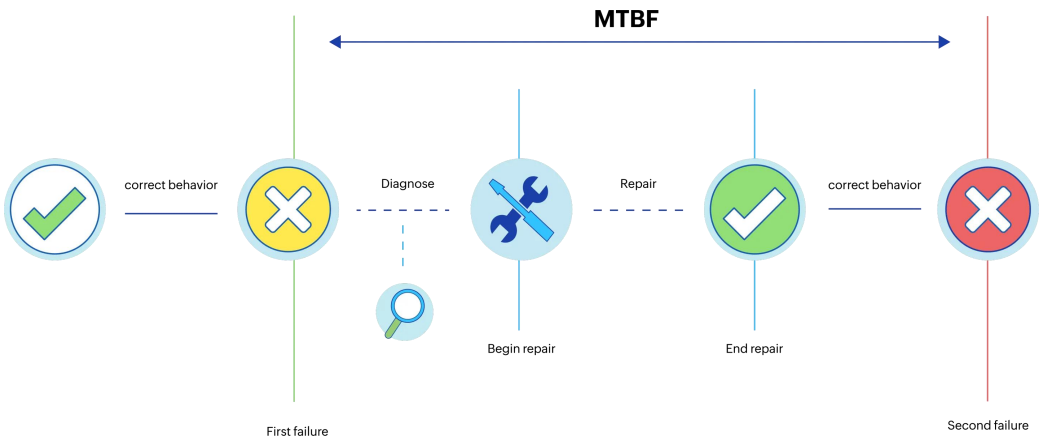
软件复杂度包括模块内部复杂性和模块间关联复杂性。过高的复杂度是软件缺陷的重要根源。通过简化软件结构、优化数据流等方式降低复杂度，提高可靠性。

3.1.5.系统配置技术（次重点★★★☆☆）

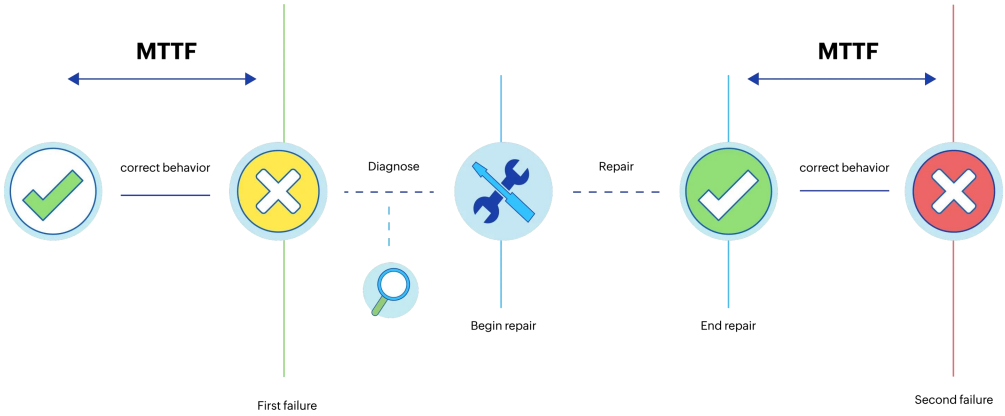
双机热备:采用主备服务器模式，通过"心跳"检测切换。服务器集群:多台服务器组成单一系统，当某台服务器故障时，其他服务器可接管。

3.2.软件可靠性的定量描述（重点★★★★★）

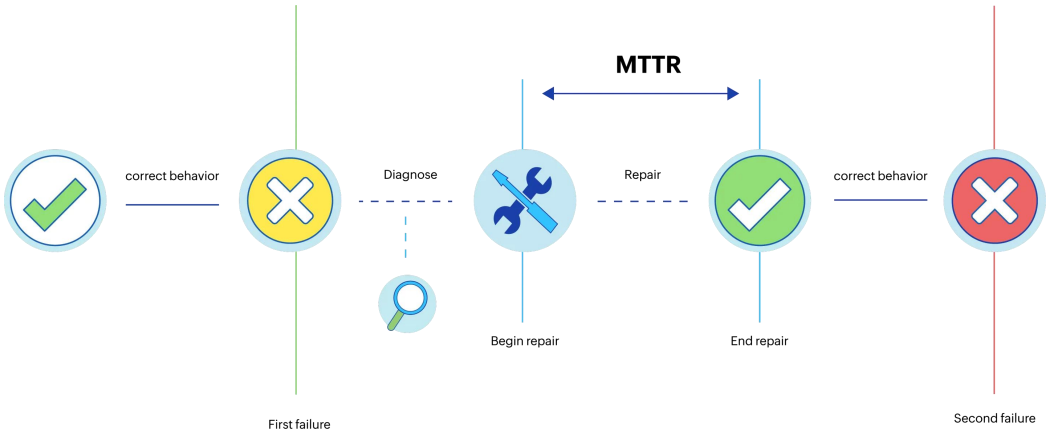
MTBF 故障间隔时间，第一次故障到第二次故障的间隔时间，单位是时间。



MTTF 平均无故障时间，单位是时间。



MTTR 平均故障修理时间，单位是时间



在实际应用中，一般 MTTR 很小，所以通常认为 MTBF 约等于 MTTF 。

3.3.影响软件可靠性的 5 大因素（重点★★★★★）

（1）运行剖面（环境）。（2）软件规模。（3）软件内部结构。

（5）软件的开发方法和开发环境。（6）软件的可靠性投入。

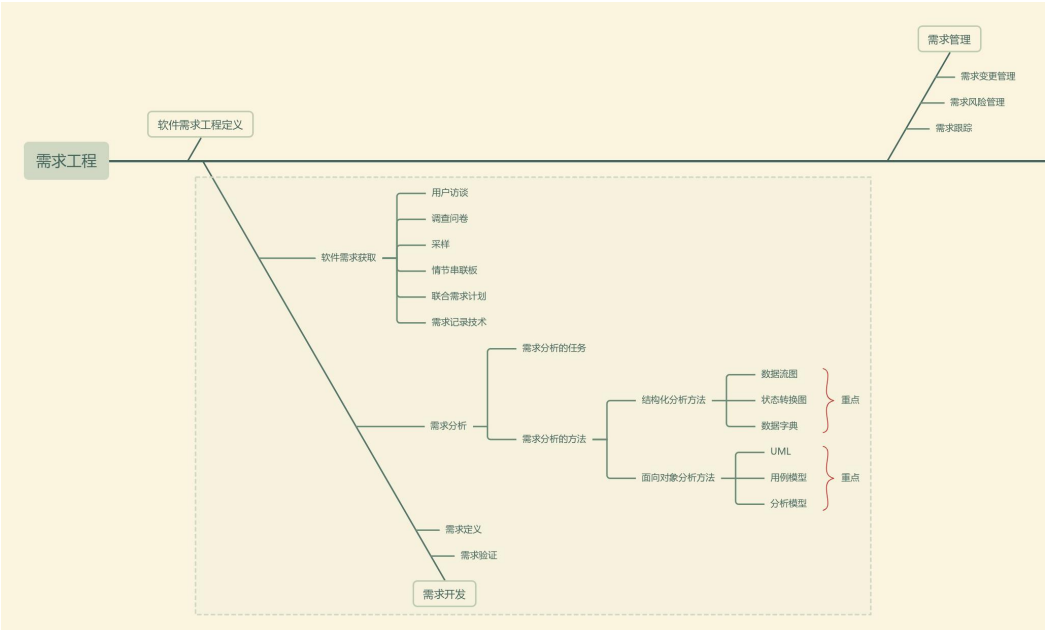
4.软件需求工程（选择丨案例丨论文）（重点★★★★★）

本章内容属于特别重点内容，凯恩总结如下。

（1）从选择角度来看，本章内容值得重点关注，特别是软件需求分析中的结构化分析数据流图，数据字典，面向对象分析中的用例模型，类图，顺序图，活动图等，以及需求管理中红字标出的相关概念。

（2）从案例考察角度来看，此章节中结构化分析，面向对象分析中的内容都可能当作一小问进行考察，直接考察相关概念，比如数据流图平衡原则，数据流图基本规范等，面向对象分析中用例图的概念，面向对象分析活动过程，类的关系，活动图，顺序图，状态图，流程图之间的异同比较等。

（3）论文部分重点考察点就是需求管理，结构化和面向对象分析方法，2023 年下半年考过《面向对象分析方法》2017 年已经考察《论软件系统建模方法及其应用》，2014 年考察过《论软件需求管理》，2011 年已经考察过《论软件需求获取技术及应用》，2009 年考察过《论信息系统建模方法》。



软件需求工程是包括创建和维护软件需求文档所必需的一切活动的过程，可分为需求开发和需求管理两大工作。需求开发包括需求获取、需求分析、编写需求规格说明书 （需求定义）和需求验证 4 个阶段。

需求管理通常包括定义需求基线、处理需求变更和需求跟踪等方面的工作。

4.1.软件需求概述（次重点★★★☆☆）

软件需求是指用户对新系统在功能、行为、性能、设计约束等方面的期望。

简单地说，软件需求就是系统必须完成的事以及必须具备的品质。需求是多层次的，包括业务需求、用户需求和系统需求，这三个不同层次从目标到具体，从整体到局部，从概念到细节。

质量功能部署（QFD） 是一种将用户要求转化成软件需求的技术，其目的是最大限度地提升软件工程过程中用户的满意度。为了达到这个目标，QFD 将软件需求分为三类：常规需求、期望需求和意外需求。

4.2.需求获取（次重点★★★☆☆）

需求获取是一个确定和理解不同的项目干系人的需求和约束的过程。这个章节你需要熟悉有哪些方法，防止案例进行考察你不知道从哪些角度入手进行解答。看熟悉就行，不需要一字一句背出来（系分的同学可能需要更加认真对待这一章，往年让你默写过需求获取的方法）。

4.2.1.用户访谈（次重点★★★☆☆）

用户访谈是获取需求的基本手段，有结构化和非结构化两种形式。为进行有效访谈，需在准备、访谈过程和后续工作三个方面进行组织。准备阶段包括确定访谈目的、用户、问题，并作出安排；访谈过程中要准时、寻找异常、深入调查、认真记录、注意措辞；后续工作包括吸收理解信息、复查结果、记录问题、发送备忘录。用户访谈具有灵活性，但也存在困难，需要系统架构设计师具备经验和沟通能力。

4.2.2.问卷调查（次重点★★★☆☆）

问卷调查可克服用户访谈的局限性，但也有缺点。制作调查表需确定问题类型、编写问题、设计格式。问卷调查的优点是高效、低成本、匿名、便于整理统计，缺点是缺乏灵活性、信息不全面、不利于细节回答、返还率低。可将用户访谈和问卷调查结合使用，先问卷调查，再针对结果进行小范围访谈。为提高问卷返还率，可向工作人员解释目的、说明重要性、拜托领导督促、参加会议解答问题、更改问题减少时间、设置奖品奖励。

4.2.3.采样（重点★★★★★）

采样是指从种群中系统地选出有代表性的样本集的过程，通过认真研究所选出的样本集，可以从整体上揭示种群的有用信息。

1．样本大小

采样技术的关键是如何确定样本集的规模，即如何确定样本大小，使样本具有代表性。有研究人员给出了一个用于确定样本大小的既简单又有效的公式：

样本大小 = α ×(可信度系数/可接受的错误)^2

其中，α 称为启发式因子，一般取值为 0.25; 可信度系数表示希望“种群数据包括了样本中的各种情况”有多大的可信度，这个值可以根据可信度从下表中查找出来。

可 信 度	可信度系数	可 信 度	可信度系数
99%	2.58	95%	1.96
98%	2.33	90%	1.65
97%	2.17	80%	1.28
96%	2.05	50%	0.67

例如，如果希望订单样本集包含的所有情况具有 90 %的可信度，那么样本大小计算如下：

样本大小 $=0.25\times(1.65/(1-0.90))^2=68.0625$

此公式在真题案例分析中出现过所以需要重点关注。也就是说，为了得到期望的可信度，需要采集 69 张订单。如果想得到更高的可信度，则采样规模会更大。如果已知每 10 张订单中可能有 1 张有问题，则启发式因子 $\alpha=0.1\times(1-0.1)$，那么样本大小为：

样本大小 $=0.1\times(1-0.1)\times(1.65/(1-0.90))^2=24.5025$

这时，采样规模将小很多，只需要选择 25 张订单。种群中选择样本的技术

主要有简单随机采样、分层采样、聚类采样、系统采样等，具体采取哪种选择方法，需要根据系统文档的数量、样本大小和实际情况进行决定。

2．采样的优缺点

采样技术可用于收集数据、采集访谈或观察用户，能加快数据收集、提高效率、降低成本、减少偏差，但样本规模的确定依赖于系统架构设计师的主观因素，对其个人经验和能力要求较高。

4.2.4.情节串联板（次重点★★★☆☆）

在需求获取过程中，系统架构设计师需探讨解决方案，可通过情节串联板技术帮助用户消除盲区、达成共识。情节串联板是一系列图片，通过图片辅助讲故事的方式叙述需求，有助于高效准确地沟通。其类型包括被动式、主动式和交互式，制作工具包括静态工具和动态工具。情节串联板的优点是生动、用户友好、交互性强，能对用户界面提供早期评审，缺点是花费时间多，降低需求获取速度。

4.2.5 联合需求计划（重点★★★★★）

联合需求计划（Joint Requirement Planning，JRP）是一种方法论，旨在通过客户或最终用户的参与来加速应用程序的设计和开发过程。该方法通常通过两到五天的集会，让开发者与顾客进行深入合作，以快速有效地探讨并达成共识，从而产生完整的需求文件。JRP 对一些问题最有歧义、需求最不清晰的领域十分有用，但会议的组织和相关人员的能力是难点。

4.2.6.其他需求获取方法（非重点☆☆☆☆☆）

收集资料。通过查阅已有的文档、报告、学术论文、行业标准和其他相关文

献来获取需求信息。

获取文档。向相关人员（如客户、业务部门、技术团队）索取与项目相关的文档，包括需求说明书、项目计划、流程图等。

参加业务实践。通过参与实际的业务流程和操作，亲身体验和观察业务活动，获取需求信息。

4.3.需求分析（次重点★★★☆☆）

一个好的需求应该具有无二义性、完整性、一致性、可测试性、确定性、可跟踪性、正确性、必要性等特性，因此，需要系统架构设计师把杂乱无章的用户要求和期望转化为用户需求，这就是需求分析的工作。

4.3.1.需求分析的任务（次重点★★★☆☆）

《软件需求》一书指出，需求分析的工作通常包括以下 7 个方面：包括绘制系统上下文范围关系图、创建用户界面原型、分析需求可行性、确定需求优先级、建立需求分析模型、创建数据字典和使用 QFD 等。

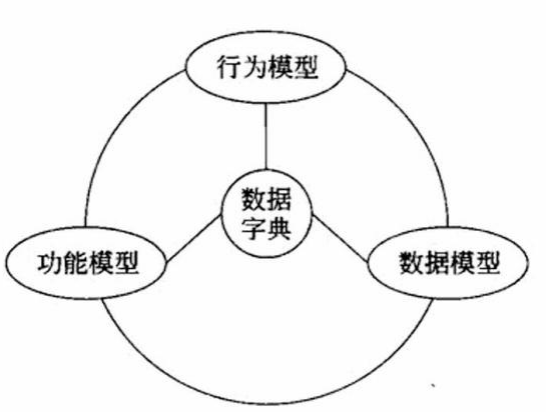
4.3.2.需求分析的方法（次重点★★★☆☆）

在软件工程实践过程中，人们提出了许多种需求分析的方法，其中主要有 SA 方法、面向对象分析方法和面向问题域的分析（PDOA） 方法。

PDOA 方法强调描述而非建模，其核心元素是问题框架，通过将问题域分为子域来获取更多信息。SA 方法关注功能的分层和分解，基于科学方法进行分析；面向对象分析方法则基于抽象、信息隐藏等理念，通过观测问题域事物的表象来建立逻辑对象。

4.4.结构化分析方法 SA（重点★★★★★）

SA 方法的基本思想是自顶向下，逐层分解，把一个大问题分解成若干个小问题，每个小问题再分解成若干个更小的问题。经过逐层分解，每个最底层的问题都是足够简单、容易解决的，于是复杂的问题也就迎刃而解了。在 SA 方法中导出的分析模型如图所示。



从图可以看出，SA 方法分析模型的核心是数据字典，围绕这个核心，有三个层次的模型，分别是数据模型、功能模型和行为模型（也称为状态模型）。在实际工作中一般使用 E-R 图表示数据模型（数据库这里提到过不展开了），用 DFD 表示功能模型，用状态转换图（STD） 表示行为模型。这三个模型有着密切的关系，它们的建立不具有严格的时序性，而是一个迭代的过程。

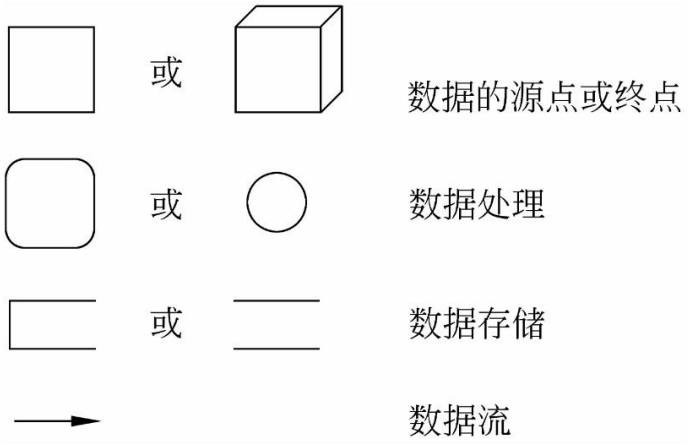
4.4.1.数据流图（重点★★★★★）

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。

1. DFD 的基本符号

在 DFD 中，通常会出现 4 种基本符号，分别是数据流、加工、数据存储和

外部实体（数据源及数据终点）。数据流是具有名字和流向的数据，在 DFD 中用标有名字的箭头表示。加工（数据处理）是对数据流的变换，一般用圆圈表示。数据存储是可访问的存储信息，一般用直线段表示。外部实体是位于被建模的系统之外的信息生产者或消费者，是不能由计算机处理的成分，它们分别表明数据处理过程的数据来源及数据去向，用标有名字的方框表示。



2．DFD 的层次

SA 方法的思路是依赖于 DFD 进行自上而下的分析。DFD 可以表现系统的高层和低层概念，通过先绘制高层 DFD，再对其中的加工进行逐步分解，直至系统被清晰描述。

看下图，圆圈就是加工，矩形是实体，箭头是数据流，两条直线是数据存储。有同学提出质疑，学员注册为啥既是加工又是存储。其实这里是简写，存储这里标注的学员注册你可以理解为学员注册表。