第5章 软件设计

学习目标

1. 理解软件体系结构与设计复用的相关概念和技术；
2. 了解软件设计策略与方法；
3. 了解软件设计工具、视图和软件设计表示方法；
4. 掌握结构化设计法的思想、原则、任务和工作步骤；
5. 掌握面向对象法三层设计思想、原则、建模工具和工作步骤；
6. 掌握数据库设计的工作步骤、设计内容和要求；
7. 理解界面设计的通用原则、交互模式及相关要求和设计步骤；
8. 培养学生以人为本、服务群众的意识；增强学生透过现象看本质、从事物本质分析问题的业务能力；培养学生精益求精、勇于创新、敢于实践、追求卓越的工匠精神。

设计是解决问题的一种方式，面对设计的各种约束和限制，往往没有确定性的设计方案。软件设计是软件开发过程中的重要组成部分，包括体系结构设计和详细设计。软件设计的输出结果是模型和工件，记录了设计所采用的主要决策及决策的合理性解释，有助于软件产品的长期可维护性。

5.1 软件体系结构与设计复用

软件体系结构(Software Architecture)由构成系统的元素、元素的相互作用、指导元素集成的模式以及这些模式的约束组成，为软件系统提供了一个结构、行为和属性的高级抽象。它不仅指定系统的组织结构和拓扑结构，显示系统需求和构成系统的元素之间的对应关系，并且提供一些设计决策的基本原理。软件体系结构是可传递和可复用的模型，通过研究软件体系结构可以预测软件的质量。

软件体系结构是项目干系人进行交流的手段，明确了对系统实现的约束条件，决定了开发和维护组织的组织结构，制约着系统的质量属性。软件体系结构使推理和控制更改更简单，有助于循序渐进的原型设计。

1. 构件

构件(Component)又称组件，是一个功能相对独立的具有可重用价值的软件单元。可以认为构件是一个封装的代码模块或大粒度运行模块，也可以认为构件是具有一定功能、能够独立工作或与其他构件组合起来协调工作的对象。

在结构化方法中，一个构件由一个功能完整的函数或过程构成，也称为模块；在面向对

象方法中，一个构件由一组对象构成，包含了一些协作的类的集合，它们的共同工作相当于某种系统功能。

2. 构件技术

对于构件，应当按可重用的要求进行设计、实现、打包、编写文档。构件应当是内聚的，并具有相当稳定的、开放的接口。

为了使构件更切合实际、更有效地被重用，构件应当具备可变性和通用性。可变性越好，构件就越易于调整，以便适用于应用的具体环境；通用性越好，其被重用的面就越广。针对不同的应用系统，只需对其可变部分进行适当地调整，重用者要根据重用的具体需要，改造构件的可变特性。

3. 软件复用

可复用性是指软件构件或产品（如设计模型、代码、文档等）能重复使用的程度。软件复用是使用已有的软件产品来开发新的软件系统的过程。

软件复用的形式大体可分为垂直式复用和水平式复用。水平式复用是复用不同应用领域中的软件元素，例如数据结构、排序算法、人机界面构件等。垂直式复用是在一类具有较多公共性的应用领域之间复用软件构件。由于在两个截然不同的应用领域之间进行软件复用潜力不大，所以垂直式复用受到广泛关注。

软件复用的范围不仅涉及源程序代码，软件开发的全生命周期都有可复用的价值，包括项目的组织、成本估计、架构、软件需求、规格说明、设计、源程序代码、用户文档和技术文档、实现、用户界面、数据结构、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。

4. 软件复用技术

软件复用是提高软件生产力和质量的重要技术，软件复用技术包括体系结构样式、设计器模式、程序族和框架等。

1）体系结构样式

体系结构样式是指对软件组成元素及其关系的类型的限定和使用约束。它从高层定义了可复用的体系结构设计。主要的体系结构样式包括以下 5 种：

（1）通用结构：如分层样式、管道过滤样式、黑板样式；

（2）分布式系统：如客户端-服务器样式、三层样式、代理样式；

（3）交互式系统：如模型-视图-控制器样式、表示-抽象-控制样式等；

（4）自适应系统：如微内核样式、反射样式；

（5）其他：如批处理样式、解释器样式、进程控制样式、基于规则的样式等。

2）设计器模式

设计器模式是指针对特定上下文中的特定公共问题的共性解决方案，它从较低层次描述了可复用的设计细节，主要设计模式包括：

（1）创建型模式：构造器模式、工厂模式、原型模式、单例模式；

（2）结构型模式：适配器模式、桥接器模式、组合模式、装饰器模式、剖面模式、享元模式、代理模式；

（3）行为型模式：命令模式、解释器模式、迭代模式、中介模式、备忘录模式、观察者模式、状态模式、策略模式、模板模式、访问者模式。

3）程序族

复用软件设计和组件的另一种途径就是设计程序族，也称软件产品线，需要识别出族中成员的共性，设计可复用可裁剪的组件来解决族内成员之间的可变性，如手机软件产品线、航空软件产品线。

4）框架

框架是“半成品”软件系统，实现了系统中的共性体系结构和组件，并通过插件等机制对尚未实现的部分进行补充和扩展，常见的组件有 Spring、MyBatis、Node.js 等。

5.2 软件设计策略与方法

有很多通用策略用以指导软件设计过程。相对于软件设计的通用策略，软件设计方法则专注于提供一组特定的符号、规范化的过程和使用该方法的指南。软件工程师通常使用这些方法作为通用的设计框架。

1. 通用策略

在软件设计过程中常见的通用策略有分而治之策略、自顶向下策略、逐步求精策略、自底向上策略、启发式策略、模式与模式语言策略，迭代与增量策略等。

2. 结构化设计方法

结构化设计是一种经典的面向过程的软件设计方法，它将系统过程分解成一个容易实现和维护的计算机程序模块，其核心是识别软件的主要模块，然后自顶向下按照各种设计规则和设计指南逐层分解和细化这些模块，模块需要满足高度内聚和松散耦合的特征。结构化设计通常在结构化分析之后使用，基于结构化分析所产生的数据流图和相关过程描述，使用各种策略将数据流图转换为软件体系结构的表示（模块结构图）。

3. 面向对象的设计方法

面向对象的设计从现实世界中客观存在的事务出发，来构造软件系统，并在系统的构造中尽可能地运用人类的自然思维方式，把系统看成对象的集合，每个对象都有自己的数据，用对象来完成所需要的任务，对象根据情况执行一定的行为。面向对象分析与设计的建模语言是 UML。后续发展到基于构件的设计，可通过诸如反射等机制来定义和访问软件设计中的元信息。尽管面向对象设计根源于数据抽象的概念，责任驱动的设计也已被广泛用作面向对象设计的另一种方法。

4. 以数据结构为中心的设计方法

以数据结构为中心的设计是一种用来计划、分析和设计信息系统的模型驱动的、以数据为中心但对过程敏感的技术，始于程序所操作的数据结构，而不是它所执行的函数。软件工程师首先描述软件输入/输出的数据结构，然后再根据这些数据结构图开发程序的控制结构，主要工具是数据模型图。

5. 基于构件的设计方法

软件构件是一个独立单元，具有良好定义的接口和依赖关系，可以被组合和独立部署。基于构件的设计所解决的主要问题是构件的提供、开发和集成，目的是提高可复用性。已有的可复用软件构件应和新软件一样满足相同的安全性需求。信任管理是另一个设计关注点，通常情况下一个构件不应依赖于其他具有更低可信度的构件或服务。

6. 其他方法

(1) 迭代与自适应方法：实现软件增量，不强调严格的软件需求和设计。如原型化方法是一种反复迭代过程，它需要设计人员和用户之间保持紧密的工作关系，通过构造一个预期系统的小规模的、不完整的但可工作的示例来与用户交互设计结果。原型设计方法鼓励并要求最终用户主动参与，这增加了最终用户对项目的信心和支持。原型更好地适应最终用户总是想改变想法的自然情况。原型是主动的模型，最终用户可以看到并与之交互。

(2) 面向方面设计：一种通过使用方面(Aspect)来实现软件需求分析中所识别出的横切关注点和对软件进行横切拓展的软件设计方法。

(3) 面对服务的体系结构：一种通过在分布式计算机上运行服务的方式来构建分布式软件的方法，通过标准协议(如 HTTP、HTTPS、SOAP)支持服务器通信与服务信息交换，一个软件系统可使用不同提供商提供的服务来构建。

(4) 模型驱动设计：模型驱动设计是一种系统设计方法，强调通过绘制图形化系统模型描述系统的技术和实现。通常从模型驱动分析中开发的逻辑模型导出系统设计模型，最终，系统设计模型将作为构造和实现新系统的蓝图。

(5) 快速应用开发。快速应用开发是一种系统设计方法，是各种结构化技术与原型化技术和联合应用开发技术的结合，用以加速系统开发。快速应用开发要求反复地使用结构化技术和原型化技术来定义用户的需求并设计最终系统。

5.3 软件设计表示

软件设计有很多软件设计工具，采用的工具不一样，呈现出的软件表示可能不一样；不同的视角描述软件体系结构，表示也不一样；同时有些表示方法可用于描述软件设计制品，有些用于体系结构设计和详细设计，还有一些表示方法专门用于特定的软件设计方法。

5.3.1 软件设计工具

1. 软件体系结构建模工具

软件体系结构建模工具为软件体系结构的可视化和设计阶段的分析推理提供工具支撑。可分为三类：

(1) 支持图形化和形式化的软件体系结构建模语言。

为软件体系结构的设计过程提供逐渐分解和细化的支持，从多个视角(如静态结构、动态结构等)来描述体系架构，如 UML、IDEF、BPMN 等。

(2) 软件体系结构描述语言(ADL)。

由于 ADL 形式化程度较高，此类工具支持对系统的定量化分析。

(3) 支持软件设计决策的建模工具。

帮助设计人员对体系结构设计中的问题、方案、决策、理由进行显式建模，完成从需求到体系结构的设计过程。部分工具还提供了体系结构方案，支持设计级的复用。

2. 模型驱动的软件体系结构工具

该类工具的核心是模型和模型转化，强调各阶段(需求、设计等)软件制品的模型化，并支持模型之间的自动和半自动转化。

3. 用户界面设计工具

界面设计包括三方面的设计：软件构件与构件之间的接口设计、软件内部与外部系统之间的接口设计、软件与使用者之间的交互设计。用户界面设计工具通常特指第三方面的设计，不仅对界面进行详细描述，还对交互流程做出完整说明。用户界面设计工具为图形化用户界面的快速设计提供了良好支持。

4. 数据库设计工具

数据库设计工具帮助用户可视化设计数据库结构，可产生数据流程图、概念数据模型、逻辑数据模型和物理数据模型，并提供数据管理功能。通常包含：用于创建复杂数据建模的实体一关系(ER)模型、正向和逆向数据库工程，为用户创建高质量数据模型、为数据完整性和一致性提供支持。一般能够支持一种或多种数据库系统。

5.3.2 软件视图

可以从各种不同的高层剖面描述软件体系结构并形成文档，这些剖面通常称为视图。每个视图表达了软件系统及体系结构的特定属性。一般可以从四个不同的视角加场景来描述软件体系结构。每一个视图只关心系统的一个侧面，四个视图结合在一起才能反映系统的软件体系结构的全部内容。

1. 逻辑视图(Logic View)

主要关注功能需求的满足性，即系统提供给最终用户的服务。在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表逻辑视图，用类图来描述逻辑视图。逻辑视图中使用的风格为面向对象的风格，逻辑视图设计中要注意的主要问题是要保持一个单一的、内聚的对象模型贯穿整个系统。

2. 开发视图(Development View)

开发视图也称为模块视图(Module View)，主要关注系统如何分解为实现单元并显式地表达它们之间的依赖关系。开发视图侧重于软件模块的组织和管理，通过系统输入/输出关系的模型图和子系统图来描述。可以在确定了软件包含的所有元素之后描述完整的开发视图，也可以在确定每个元素之前，列出开发视图原则。

3. 进程视图(Process View)

进程视图关注系统的并发问题，侧重于系统的运行特性，主要关注一些非功能性的需求，例如系统的性能和可用性。进程视图强调并发性、分布性、系统集成性和容错能力，以及逻辑视图中的主要抽象如何适合进程结构。

4. 物理视图(Physical View)

物理视图主要关注系统的物理结构与分布问题，如考虑如何把软件映射到硬件上，它通常要考虑到解决系统拓扑结构、系统安装、通信等问题。

面向对象开发方法中也把软件视图分为五类：用例视图、逻辑视图、并发视图、组件视图和部署视图。

场景(Scenarios)可以看作那些重要系统活动的抽象，它使4个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。在开发体系结构时，它可以帮助设计者找到体系结构的构件和它们之间的作用关系。同时，也可以用场景来分析一个特定的视图，或描述不同视图构件间是如何相互作用的。场景可以用文本表示，也可以用图形表示。

5.3.3 软件设计表示方法

软件设计通常会使用多种表示方法。这里将软件设计的表示方法分为结构描述(静态视图)、行为描述(动态视图)两类。

1. 结构描述

以下基于图形化或形式化方法的软件设计表示方法均可用于描述软件的结构,即描述构成系统的主要组件及其互连方式,作为软件设计的静态视图。

(1) 体系结构描述语言(ADL):以组件和连接件的方式描述软件体系结构的文本语言,通常是形式化的。

(2) 类图与对象图:用于表示一组类(或对象)及其相互关系。

(3) 构件图:用于表示一组构件及其相互关系,这里的"构件"是指遵从抽象接口定义并提供相应接口实现的物理部件。

(4) 类—职责—协作卡(CRC):用于标识组件(类)的名字、职责、组件之间协作关系的名字。

(5) 部署图:用于表示一组物理节点及其相互关系,并建模表示软件的物理特性。

(6) 实体关系图(ERD):用于表示数据的概念模型。

(7) 接口描述语言(DL):与编程语言类似的语言,用于定义软件组件的接口。

(8) 结构图:用于描述程序的调用结构,即一个模块会调用哪些其他模块以及个模块会被哪些其他模块所调用。

2. 行为描述

以下表示方法和语言用于描述软件系统与组件的动态行为,有些是图形化形式的,有些是文本形式的,大多适用于详细设计阶段。行为描述还可以包含设计决策的依据与理由。

(1) 活动图:用于描述活动之间的控制流,也可用于表示多活动之间的并发。

(2) 通信图:用于描述一组对象之间发生的交互,重点描述对象、对象间的链接,以及它们通过这些链接交换的消息。

(3) 数据流图(DFD):用于显示元素之间的数据流。数据流图是对操作类元素所构成的网络中的信息流的描述,其中每个元素使用或修改流入该元素的信息,并产生流出该元素的信息。由于数据流可用于识别可能攻击和暴露保密信息的路径,数据流及数据流图也可用于安全性分析。

(4) 决策表与决策图:用于表示条件与动作的复杂组合。

(5) 流图:用于表示控制流及被执行的关联操作。

(6) 时序图:用于显示一组对象之间的交互,主要强调对象之间消息传递的时间顺序。

(7) 状态转换图与状态图:用于显示从一个状态到另一个状态的控制流,以及状态机中各组件基于其当前状态的行为变化。

(8) 形式化规约语言:是一种文本语言,使用基本的数学符号来严格、抽象地定义软件组件的接口与行为,通常采用前置条件和后置条件的形式。

(9) 伪代码与程序设计语言(PDL):类似于结构化程序设计语言,用于描述过程或方法的行为,通常用在详细设计阶段。

5.4 结构化设计方法

结构化设计是一种面向数据流的方法，把系统看成一些与数据交互的过程，数据与过程隔离保存，当程序运行时，就创建或者修改数据文件。它以软件需求规格说明书（SRS）和结构化分析（SA）阶段所产生的文档、数据流图和数据字典等为基础，是一个自顶向下、逐步求精和模块化的过程。

5.4.1 结构化设计的思想

结构化分析方法的基本思想是自顶向下逐层分解。对于一个复杂的问题，很难一下子考虑问题的所有方面和全部细节，通常可以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，经过多次逐层分解，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。

结构化设计把系统看作一个过程的集合体加数据存储，利用数据流图（Data Flow Diagram，DFD）、数据字典（Data Dictionary，DD）来分析数据，利用功能结构图和模块结构图来描述系统系统体系结构，利用结构化语言、判定表、判定树描述数据加工的过程。

5.4.2 结构化设计的原则

在结构化方法中，模块化是一个很重要的概念，它是将一个待开发的软件分解成为若干个小的简单部分——模块，每个模块可以独立开发、测试。这是一种复杂问题的“分而治之”原则，其目的是使程序的结构清晰，易于测试与修改。

通常将模块的接口和功能定义为其外部特性，将模块的局部数据和实现该模块的程序代码称为内部特性。而在模块设计时，最重要的原则就是实现信息隐蔽和模块独立。

信息隐蔽原则：信息隐蔽是开发整体程序结构时使用的法则，即将每个程序的成分隐蔽或封装在一个单一的设计模块中，并且尽可能少地暴露其内部的处理。通常将难的决策、可能修改的决策、数据结构的内部连接，以及对它所做的操作细节、内部特征码、与计算机硬件有关的细节等隐蔽起来。通过信息隐蔽可以提高软件的可修改性、可测试性和可移植性，它也是现代软件设计的一个关键原则。

模块独立原则：模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系最简单。保持模块的高度独立性，也是设计过程中的一个很重要的原则。通常用耦合（模块之间联系的紧密程度）和内聚（模块内部各元素之间联系的紧密程度）两个标准来衡量，设计的目标是高内聚、低耦合。

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。

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，松敞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。

除了满足以上两大基本原则之外，通常在模块分解时还需要注意：保持模块的大小适中，尽可能减少调用的深度，直接调用该模块的个数应该尽量大，但调用其他模块的个数则不宜过大；保证模块是单入口、单出口的；模块的作用域应该在控制域之内；功能应该是可预测的。

5.4.3 结构化设计的任务

结构化设计方法特别适合数据处理领域的问题，但是不适合解决大规模的、特别复杂的项目，主要包括以下四个内容，最终表现形式主要是模块结构图，模块结构图中的元素包括模块、调用、数据、控制信息和转接符号。

(1) 体系结构设计：根据数据流图进行体系结构设计，定义软件的主要结构元素及其关系。

(2) 数据设计：根据数据字典和实体关系图进行数据设计，基于实体联系图确定软件涉及的文件系统的结构及数据库的表结构。

(3) 接口设计：根据数据流图进行接口设计，描述用户界面，软件和其他硬件设备、其他软件系统及使用人员的外部接口，以及各种构件之间的内部接口。

(4) 过程设计：根据加工规格说明书和控制规格说明书进行过程设计，确定软件各个组成部分内的算法及内部数据结构，并选定某种过程的表达形式来描述各种算法。

5.4.4 结构化设计的两个阶段

结构化设计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。

概要设计：又称为总体结构设计，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。

详细设计：主要任务是在概要设计将系统开发的总任务分解成许多个基本的、具体的任务的基础上，进一步为每个具体任务选择适当的技术手段和处理方法。

5.4.5 结构化设计的工作步骤

(1) 分析研究业务流程：调研了解业务流程，绘制出业务流程图。

(2) 建立系统逻辑模型。在业务流程图的基础上，舍去实物，保留数据流，画出数据流图，建立数据字典。

(3) 进行概要设计：设计软件的结构、确定系统是由哪些模块组成，以及每个模块之间的关系。它采用结构图(包括模块、调用、数据)来描述程序的结构，此外还可以使用层次图和 HIPO(层次图加输入/处理/输出图)。

(4) 进行详细设计：确定应该如何具体地实现所要求的系统，得出对目标系统的精确描述。采用自顶向下、逐步求精的设计方式和单入口单出口的控制结构。常使用的工具包括程序流程图、盒图、PAD 图、PDL 等。

5.4.6 结构化设计案例

1. 业务描述

由需要购置设备的部门填写申购表，将此表格送交设备处，设备科填写预算表送财务处，财务处核对后，将资金返回设备处，设备处利用资金购买设备，将购得设备送需要购置设备的部门，将收据送财务处。业务流程图如图 5-1 所示。

2. 数据流图

系统分析阶段根据业务流程图，去除实物等计算机不可处理的东西，通过现象看本质，分析计算机能处理的数据，梳理出数据流，并本着精益求精、勇于创新敢于实践、追求卓越的

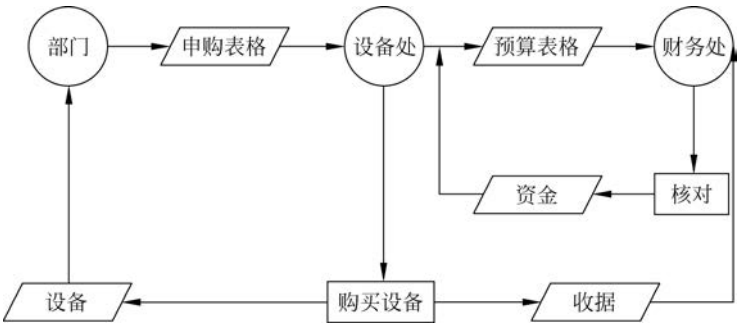

图 5-1　业务流程图

工匠精神对数据流进行核查，甚至创新地进行业务重构，重整数据流，建立系统的逻辑模型——数据流图，同时建立数据字典对数据流图进行补充说明，如图 5-2 所示。

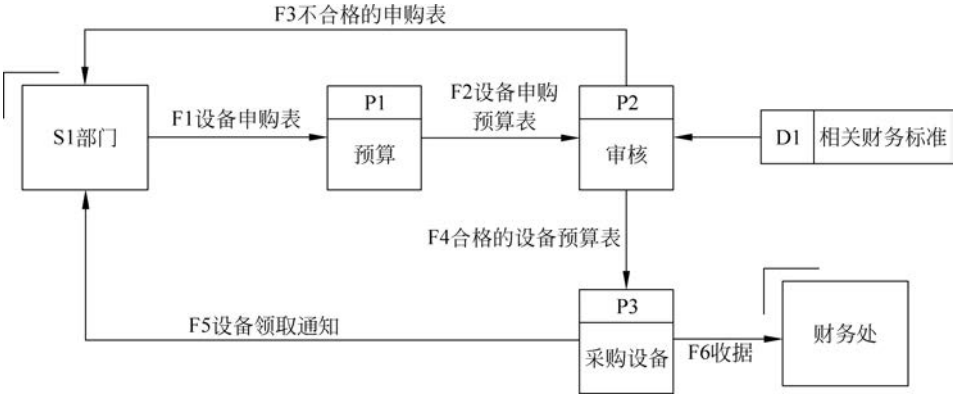

图 5-2　数据流图

3. 模块结构图

系统设计阶段特别是概要设计阶段的主要任务是根据分析阶段获得的数据流图，采用变换分析法，将数据流图转换为模块结构图——系统体系结构，如图 5-3 所示。

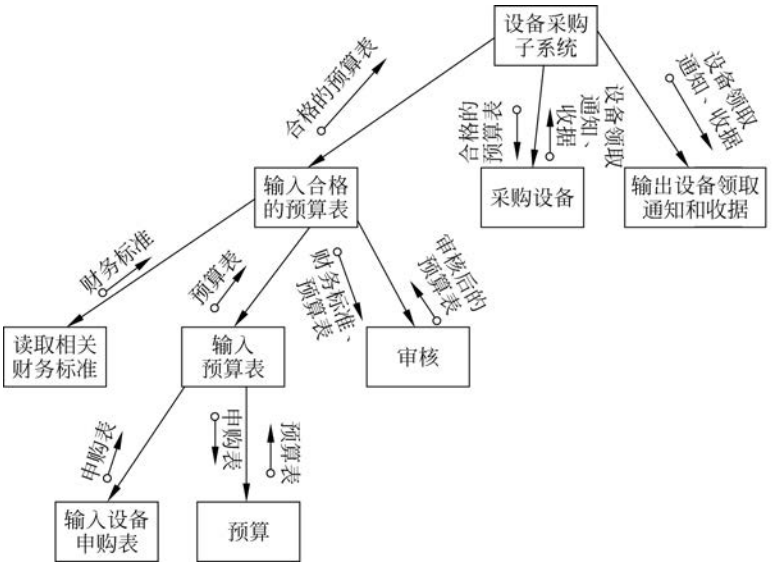

图 5-3　模块结构图

然后在利用 IPO 图、NS 图、数据流图等对各模块进行详细设计。

5.5 面向对象设计法

与传统方法相比，把系统看成对象的集合，每个对象都有自己的数据，用对象来完成所需要的任务，对象根据情况执行一定的行为。面向对象的设计从现实世界中客观存在的事物出发，来构造软件系统，更接近人类的自然思维方式。

5.5.1 面向对象三层设计思想

面向对象设计法从现实世界中客观存在的事物出发来建立软件系统，强调直接以问题域（现实世界）中的事物为中心来思考问题、认识问题，并根据这些事物的本质特征，把它们抽象地表示为系统中的对象，作为系统的基本构成单位。在面向对象方法中，把一切都看成是对象。具有共同属性和操作的对象可以抽象成类，类又可以被分为实体类、接口类和控制类三个层次。

（1）实体类：实体类的对象是显示世界中真实的实体，如人类、动物等；

（2）接口类：接口类的对象为用户提供一种与系统交互的方式，分为人和系统两大类，其中人的接口可以是显示屏、窗口、Web 窗体、菜单等；

（3）控制类：该类的对象用来控制活动流，充当协调者。

5.5.2 面向对象设计原则

（1）单一职责原则：设计目的单一的类。

（2）开放—封闭原则：对扩展开放，对修改封闭（多扩展，少修改）。

（3）李氏替换原则：子类可以替换父类。

（4）依赖倒置原则：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程。

（5）接口隔离原则：使用多个专门的接口比使用单一的总接口要好。

（6）组合重用原则：要尽量使用组合，而不是继承关系达到重用的目的。

（7）迪米特原则（最少知识法则）：一个对象应当对其他对象有尽可能少的了解。

5.5.3 面向对象软件设计建模工具

UML 一系列图可以分为结构图和行为图或者分为静态图和动态图。

静态图/结构图包括：用例图、类图、对象图、包图、组合结构图、构件图、部署图。

动态图/行为图包括：顺序图/序列图、通信图/协作图、定时图、状态图、活动图、交互概览图。

5.5.4 面向对象设计的工作步骤

面向对象开发方法分分析与设计两个阶段，但并不像结构化方法那么明显，很多情况下都是迭代增量的一个开发过程，两个阶段产生的分析模型和设计模型所用的表示图形都一样，知识详略程度不一样而已，聚焦点不一样而已。

1. 面向对象的分析(OOA)

运用面向对象方法,对问题域和系统责任进行分析和理解,找出描述问题域及系统责任所需的对象,定义对象的属性、操作以及它们之间的关系。其目标是建立一个符合问题域、满足用户需求的 OOA 模型。

问题域即被开发系统的应用领域,即在现实世界中由这个系统进行处理的业务范围。系统责任指所开发的系统应该具备的职能。

2. 面向对象的设计(OOD)

从 OOA 到 OOD 不是转换,而是调整和增补。使 OOA 作为 OOD 模型的问题域部分,增补其他四部分,成为完整的 OOD 模型。有不同的侧重点和不同的策略 OOA 主要针对问题域,识别有关的对象以及它们之间的关系,产生一个映射问题域,满足用户需求,独立于实现的 OOA 模型。OOD 主要解决与实现有关的问题,基于 OOA 模型,针对具体的软、硬件条件产生一个可实现的 OOD 模型。

5.5.5 面向对象设计案例

1. 业务描述

网络的普及带给了人们更多的学习途径,随之而来的管理远程网络教学的"远程网络教学系统"诞生了。"远程网络教学系统"的功能需求如下:

管理员进入系统后,可以创建课程,指定任课教师,将课程信息保存在数据库中并可以对课程进行改动和删除。

教师进入系统后,可以更新维护课程,上传课件、教学视频等。

学生登录网站后,可以选课、浏览课件、查找课件、下载课件、观看教学视频,学习课程,进行课程测试,系统自动批卷。

用户需要登录"远程网络教学系统"后才能正常使用该系统的所有功能。如果忘记密码,可与通过"找回密码"功能恢复密码。

在"远程网络教学系统"中,学生或教师可以在客户的 PC 上通过浏览器,如 IE 9.0 等,登录到远程网络教学系统中。在 Web 服务器端,安装 Web 服务器软件,部署远程网络教学系统,Web 服务器与数据库服务器连接。数据库服务器使用 SQL Server 2000 提供数据服务。

2. 用例模型

根据业务描述,首先需要建立用例模型,即用例图和用例描述。用例图如图 5-4 所示。

用例描述可以使用正文描述(包括用例名、参与者、前置条件、事件流和后置条件),也可以用活动图进行描述。

3. 静态模型

主要是对系统的静态结构进行描述,依据用例图,利用三层设计原理分别先后确定问题域、GUI 类和数据访问类,其中最核心的是问题域类的确定。通过现象看本质,分析用例实现需要哪些对象类来支撑,每个对象类应该定义哪些属性和操作,并本着精益求精、勇于创新、敢于实践、追求卓越的工匠精神对对象类进行核查。运用动名词方法找出候选类,然后进行筛选,确定如图 5-5 所示的问题域类图,再利用 CRC 卡片法检查其合理性。

4. 动态模型

根据用例模型和静态模型,为每一个用例建立相应顺序图或协助图。顺序图横轴是对

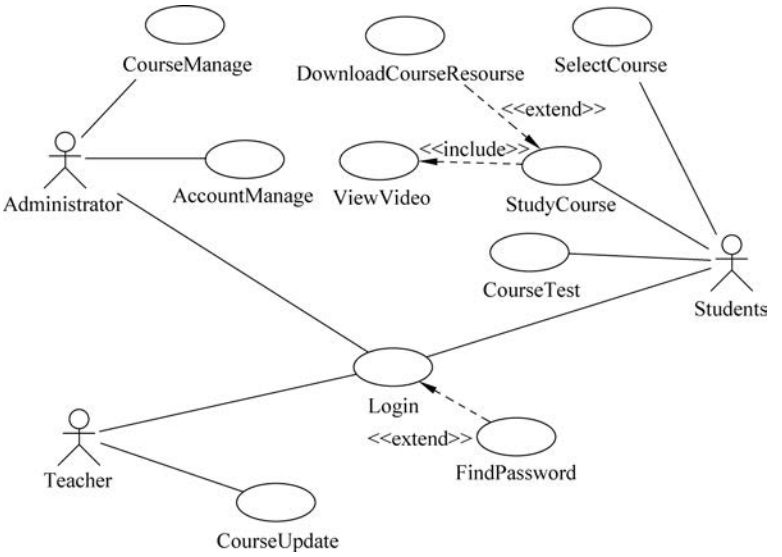

图 5-4　用例图

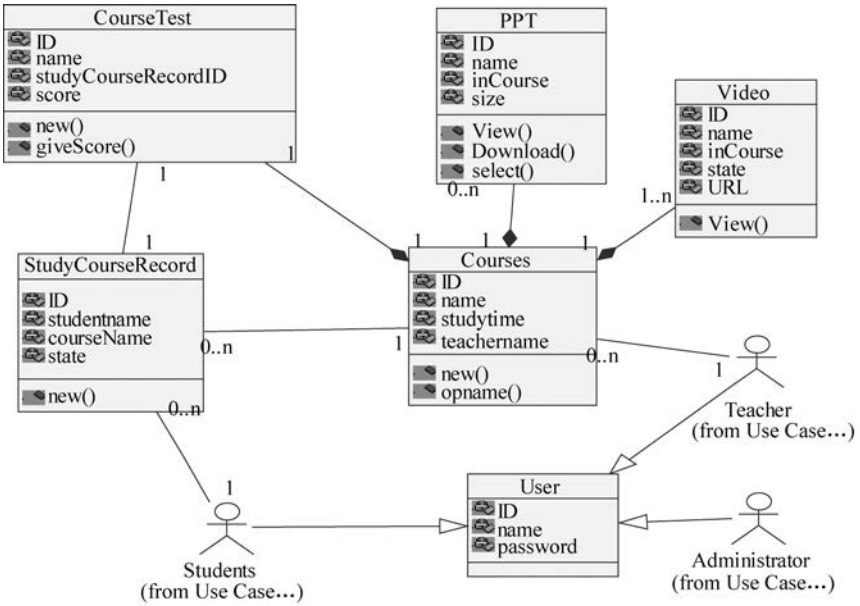

图 5-5　类图

象轴，所呈现的对象一定要跟类图中的类对应，纵轴为时间轴，表示时间自上而下顺延。描述其动态实现过程。如创建课程顺序图如图 5-6 所示，学生选课顺序图如图 5-7 所示。

5. 系统部署

系统部署主要描述系统的软硬件分布情况，可分别用组件图和部署图来表示，部署图中用立方体表示硬件。部署图中用带阴影的立方体表示处理器，有一定的数据处理能力，如服务器、工作站等；不带阴影的立方体表示设备，如打印机、扫描仪、网卡等。组件图中用一个大矩形和两个小矩形表示组件，即一组文件，分组标准不一样，组件图也不一样。本例组件图和部署图如图 5-8 和图 5-9 所示。

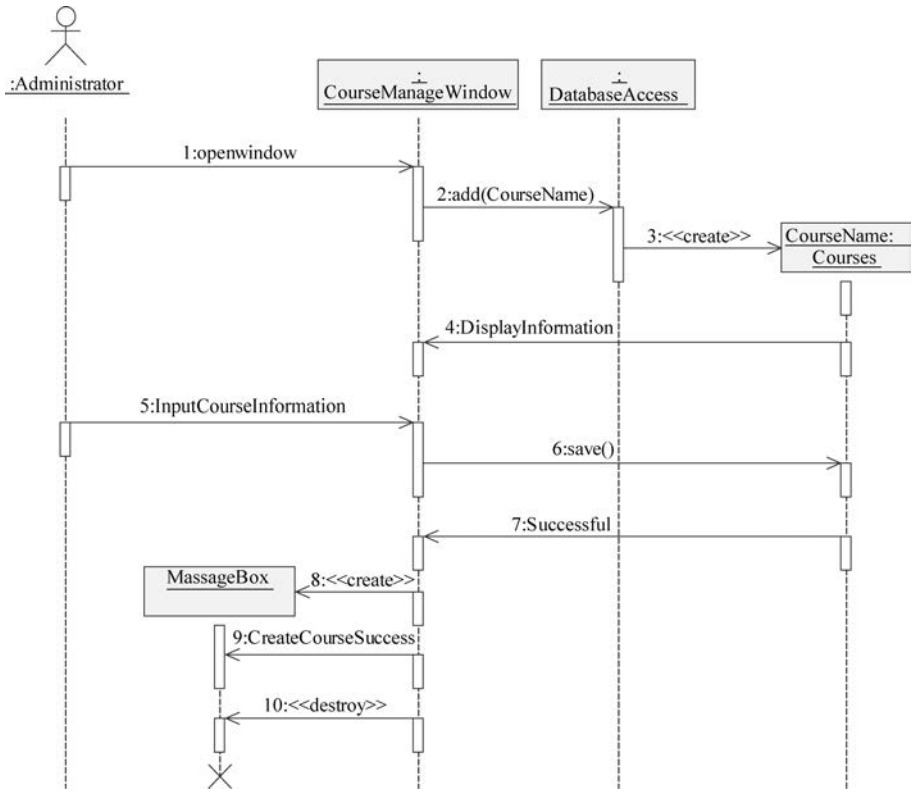

图 5-6　创建课程顺序图

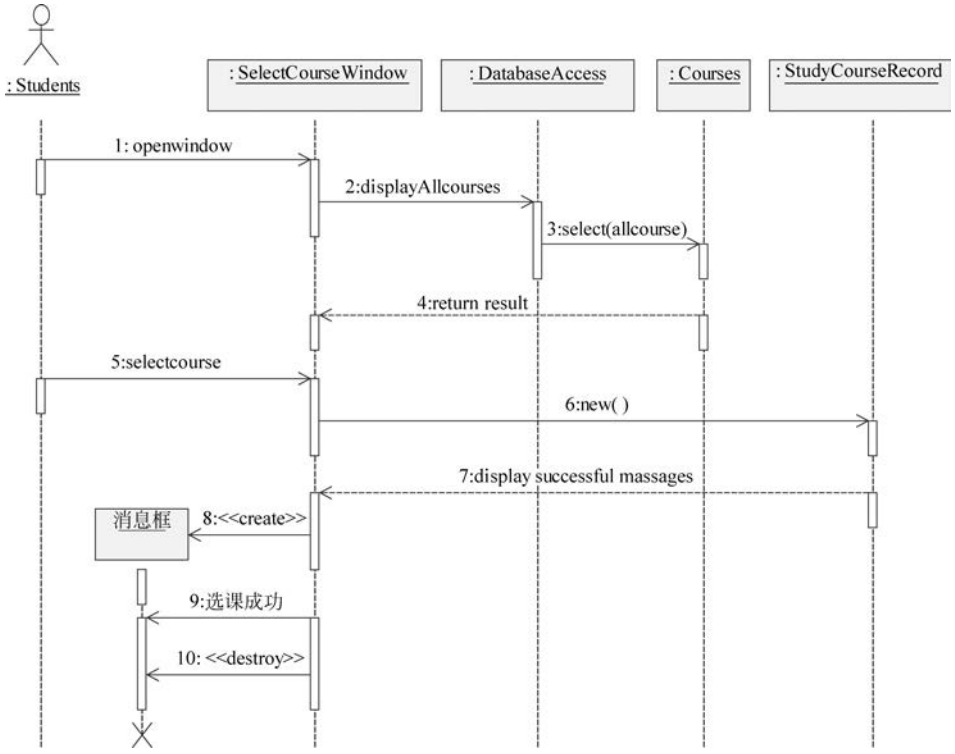

图 5-7　学生选课顺序图

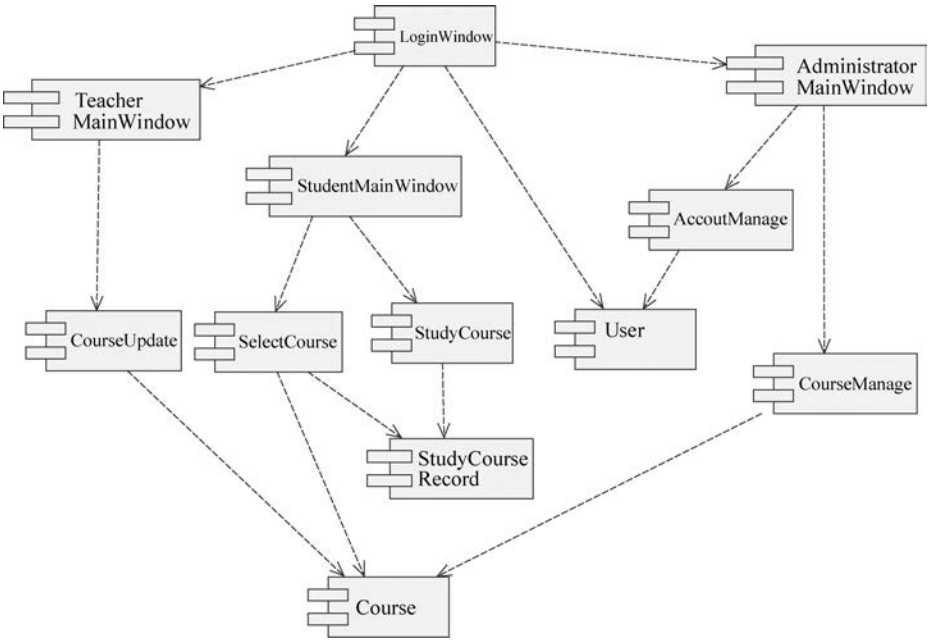

图 5-8　组件图

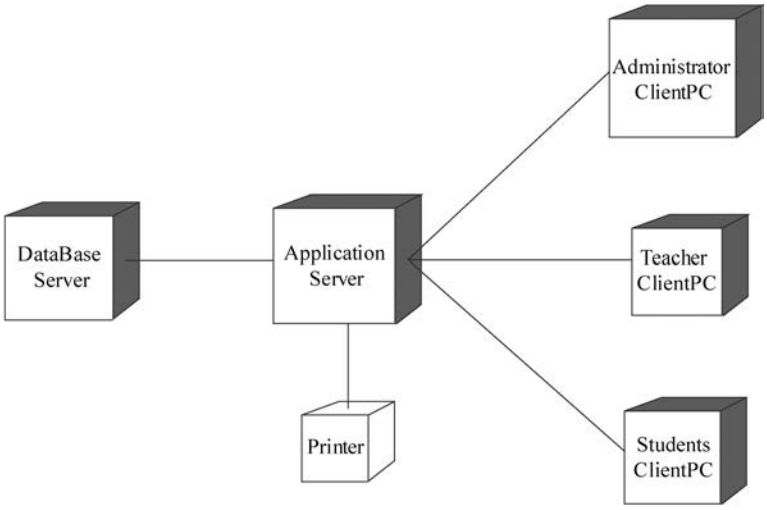

图 5-9　部署图

5.6　数据库设计

按照规范设计，将数据库的设计过程分为六个阶段：系统需求分析阶段、概念结构设计阶段、逻辑结构设计阶段、物理结构设计阶段、数据库实施阶段、数据库运行与维护阶段，其中需求分析和概念结构设计独立于任何数据库管理系统。

1. 系统需求分析

需求分析的任务：对现实世界要处理的对象进行详细的调查，通过对原系统的了解，收

集支持新系统的基础数据并对其进行处理，在此基础上确定新系统的功能。可细分为调查分析用户活动；收集和分析需求数据，确定系统边界信息需求，处理需求，安全性和完整性需求；编写系统分析报告。

2. 概念结构设计

概念结构设计的目标是设计数据库的 E-R 模型图，确认需求信息的正确性和完整性。具体来说，就是从需求分析中找到实体，确认实体的属性、确认实体的关系，画出 ER 图。

3. 逻辑结构设计

逻辑结构设计的任务是将概念结构设计阶段完成的实体模型转换成特定的 DBMS 所支持的数据模型的过程。逻辑结构设计的目的是将 E-R 图中的实体、属性和联系转换为关系模式。

实体间关系转换遵循的原则：

一个实体转换为一个关系模式，实体的属性就是关系的属性，实体的键就是关系的键。

一个联系转换为一个关系模式，与该联系相连的各实体的键以及联系的属性均转换为该关系的属性。

应用数据库设计的范式理论对初始关系模型进行优化。数据库设计的三大范式如下：

第一范式：每一个分类必须是一个不可分的数据项。属性不可再分，确保每列的原子性。

第二范式：要求每个表只描述一件事情，每条记录有唯一标识列。

第三范式：数据库表中不包含已在其他表中已包含的非主关键字信息。

4. 物理结构设计

物理结构设计：对于给定的逻辑数据模型，选取一个最适合应用环境的物理结构，确定数据库管理系统、数据存储方法和存储结构等，评价时间和空间效率。

5. 数据库实施

数据库实施是根据逻辑设计和物理设计的结果，在计算机上建立实际的数据库结构、装入数据、进行测试和试运行的过程。

6. 数据库运行与维护

数据库运行与维护的主要任务包括维护数据库的安全性与完整性、监测并改善数据库性能、重新组织和构造数据库，只要有数据库系统在运行，就需要不断地进行修改、调整和维护。

5.7 用户界面设计

用户界面(UI，User Interface)也称人机界面，是人机交互、操作逻辑和界面表现的整体设计，是软件设计过程中的必要组成部分。用户界面的设计质量与用户体验直接相关，是产品最接近用户的部分，是产品的“脸面”。

如果说计算机由硬件和软件构成，硬件为计算机提供信息处理的环境支持，软件为计算机提供信息处理的方案，帮助用户解决问题，用户使用电脑实际是在使用软件，本质上是与软件的用户界面打交道的过程。为使软件能发挥全部潜能，用户界面的设计应与其目标用户的技能、经验和期望相吻合。用户界面设计跟用户体验直接相关，必须要有用户思维，要

站在用户的角度思考设计人机交互方式让用户操作和控制软件系统。

5.7.1 通用界面设计原则

通用界面设计原则包括以下 7 方面。

(1) 易学性。软件界面应该简单容易上手,减少用户的记忆负担,从而用户可以快速掌握。

(2) 用户友好性。界面应该使用用户熟悉的术语和概念。

(3) 一致性。不同的界面应表现一致,即同类操作应采用相同方式加以执行。

(4) 自然性。用户界面的使用方式和交互行为不应让用户感到超出其日常经验,界面元素设计应符合常规逻辑,与用户习惯一致,保证用户可理解,如用户印象里“齿轮”表示“设置”功能,“头像”表示个人中心功能。

(5) 可恢复性。界面应允许用户犯错,并具有错误发生后返回正常状态的机制。

(6) 用户指南。当错误发生时,界面应为用户提供有意义的反馈和上下文相关的帮助。

(7) 用户多样性。界面应为各类用户及残疾人(如盲人、聋哑人、视弱者、色盲者等)提供符合各自特征的交互机制。

5.7.2 用户界面设计的关键问题

用户界面设计的内容包含两方面:用户与软件进行交互的方式(用户交互);软件反馈给用户的信息如何呈现给用户的方式(信息呈现)。用户界面设计应在恰当的用户交互方式、信息呈现方式、用户的背景和经验、可用设备的特性 4 方面之间折中。

5.7.3 用户交互模式的设计

用户交互模式分为以下 6 类。

(1) 问答:交互过程被严格限制为用户和软件之间的一一问答,即用户向软件提出一个问题,软件向用户返回该问题的答案。

(2) 直接操纵:用户与计算机屏幕上的对象进行交互,通常经由定位设备(如鼠标、轨迹球、触摸屏上的手指)来操纵屏幕对象,同时触发该对象执行特定的动作。

(3) 菜单选择:用户从菜单列表选中相应的操作。

(4) 填表:用户填写表格字段。表格中也可包含菜单,用户可以从中选择启动特定操作的动作按钮。

(5) 命令语言:用户发出一个命令(包含相应的参数),以告诉软件应该做什么。

(6) 自然语言:用户以自然语言发出指令,该指令进一步被解释和翻译成软件内部的可执行指令。

5.7.4 信息呈现设计

信息呈现可以采用文字或图形形式。好的用户界面设计应使信息呈现的方式与信息本身分离开来。MVC(Model-View-Controller)模型是一种将信息本身与其呈现方式有效分离的机制。

软件工程师在设计信息呈现时还需要考虑响应时间和反馈。响应时间是指用户从执行

特定的操作开始直到软件给出特定响应所经历的时间。软件执行操作过程中，应给出当前完成进度，完成后进行反馈（例如，重述用户的输入）。

另外，当要呈现给用户的信息量很大时，可以对其进行抽象，

根据信息展示的类型，设计人员可使用颜色来增强界面的表现力，但须遵循以下指导原则：

（1）限制使用的颜色数量。

（2）通过颜色的改变来反映软件执行状态的变化。

（3）使用颜色编码支持用户任务。

（4）颜色编码的使用应慎重并且一致。

（5）使用颜色帮助色盲及色弱用户进行信息访问（如改变颜色的饱和度和亮度，尽量避免使用蓝色和红色的组合等）。

（6）不要仅仅依赖颜色为残疾人用户（如盲人、视弱者、色盲者等）传递重要信息。

5.7.5 用户界面设计过程

用户界面设计是一个迭代的过程，包括三个核心活动：

（1）用户分析。设计人员分析用户负责执行的任务、所处的工作环境、用户如何与其他用户进行交互。

（2）软件原型开发。通过开发软件原型，引导用户提出对当前界面设计的意见，以不断完善界面设计，支持用户界面的持续演化直到用户完全满意。

（3）界面评估。在评估过程中设计人员需观察和评估用户在界面演化过程中的体验，发现待改进问题，进入下一轮迭代。

5.7.6 本地化和国际化

用户界面设计通常需要考虑本地化和国际化，让软件能够适应不同语言、不同地区、目标市场的不同技术需求。国际化是指设计软件使之能够通过简单的修改即可被应用到不同语言和不同地区的过程。本地化是指通过增加本地文字翻译，使软件能够适应特定地区或语言的过程。本地化和国际化需要考虑的因素包括符号，数字、货币、时间和计量单位等。

5.7.7 隐喻和概念模型

界面设计人员可使用隐喻和概念模型建立软件与现实世界之间的映射，从而帮助用户更快地学习和使用界面，如使用垃圾桶图标表示指令“删除文件”，作为对该概念的隐喻。需要注意的是，对于特定概念，设计人员不应同时使用多种隐喻，以避免用户混淆。因为不同文化中隐喻的含义和用法可能是不同的，隐喻也可能会为国际化带来潜在问题。

5.7.8 CRAP 设计原则

界面设计中的各个元素需要分清主次关系，图片、文字、线条等都需要有一定的排版秩序才能更好地抓住重点。美国设计大师 Robin Williams 在《写给大家看的设计书》中提出了视觉设计的 CRAP 原则。CRAP 原则是四项基本设计原理，包括对比（Contrast）、重复（Repetition）、对齐（Alignment）、亲密（Proximity），已经被设计师广泛应用。CRAP 原则简

单实用，在网页设计中对文字的排版也非常适用。

(1) 对比(Contrast)：如果两个项不完全相同，就应当使之不同，而且应当是截然不同(强烈)的。对比的目的有两个：一是增强页面的表现效果；二是有助于界面信息的组织。

(2) 重复(Repetition)：设计的某些方面(元素)需要在整个作品中重复，重复的目的就是统一，并增强视觉效果。

(3) 对齐(Alignment)：任何元素都不能在界面上随意安放。每一项都应当与界面上的某个内容存在某种视觉联系。对齐的目的是使界面统一而有条理。

(4) 亲密性(Proximity)：将相关的项组织在一起，移动这些项，使它们的物理位置相互靠近。亲密性的目的是实现界面信息的组织化，形成视觉的模块化。在你将界面中的相关元素放在一起展示时，也使界面的空白区域(留白)更加整洁、美观。

5.7.9 用户界面设计的流程

用户界面设计的流程，其实就是设计原则中的任务项的倒叙排列。如下：

(1) 理解产品目标及核心功能；

(2) 根据不同硬件设备分别设计；

(3) 根据用户习惯选择元素；

(4) 优化界面逻辑；

(5) 精简界面元素；

(6) 突出核心功能；

(7) 用户测试；

(8) 修改初稿；

(9) 审核提交设计。

优秀的用户界面设计，通常有以下特征：引导用户视觉；配色合理(与产品功能相符、色彩搭配科学)；考虑用户场景。用户界面的好坏，最终还是要用产品说话，要让用户评价的。

5.8 软件设计质量分析与评价

软件质量是软件产品具有满足规定的或隐含要求能力要求有关的特征与特征总和。影响软件设计的质量因素有很多，包括可维护性、可移植性、易测试性、易用性、正确性、健壮性等。

质量属性可分为 3 类：运行时可辨识的质量属性(如性能、安全性、有效性、功能性、易用性)，运行时不可辨识的质量属性(如可修改性、可移植性、可复用性、易测试性)，与架构内在质量相关的属性(如概念完整性、正确性、完整性)。它们之间存在细微的差异。

5.8.1 软件质量分析与评价技术

分析和评价软件设计质量的工具和技术有很多，概括起来可分为以下 3 种。

1. 软件设计评审

用于判定设计制品质量的形式化或非形式化技术，例如，体系结构评审、设计评审与审

查、基于场景的技术、需求跟踪等。软件设计评审可以评价安全性，也可以评审安装、操作和使用等辅助文档（如用户手册与帮助文件）。

2. 静态分析

用于评价设计的形式化或非形式化的静态分析方法，例如，故障树分析[①]和自动交叉检查等。这里的“静态”是指不需要执行软件即可对质量进行分析和评价。例如，针对安全性，可进行设计的脆弱性分析，通过静态分析发现安全漏洞。形式化的设计分析使用数学模型帮助设计人员预测软件的行为和验证软件的性能，从而不再完全依赖于测试。形式化的设计分析也可用于检测设计规格说明书和设计中的错误（不精确性、歧义性、其他错误等）。

3. 仿真与原型法

用于评价设计的动态技术，如性能仿真和可行性原型等。

5.8.2 软件质量度量

度量可用于评估或定量估算软件设计的各个方面，如规模、结构或质量。现有大多数度量都依赖于软件设计所采用的方法，分成两类：

1. 基于功能的结构化设计度量

结构化设计通常采用结构化图表（层次图）描述设计方案，基于这些图表可计算软件功能分解相关的各种度量值。

2. 面向对象的设计度量

设计结构通常用类图表示，并基于此计算出类的各种度量值以及每个类中内部属性的度量值。

5.8.3 软件质量评估

1. 三层结构模型

根据软件质量标准 GB/T 8566—2001G，软件质量评估通常从对软件质量框架的分析开始。软件质量框架是一个“质量特征—质量子特征—度量因子”的三层结构模型：

第一层质量特征是面向管理的质量特征，每一个质量特征是用以描述和评价软件质量的一组属性，代表软件质量的一个方面。软件质量不仅从该软件外部表现出来的特征来确定，而且必须从其内部所具有的特征来确定。

第二层的质量子特征是上层质量特征的细化，一个特定的子特征可以对应若干个质量特征。软件质量子特征是管理人员和技术人员关于软件质量问题的通信渠道。

最下面一层是软件质量度量因子（包括各种参数），用来度量质量特征。定量化的度量因子可以直接测量或统计得到，为最终得到软件质量子特征值和特征值提供依据。

2. 软件质量特征

按照软件质量标准 GB/T 8566—2001G，软件质量可以用下列特征来评价：

（1）功能特征：与一组功能及其指定性质有关的一组属性，这里的功能是满足明确或隐含需求的那些功能。

（2）可靠特征：在规定的时间和条件下，与软件维持其性能水平的能力有关的一组

① 故障树分析是自上至下的演绎式失效分析方法。——编者注

属性。

(3) 易用特征：由一组规定或潜在的用户为使用软件所需做的努力和所做的评价有关的一组属性。

(4) 效率特征：与在规定条件下软件的性能水平与所使用资源量有关系的一组属性。

(5) 可维护特征：与进行指定的修改所需的努力有关的一组属性。

(6) 可移植特征：与软件从一个环境转移到另一个环境的能力有关的一组属性。

3. 软件质量评估指标选取原则

选择合适的指标体系并使其量化是软件测试与评估的关键。评估指标可以分为定性指标和定量指标两种。理论上讲，为了能够科学客观地反映软件的质量特征，应该尽量选择定量指标。但是对于大多数软件来说，并不是所有的质量特征都可以用定量指标进行描述，所以不可避免地要采用一定的定性指标。在选取评估指标时，应做到以下原则：

(1) 针对性：即不同于一般软件系统，能够反映评估软件的本质特征，具体表现就是功能性与高可靠性。

(2) 可测性：即能够定量表示，可以通过数学计算、平台测试、经验统计等方法得到具体数据。

(3) 简明性：即易于被各方理解和接受。

(4) 完备性：即选择的指标应覆盖分析目标所涉及的范围。

(5) 客观性：即客观反映软件本质特征，不能因人而异。

应该注意的是，选择的评估指标不是越多越好，关键在于指标在评估中所起作用的大小。如果评估时指标太多，不仅增加结果的复杂性，有时甚至会影响评估的客观性。指标的确定一般是采用自顶向下的方法，逐层分解，并且需要在动态过程中反复综合平衡。

5.8.4 软件质量评估指标体系

1. 功能性指标

功能性是软件最重要的质量特征之一，可以细化成完备性和正确性。目前对软件的功能性评价主要采用定性评价方法。

(1) 完备性：完备性是与软件功能完整、齐全有关的软件属性。如果软件实际完成的功能少于或不符合研制任务书所规定的明确或隐含的那些功能，则不能说该软件的功能是完备的。

(2) 正确性：正确性是与能否得到正确或相符的结果或效果有关的软件属性。软件的正确性在很大程度上与软件模块的工程模型和软件编制人员的编程水平有关。

对这两个子特征的评价依据主要是软件功能性测试的结果，评价标准则是软件实际运行中所表现的功能与规定功能的符合程度。在软件的研制任务书中，明确规定了该软件应该完成的功能，那么即将进行验收测试的软件就应该具备这些明确或隐含的功能。

目前，对于软件的功能性测试主要针对每种功能设计若干典型测试用例，软件测试过程中运行测试用例，然后将得到的结果与已知标准答案进行比较。所以，测试用例集的全面性、典型性和正确性是功能性评价的关键。

2. 可靠性指标

根据相关的软件测试与评估要求，可靠性可以细化为成熟性、稳定性、易恢复性等。对

于软件的可靠性评价主要采用定量评价方法。即选择合适的可靠性度量因子，然后分析可靠性数据而得到参数具体值，最后进行评价。

设计软件时不仅要考虑功能性问题，还要考虑许多关键的非功能需求的满足问题，如性能、安全保密性、可靠性、易用性等。这些问题是与软件所在的应用领域无关的通用问题，一般不是软件功能分解的单元，而是以系统方式影响组件的性能或语义的属性。软件设计中非功能性的关键问题包括：并发性、事件控制与处理、数据持久性、组件的分布、异常处理与容错、交互与表现、保密安全性等。

复习思考题

1. 什么是软件体系结构？软件设计复用的相关技术有哪些？

2. 软件设计策略有哪些？各有什么优缺点？

3. 软件设计视图可以分哪几种？各包含哪些图形工具？

4. 结构化设计法的思想是什么？有哪些设计原则？概要设计和详细设计的任务是什么？可用哪些模型或图来表示？

5. 面向对象法三层设计思想是什么？设计的先后次序如何？面向对象的设计原则有哪些？

6. 目前面向对象建模的主流工具 UML 中有哪些图形？哪些可用于建模静态模型？哪些可用于建模动态模型？

7. 数据库设计的工作步骤有哪些？各自的设计内容和要求是什么？

8. 界面设计的通用原则有哪些？常用的人机交互模式有哪些？