

产生软件工程的基因(1)计划方法,在预算内开发出高质量(2)软件开发
者无法制定具体计划,无法实现项目所需的功能,无法实现客户的需求
3)软件工程的重要性,在软件已有的工程:(1)软件开发具有复杂性和多变性(2)
软件产品必须适应着环境需求和国际环境的变化而变化
4)产品的质量将随着环境需求和国际环境的变化而变化
5)产品的质量将随着环境需求和国际环境的变化而变化
6)产品的质量将随着环境需求和国际环境的变化而变化

7)产品的质量将随着环境需求和国际环境的变化而变化
8)产品的质量将随着环境需求和国际环境的变化而变化
9)产品的质量将随着环境需求和国际环境的变化而变化
10)产品的质量将随着环境需求和国际环境的变化而变化

模块型降低成本高,灵活性强。

复杂性降低,分布式易表达。

2模型/视图/控制

在模型/视图/控制器(Model View/Controller, MVC)体系结构中,
子系统被分成三类模型子系统维护域的知识,视图子系统为用户提供
信息,控制器子系统管理与用户的交互顺序。模型子系统状态的改变由视
觉系统通过执行一个通知机制来传播。

优点:

职责分离,结构清晰易维护。

模型可复用,通用性强稳固。

组件解耦,扩展修改成本低。

支持并行开发,提升效率。

缺点:

学习成本高,需掌握组件交互。

小型项目冗余,增加复杂度。

数据传递繁琐,同步管理困难。

组件依赖性强,维护难度大。

3客户端服务器

在客户端服务器体系结构中,服务器子系统为客户端系统提供

共享资源,客户端负责交互,对服务器请求,往往通过远程过程调用机

制,客户对系统资源的访问机实现。客户端向一个或多个服务器请求服务,服务器

对客户提供一无所知的,客户端服务器体系结构风格非常适合管理大量

优点:

数据一致性。

子系统解耦,低耦合易维护。

适合复杂数据处理,逻辑集中。

新增子系统透明升级。

缺点:

集成困难。

集中式管理,响应速度慢。

集中式管理,高并发易挂机,下降。

单点故障风险高,需容灾机制。

日期:

年月日

第 页

第

页

数据的分布式系统

优点：

分布式部署，适合海量数据处理。

廉价分离，高性价比服务部署。

服务器可扩展，性能提升很快。

数据业务集中管理，确保一致性。

服务器易成瓶颈，数据均衡与容灾。

依赖网络延迟或故障影响响应性。

服务器故障更可能影响整个系统。

经济适用，降低硬件部署成本。

缺点：

对各体系统风格是否服务整体风格的一种演化，子系统

必须高度可以互相兼容，每个子系统可以请求服务或者提供服务。

的实体可以向其它实体求取分成向其它端提供服务。

缺点：

中心化，冗余点很多，容错性差。

资源浪费，节点消耗资源分散。

计算负载重，资源利用率高成本低。

应用态网络，脆弱性大。

心机调用，协议实现复杂(如共识算法)。

日期： 年 月 日

第 页

僵持状态，稳定性差。

权限管理难，易反客为主攻击。

统一设计实现成本高，可能增加设计成本。

5层

三层体系结构风格同3个层次组织于系统。

操作层 包括所有的与用户打交道的边界对象，像窗体、菜单网页等都包

含在内。

应用逻辑层 包括所有的控制和实体对象，实现处理规则检查和应用相

所需要的通知。

存储层 实现对持久性对象的存储检索和查询。

优点：

分层解耦，修改一层不易影响其他层。

业务逻辑集中，可维护性与扩展性好。

各层技术独立，满足不同开发需求。

应用层逻辑可复用，端复用。

缺点：

小型项目固守分层导致冗余，开发成本增加。

数据跨层传递有延迟，复杂业务调用效率低。

三层需独立部署，配置协调度高，问题定位困难，连环调试难度大。

6层

四层体系结构风格是一种特殊的三层体系结构，它的接口层(即上

日期： 年 月 日

第 页

price) 会议层、表示层 (Presentation Layer) 和表示数据层 (Presentation Layer)

Spatial

优点:

能根据应用需求灵活地处理逻辑。

服务端可直接获得资源，降低网络传输压力。

统一服务端渲染，进而降低跨域复杂度。

缺点:

层级增加，测试与部署复杂度上升。

数据链路冗长，可能影响响应延时。

导致外挂本机支持困难。

管道和过滤器:

管道和过滤器体系结构相对简单，子系统从输入处接收到

输出数据，并通过输出处将结果送到其它子系统。子系统被称为过滤器。

器。子系统之间的关联被称为管道。

优点:

过滤器相对独立，修改替换成本较低。

通用过滤器可复用，减少开发量。

操作灵活可重用，支持动态调整。

部分场景可并行处理提升效率。

缺点:

数据格式需严格统一， 进而成本高。

分析: 所传递消息延时，复杂数据处理差。

解决: 在设计，出外部存储支持上下文。

数据链路，使用排查与监控困难。

什么是异常，引起异常的原因有哪些？在C++中怎么捕获异常？

答: 异常是程序在运行过程中遇到错误情况，例如空指针引用。

数组越界访问或读取失败等。异常程序是将程序执行一些非正常命令，异常序出现的周围有关程序设计时出现的编程错误或运行时检测到的错误。

一般可以通过异常处理解决这些问题。一、将本地入侵系统作为程序，如病毒。对语言的异常处理有助于处理在程序运行期间发生的任何意外情况。异常处理功能使用 try...catch 语句，关键字来尝试执行可能的操作，然后在发生异常后在合理的情况下处理故障，以及在事后清除资源。

分析: 对对象模型进行封装以精简耦合和持久存储模式。

参考: 部门和课件 15.2.96 页

模型类描述这一种转换逻辑，说明同一类对象避免质量下降，而且能

完成最终实现的转换方法，包括:

优化类模型。其系统模型的性能需求，这也将减少关联。

“力图使类的连接增加冗余关联，以提高效率添加导致属性以减少对系统

的访问。

将关联映射到集合这一活动将关联映射到源代码结构，如同和同集

将一个类的映射到另一个类这一过程描述为反向的映射操作行为。

将其模型映射到存储模式，这一活动将类模型映射到一种存储模式，如

散列存储模式。

名:

对系统设计的编程环境

模型转换与重构

模型转换

通过调整类结构化来将模型使其更贴近业务逻辑完整性。

操作添加删除或重命名类属性关联(如将重复的公有属性会并为

独立attribute)

去除冗余模型复杂度。确保与需求一致(如将全局模型中的文化关系明确为

属性值)。

重构(源代码重构)

在不改变系统功能的前提下通过代码重构。

技术手段 / 方法将子类公有字段或方法移至父类，避免重叠代码(

将子类和父类的公共字段或方法移至父类，避免重叠代码(

1. 对于数据库作业务逻辑(即权限校验, 跟踪记录)。

逆向工程(代码模型)

逆向代码生成 UML 模型，恢复或验证设计(DSLA, 即类图生成类图, 提供属性和方法)。

3. 属性和方法

3. 关联的代理实现

引脚集成

更新 Tool 中自动化的集合。

2. 对于将模型转换到持久存储模式(以关系数据库为例)

1. 类与属性映射

类表每个类的列名和类名一致(如�名和类名一致)。

属性列属性名列名(如 Child Order 和 Child Order) 数据类型匹配数据库(如 justi

ng VARCHAR(100))。

主键自然主键, 使用业务层一属性(如 id)。

代理主键添加前缀(如 User_id)。

2. 关联的数据表实现

→ 1 / 1 多关联。

日期: 年 月 日 第 页

日期: 年 月 日 第 页

WHERE advertiser_id = 1;

多对多关联

关联表：创建独立表存储双向链接（如：order 和 product 会同时出现在同一张表中）

继承关系映射

继承模型

垂直映射

子类分别继承父类（如：order 和 product，共享主键并通过类

抽象区分（如：order 和 product 都实现 AbstractOrder）

水平映射将超类字段复制到子类表（如：product 包含 order 的所有字段）。

删除超类

3. 性能优化策略

访问路径优化减少关联遍历避免多层次嵌套查询，添加冗余关联（如在

cart 表中插入商品名称 name）

索引与排序对“多”端集合字段建立索引（如：Cart 表的 order_id 字段）

对索引

延迟加载与缓存递进创建高成本对象（如：在访问时加载 product 的

细描述）。

散列计算结果用私有字段缓存高成本计算值（如 total price 缓存 product

price 结果）。

三、关键原则总结

统一模型、代码数据命名统一（如：类名 Product 表名 product 字段名

product）

全局性向对象语言（如：C# 直接实现关联同构集合）。

日期： 年 月 日

第 页

日期： 年 月 日

第 页

复杂数据通过外键、关联表处理复杂关联语义需通过垂直 / 水平

映射转换。

可维护的数据模型转换和重构需保留业务逻辑，避免过度优化导致代码

难以理解。

持久化层分离时表与存储逻辑分离，通过 ORM 工具（如 Hibernate）实现。

如何根据关联对象，属性和方法？

1. 标识关联

关联是两个或多个对象之间的关系。标识关联通过访问或访问属性

描述对象间的关系（如：“用户拥有订单”），查询时可直接在对象中用关联

方（order）类包含 user 属性或在多对多关系或需求录额外信息时创

建中间模型（orderUser 表记录用户与订单的关系及购买时间）。避免冗余，

仅在关联属性上管理冗余，中间模型（如本张关联需细化为有效期）。

2. 标识对象

对存在程序中承担单一职责的实体，建议用名词或名词短语（如：order

、“order”）需具备唯一性（如：order 和 product 的 id），具体对象对应现

实实体（如用户、商品），抽象对象封装逻辑（如：order 为负责税费计算）。

3. 标识属性

状态信息命名使用名词或名词短语（如 status、totalPrice）。

不称属性（如：order）避免冗余（如：order.product_id，敏感

属性需隐藏（如：order.totalPrice）对复杂对象有唯一性指针，标记

属性时，应使属性名与对象一致，以提高语义清晰度。例如，订单的属性可能包括“状态”、“日

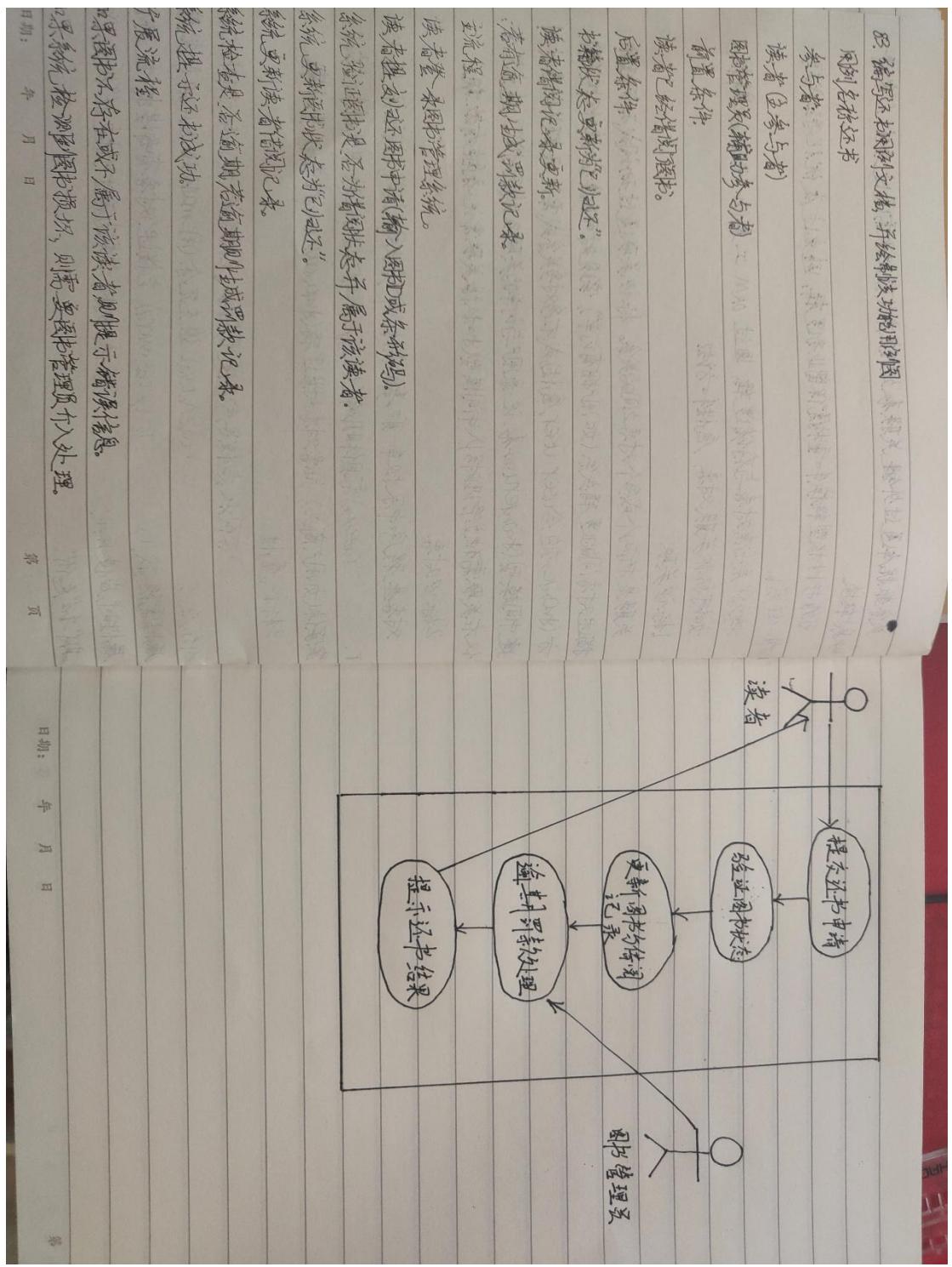
期”和“总价”。

日期： 年 月 日

第 页

日期： 年 月 日

第 页



第一部分(教材):

第五章

5-1 考虑一下带有图形用户界面的文件系统, 如Macintosh的Finder, Microsoft的Windows Explorer 和Linux的 KDE。从描述怎样从一个软盘复制一个文件到硬盘的同例中标识出了如下对象: File, Icon, Trashcan, Folder, Disk 和 pointer。

说明哪些对象是实体对象, 哪些对象是边界对象, 哪些对象是控制对象(p166)。

答: 实体对象: Folder, Disk, File

边界对象: Icon, pointer, Trashcan

控制对象: 本题没有控制对象。

5-2 假设如前所述的同一文件系统, 考虑一个包含了从软盘上选择文件并拖动该文件到Folder再释放鼠标的场景。标识和定义至少一个关联到此场景的控制对象(p166)。

控制对象的标识和定义如下:

标识: 文件选择器

定义: 文件选择器是一个用于选择文件并拖动到文件夹的交互组件。它可以显示软盘上的文件列表, 并提供可拖动的文件项。用户可以通过拖动文件项选择文件, 并将其释放到指定的文件夹中。

文件选择器的功能包括:

显示软盘中的文件列表

支持拖拽操作

实时显示拖拽过程中的视觉效果

鼠标释放时将选中的文件添加到文件夹中

在交互过程中，文件选择器可以提供相关的事件方法，方便开发者在需要的时候进行处理，例如选择文件、拖拽过程中
的回调、鼠标释放后的处理等。

通过使用文件选择器组件，用户可以方便地选择文件并
将其拖动到指定的文件夹中，实现了以硬盘选择文件并施
加到文件夹的场景需求。

解决方案选择器组件主要由三部分组成：一、标准文件选择器二、定义文件
选择器是一个用于选择文件并拖动到文件夹的交互组件。它
具备以下功能：
1. 显示硬盘中的文件列表；
2. 支持拖拽
操作，并实时察觉拖拽效果；
3. 鼠标释放时自动将所选
文件添加到目标文件夹。
三、事件处理在交互过程中，文件选
择器提供事件处理方法，使开发者根据实际需求进行定
制化操作，如选择文件、拖拽过程回调及鼠标释放后处理等。

四、应用如果通过该组件，用户能够便捷地完成以硬盘选
件并移至文件夹的操作，满足相关业务需求。

5.6 考虑图 5-32 所示的对称模型（选自 Jack son, 1995），给出
有关 `Calendar` 日历的知识，列出有关该模型的所有问题。修改该
模型的每一个问题（p166）

修改结果如图：

公案图 5-32 的问题与关联的多重性有关。

1. (Week) 周月份边界的问题。现实中，一周可能横跨两个(或
者更多)不同的月。但原模型通过“简单关联”(即从 1 月 2 日开始，到 2 月 3 日结束)。但原模型通过“简单关
联 month 和 Week”，错误地表示“周完全属于单个月份”。

2. 其他关联的多重性需明确规定。

年与月：一年严格包含 12 个月(原模型未显式约束)。

月与年：月份不会跨年(原模型未说明)。

周与日：一周固定为 7 天(原模型未强调)。

第六章

6. 将一个系统分解成多个子系统会降低复杂性，同时开发者
也通过简化各模块、增加这些模块的一致性来处理这一文

来的，但分解后会增加一些不同的复杂性：更小的模块

意味着更多的模块及接口。如果内聚度较低，将子系统分解成更小模块的耦合性原则，那么这是否能保持模块数目和比较小的竞争性原则是什么呢？(p111)

保持各模块数目和较小的竞争性原则可以被称为“一致性原则”或“简洁的原则”。这个原则鼓励开发者在系统设计追求简洁性，即尽量减少模块的总数。对低系统整体的复杂性、简约原则强调在系统分解时更清晰，避免过度分解导致过多的模块和接口，从而增加维护和理解的难度。通过保持一致性，开发者可以更容易地理解整个系统，并降低开发、测试和维护的复杂性。

6.2 我们将设计目标分成了三类：性能、可靠性和维护和最终用户。将一类或多类设计目标赋值以下例子。当用户发出任何命令后系统必须在一秒钟内将信息反馈给用户。即使在网络失败的情况下，火车票发行器Ticket Distributor 必须能够成功地接受火车票。

自动出纳机Automated Teller Machine必须能够根据字符修改火车票发行器Ticket Distributor 的房间必须考虑安装新的接线以防火警的数目有所上升。

自动出纳机Automated Teller Machine必须能够根据字符修改火车票发行器Ticket Distributor。

我们将采取更的子系统分解方案。(111)

答：应该使用桥接模式定义存储子系统的通用接口。因此，可以提供不同的存储子系统实现来处理特定的文件系统。通过桥接模式解决将文件从一个文件系统移动到另一个文件系统的问题。为了解决这个问题，需要开发一个单独的数据应用程 序。

6.3 在许多体系结构中，例如三层或四层体系结构，持久性对象的存储由专门的一层来处理。就你的观点而言，是哪些设计导致了这一决策？(p111)?

自动出纳机Automated Teller Machine必须能够根据字符修改火车票发行器Ticket Distributor。

答：安全性集中保护数据一致性，事务安全和完整性

一致性指标

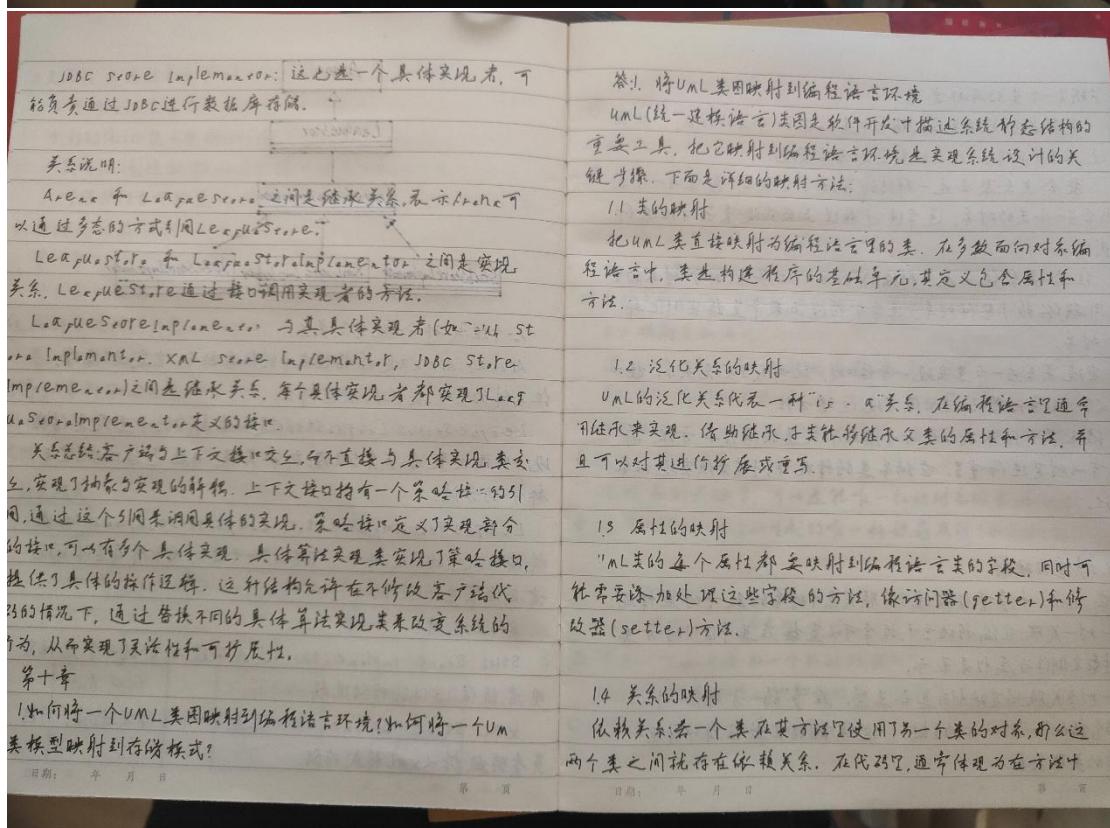
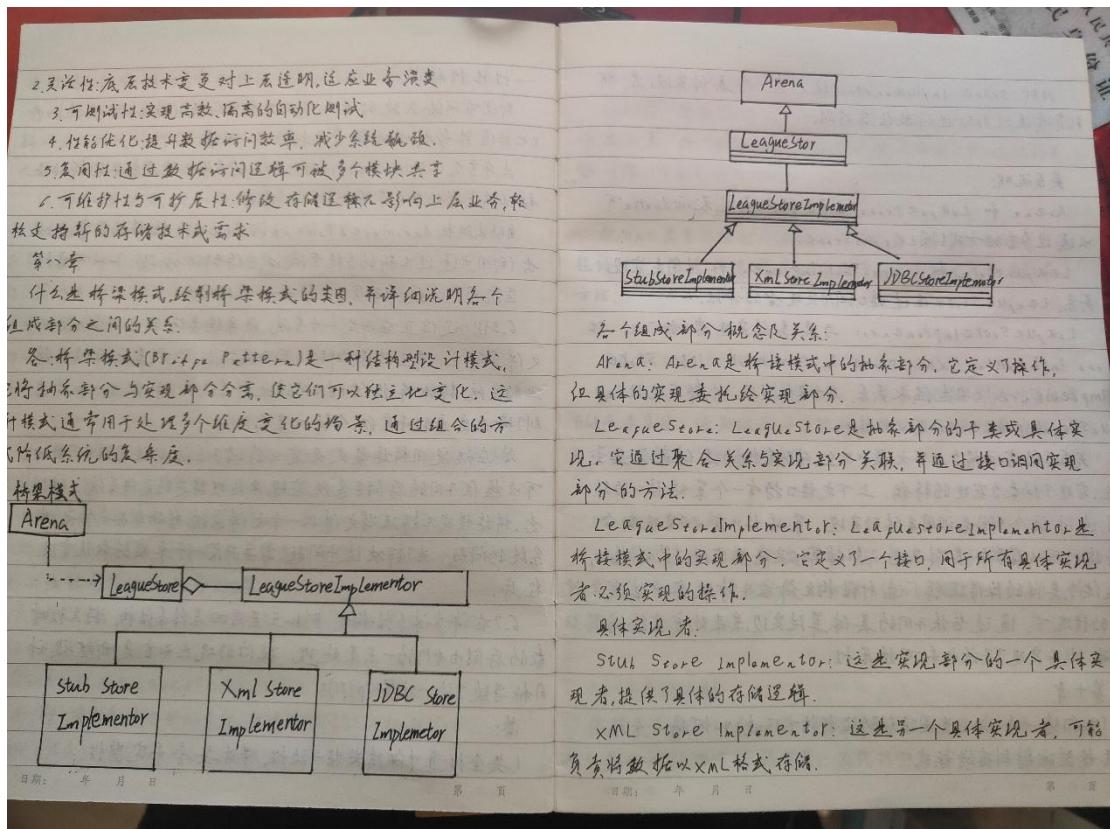
即使在网络失败的情况下，火车票发行器Ticket Distributor 也必须能够成功地接受火车票。——可靠性

火车票发行器Ticket Distributor 的房间必须考虑安装新的接线以防火警的数目有所上升。——维护

自动出纳机Automated Teller Machine必须能够根据字符修改火车票发行器Ticket Distributor。——可靠性

医院管理系统中的医生：护士——最终用户

6.3 你假设你正在开发一个系统，该系统要将数据存储在UNIX 上的系统新版本，提供对不同的文件系统支持。给出一个考虑



<p>声明另一个类的局部变量。</p> <p>关联关系: 关联关系表明一个类知道另一个类的存在，一般通过在类中声明另一个类的成员变量来实现。</p> <p>聚合关系: 聚合是一种弱的“has-a”关系，意味着一个类包含另一个类的对象，通常通过构造函数或设置器方法进行赋值。</p> <p>组合关系: 组合是一种强的“has-a”关系，部分对象的生命周期依赖于整体对象，通常在构造函数中直接实例化部分对象。</p> <p>实现关系: 当一个类实现一个接口时，就形成了实现关系。接口定义了一组方法签名，类必须实现这些方法。</p> <p>继承关系: 继承关系表示子类继承父类的属性和方法，并且可以对其实现重写。在抽象类的情况下，子类必须实现抽象方法。</p>	<p>多对多关联映射为关联表，通常会创建一个新的类来专门处理这种多对多的关系。实现为一个单独的表，该表包含两个外键列，分别指向关联的两个表的主键，通过这个中间表来建立多对多的关系，记录关联组合情况。</p> <p>2. 将UML类模型映射到存储模式:</p> <p>把UML类模型映射到存储模式（通常是关系数据库），能让你持久化存储。以下是具体的映射步骤：</p> <p>2.1 映射类和属性:</p> <ul style="list-style-type: none"> 每个UML类对应一个数据库表，表名与类名相同。 类的每个属性对应表的一列，列名与属性名相同。 为数据库列选择合适的数据类型，要根据属性的类型和取值范围来确定。 选择表的键字，可以选择唯一标识对象的类属性集合，也可以添加一个自动生成的唯一标识符属性（如自增主键）。 <p>2.2 映射关联:</p> <p>一对多关联和一对多关联使用外键实现所调的完全关联。在“多”的一方表中添加一个外键列，指向“一”的一方表的主键。</p> <p>多对多关联，实现为一个单独的表，该表包含两个外键列，分别指向关联的两个表的主键。</p>
--	---

<p>2.3 映射继承关系:</p> <p>关系数据库不直接支持继承，不过可以采用以下两种方案进行映射：</p> <ul style="list-style-type: none"> 垂直映射方案: 每个类用一个表表示，通过外键连接子类表和超类表。子类表包含子类特有的属性列，同时通过外键引用超类表的主键。 水平映射方案: 将起始属性推到子类中，本质上是复制与子类对应的表的列。每个子类表包含起始类和子类的所有属性列。 <p>2. 图示化管理系统的类图包括哪些属性和操作，绘制类图，并把它映射到C#、Java编程环境。</p> <p>答：借阅者类 (Borrower) 通常会包括以下属性和操作：</p> <p>属性:</p> <ul style="list-style-type: none"> Id: 借阅者的唯一标识符，例如借阅证号。 Name: 借阅者的姓名。 ContactInfo: 联系信息。 MembershipDate: 成为会员的日期。 BorrowRecordList: 当前借阅的书籍列表。 <p>操作:</p> <ul style="list-style-type: none"> BorrowBook(): 借书操作，从系统中借出一本书籍。 ReturnBook(): 还书操作，将书籍归还给图书馆。 RenewBook(): 延借操作，延长借阅期限。 	<p>GetBorrowerHistory(), 获取借阅历史记录。</p> <p>GetOverdueBooksList(), 获取逾期未还的书籍列表。</p> <p>类图:</p> <p>C# .NET 中的类定义</p> <pre>using System; using System.Collections.Generic; public class Borrower</pre> <p>日期: 年 月 日 第 页</p>
---	--

```
public int get: set; }  
public String Name { get: set; }  
public setting Configuration { get: set; }  
public Datetime MemberSopDate { get: set; }  
public List<Book> bookborrowed { get: set; }
```

```
public Borrower()  
{  
    Book borrowed = new List<Book>();  
}
```

```
1. 内容耦合 (Content Coupling)
```

```
定义：一个模块直接访问或修改另一个模块的内部数
```

```
据（如私有变量、代码逻辑），或控制另一个模块的执行流程（
```

```
如修改至其他模块的代码行）。
```

```
特点：最严格的依赖，模块间完全“绑定”。
```

```
2. 公共耦合 (Common Coupling)
```

```
定义：多个模块共享一个全局数据区（如全局变量、共享内存、
```

```
公共数据字典），通过修改全局数据间接交互。
```

```
特点：全局数据的修改可能影响所有依赖它的模块，难以
```

```
追踪责任。
```

```
3. 外部耦合 (External Coupling)
```

```
定义：模块间通过外部环境（如接口协议、文件格式）进行交
```

```
互，但依赖的是些标准化的外部约束（如指定的输入输出格式、
```

```
硬编码的接口参数）。
```

```
特点：依赖外部环境的具体实现，而并非抽象接口。
```

```
4. 控制耦合 (Control Coupling)
```

```
定义：一个模块向另一个模块传递控制信息（如标志位、状态
```

1

// 实现读写逻辑
// 将上层模块 A 的行为被模块 B 的控制参数直接干预，二者

1

public List<Book> getHistoryBooks() {

/// 实现获取历史记录逻辑
return new List<Book>(); // 示例返回值

1

public List<Book> getOralBooks() {

/// 实现获取语音书籍逻辑
return new List<Book>(); // 示例返回值

1

public void book

/// 定义：模块间仅通过管道简单数据(如控制信息)进行交互
(如共享类型、值载体、DTO 对象)。
特点：依赖仅基于数据本身，无逻辑或控制的干预，是理想

的低耦合“”。

6. 标记“” (Stamp, “” open, 或“” 故障结构耦合”)

定义：模块间传递一个复杂数据结构(如结构体、对象)，仅
仅变更一部分字段。
特点：此数据耦合略高(依赖数据结构的联合定义)，但
仍属于低耦合范畴。

边界用例

1) 边界系统与外部实体(如用户、其他系统)之间交互的用例。
这些用例通常涉及系统的输入和输出，以及系统如何处理这些
输入和输出。边界用例有助于确保系统在不同情况下的行为符
合预期，尤其是在处理异常和边缘情况时。

二部分：

漏洞分析

结合 (Cohesion) 是描述模块组织成系统之间相对
而言

方法是一种用于执行特定任务的代码块。方法由方法名
称、返回类型、参数列表和方法体组成。

日期： 年 月 日 第 页

第 页

内聚 (Cohesion) —— 模块内部功能相关性的度量
内聚是模块 (类、方法、子系统或组件) 内部各组成部分之间功能相关的紧密程度。高内聚意味着模块内部的元素高度相关，共同完成一个明确的任务；而低内聚则表明模块内部的元素关联性较弱，可能承担多个不相关的职责。

内聚性在软件设计质量的重要指标，高内聚的模块通常

具有以下优势：

可读性高、代码逻辑清晰，易于理解。
可维护性强，修改时影响范围小，不易引入错误。
可重用性好，模块化、单一，便于在不同场景复用。
耦合度低，模块间依赖关系合理，系统更健壮。

服务

服务是指由集成者组件通过明确规定接口提供得符合“高内聚低耦合”原则的可复用功能单元，在UML中表现得为接口定义，而不是实现通过接口来实现。在UML中，服务表示为接口定义和协作图中的交互；在实现层面，通常以继承或接口实现呈现。

软件体系结构

软件体系结构是指软件系统中不同组件之间的关系和交互方式，以及这些组件的组织结构和层次关系。它包括了软件系统的整体架构、模块化设计、数据流程、接口定义、组件之间的通信和协作方式等方面。

1. 模型 (Model)

负责应用程序的数据和业务逻辑，模型对系统应用程

序的数据结构，以及对数据的存取、修改和

处理。

2. 视图 (View): 负责在用户界面上显示数据或接受用户输入。

视图是模型的可视化呈现，它将数据呈现给用户，并捕捉用户的行为。

日期：年月日

第 页

日期：年月日

第 页

每一层还可以向后与其相邻下一层更低的层次。

划分

处理复杂性的一种方法是划分，将系统分解成对等的子系统，每个子系统负责一类不同的任务。《软件工程》中，“划分”可能涉及多种概念，以下是一些相关的名词解释：一、模块划分：将软件系统按照功能、性质等因素分解为若干个具有相对独立性、高内聚性和低耦合性的模块的过程。目的是使软件结构清晰，易于理解、开发和维护。二、功能划分：根据软件系统要实现的不同功能进行分类和划分，将相关功能组合在一起形成不同的功能模块，有助于明确软件各部分的职责。三、数据划分：把软件系统中的数据按照其特征、用途等进行分类，例如，将不同类型的数据存储在不同的数据仓库中，或者将数据分为输入数据、处理数据和输出数据等，便于数据的管理和处理。

四、测试用例划分：测试用例的范围和情况进行分类，把具有相似特征或需要验证相似功能的测试用例划分为一组，有助于提高测试效率和覆盖率，例如按照功能模块、输入数据类型等进行划分。

C/S

C/S Client/Server，客户端/服务器模式是一种分布计算架构，将系统功能划分为客户端和服务器两部分：

1. 客户端：用户直接交互的前端程序，运行于用户设备（如PC、手机），负责收集用户输入、展示界面，并向服务器发起请求。

日期： 年 月 日

兼职数据或执行特定操作（如查询数据、提交操作）。示例：电视上的微信客户端、steam游戏客户端。

服务器：提供服务、资源或数据的后端组件。它通常运行在网络上的一个或多个计算机上，等待来自客户端的请求，并对这些请求进行处理，负责处理客户端请求、存储数据、执行商业逻辑（如数据库计算、权限验证），并将结果返回给客户端。示例：微服务集群、银行交易处理服务器。

核心特点：

一、明确客户端应用体验，服务器专注数据与逻辑。

二、提升系统效率。

三、分离。

典型场景：企业ERP系统、大型游戏（如《魔兽世界》）、银行柜员系统等。

单元测试

在面向对象软件工程中，单元测试是指对软件中的最小可测试单元进行测试的过程，这最小可测试单元通常是一个类或方法，目的是确保每个单元在其功能上都是正确的，从而帮助提高整个软件系统的质量和稳定性。单元测试包括测试对象、测试目的、测试内容、测试方法。

属性

属性是指类的特征，一个类、实体或对象所具有的特性或状

日期： 年 月 日

第 页

类：属性通常用于描述类的特征，以及类与其他类之间的关系，以及对类的某一方面的信息，并且可以是任何类型的值，如整数、字符串、布尔值或其他对象。属性通常包括类的名称、类型、访问权限和默认值等信息。

实现关系

将一个类或其部分实现为另一个类或多个类的接口（可以是多个）的功能，实现类与接口之间最常见的是关系。在UML类图设计中，实现用一条带空心三角箭头的虚线表示，从实现指向实现的接口。

依赖关系
依赖关系是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、临时性的、非常规的，但是类B的变化会影响到类A。比如某人要过河，需要借同一条船，此时人与船之间的关系就生依赖。在UML类图设计中，依赖关系用由类A指向类B的带箭头虚线表示。

UML
类：UML是一种标准化的可视化建模语言，用于对软件系统进行描述、设计、构造和文档化。它通过图形化的方式表示系统的结构、行为、交互和架构，帮助开发团队和相关者更好地理解并沟通软件设计。
类的常见关联
继承关系（泛化关系）
继承指的是一个类（称为子类）继承另外的一个类（称为父类）的能力，并可以增加它自己的新功能的能力。在UML类图设计中，继承用一本带空心三角箭头的实线表示，从子类指向父类。
聚合关系

聚合是关联关系的一种特例，它体现的是整体与部分的关系，此时整体与部分之间是可分离的，它们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享。在UML类图设计中，聚合关系以空心菱形加实线表示。

组合关系

组合也是关联关系的一种特例，这种关系比聚合更强，也称为强聚合。它同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束，比如人和人的大脑。在UML类图设计中，组合关系以实心箭头加尖端表示。

UML类的每个属性的可访问性

public 公用的 + 前缀私有，该属性对所有类可见

protected 受保护的 # 前缀表示，对该类的子类可见

private 私有的 # 前缀表示，只对该类本身可见

packager 包用 ~ 前缀表示，只对同一包声明的其他类可见

私向工程

也指从高层次的设计或需求出发，通过系统化的手段逐步实

现最终产品或代码的过程。这是一种传统的、从抽象到具体的开

发方法，与逆向工程相反

正向工程的核心特点：

1. 从设计到实现

从需求分析、系统设计开始，逐步生成代码、测试和部署。

例句：

需求 框架设计 详细设计 编码 测试 支付

2. 遵循标准化流程

日期： 年 月 日

强调于结构化或面向对象的开发方法（如瀑布模型RUP等），强调文档和设计的完整性；

3. 工具支持

可能借助建模工具（如UML工具）自动生成代码框架（如通过类图生成 Java/C++代码）。

正向工程是“自顶向下”的传统开发路径，强调计划和设计先

（1. 连合需求明确的项目）

逆向工程

逆向工程是一种软件技术，它通过分析已有的软件系统，将

其内部结构、功能和设计等信息提取出来，转化为更容易理解的形

式（如代码到UML模型图），以帮助人们对软件进行维护、改进、复

用或研究。

边界对象

边界对象（boundary object）是一种设计概念，通常用于

该系统与外部环境（如用户、其他系统或外部设备）之间的交互

模块。它也面向对系统设计中的一个重要组成部分，主要用于对内

与外部世界的通信逻辑，将系统的内部实现与外部完全隔离，从而提高系统的可维护性和可扩展性。

实体对称

实体对称是用来描述具有持久性、唯一标识的事物的软件

构件，它是对现实世界中客观存在的、可区分的实体的抽象。

进阶，这些实体通常是在软件系统中需要被长期存储、管理和

更新，这些实体通常是在软件系统中需要被长期存储、管理和

操作的对象,即在学生管理系统的“学生”课程,在电商系统中的“商品”。

“整体”和“部分”都有其明确的阶次标记,通过这个标记可以在系统中唯一确定该实体对象。

组合

组合关系(Composition)是一种表示整体与部分强依赖的关联模式。其中部分对系完全归属于整体,对全局的整个严格绑定,不可独立存在,其核心特征是同时共死整体销毁时,部分随之销毁。在类图中组合关系通常使用实心菱形来表示。

聚集

“聚集”是一种强依赖的关联关系,用来表示“整体”与“部分”的关系。它描述了一个或多个其他对象(称为“部分”)组成的情况,但这些“部分”对象可以在不依赖于整个组合的情况下独立存在,换句话说,聚集展示了“整体”对象的所有权关系,使“整体”对象被销毁或删除,部分对象仍然可以继续存在。

拆分

拆分是指在一个较一般化的概念或设计基础上,划分为特定需求或场景对其进行调整、修改或拆分的过程。拆分可以使软件在不同情况下能够执行其他工作,并能提高代码的可重用性和灵活度。

关联

两个或者多个类的对象之间存在某种逻辑上的联系,彼此可以交互,但每个对象保持独立性。(P449)

c. 学生(Student)课程(Course)有关联,可以通过选择课程,课程可以

被学生选择。

作者(Author)和书籍(Book)有关联,作者可以写多本书,一本书只有一个作者。

控制關係

控制关系是一种特殊类型的联系,主要用于管理和协调其化对象们为以确保系统按照预定的规则和流程运行。

泛化

泛化是建模的关键活动,它体现了多个底层概念之间的共性,通过抽象出一个更高级更具代表性的概念涵盖这些底层概念。简单来说,就是把一系列具有共同特征的具体事物归为一个更普遍的类别。

方法

谷方法(面向对象方法)在面向对象编程中,方法是指与类或对象绑定的函数,用来定义该类型对象的行为特征。它通过对类特定操作的实现,遵循计算平均、验证输入等,使代码具备通用性、可重用性和结构化特征。例如,对某个类的“生成新字符串长度”的方法,与搜索字符串“这是一句方法”。
不同编程方法都有相应的语法规则,且通过参数传递和返回值等情况,实现对方法的调用。

数据抽象

共享聚集

共享聚集表示“整体”对象与“部分”对象之间的关系,其中“部分”可以属于“整体”之外的其他对象。在普通聚集(聚集关系),“部分”属于唯一的“整体”,而在共享聚集中,“部分”可以被多个“整体”共享。

表示方式

UML中的空心菱形箭头指向“整体”。

日期: 年 月 日

实体对象

表示系统将跟踪的持久信息。

实体对象是指在现实世界或软件系统所描述的领域中，具有独立存在意义和明确边界的事物。以下是其详细解释。

定义与特点：定义实体对象是对客观世界中实体的抽象表示，在软件系统中通常被建模为类的实例。特点：具有自己的属性和行为。属性用于描述实体对象的特征，比如一个“学生”实体对象，可能有姓名、年龄、学号等属性。行为则表示实体对象能够执行的操作，例如学生可以进行选课、考试等操作。

作用：数据存储与管理实体对象可以将相关的数据封装在一起，方便进行存储、检索和管理。例如，将客户的各种信息封装在“客户”实体对象中，便于在数据库中进行统一的存储和查询。业务逻辑可以在通过实体对象的行为来实现业务逻辑。比如在一个订单系统中，“订单”实体对象可能具有计算订单总价、添加商品到订单等行为，这些行为体现了订单处理的业务逻辑。

系统交互：实体对象是不同模块之间进行交互的重要载体，不同的模块可以通过操作实体对象来实现信息的传递和功能的协同。例如，在一个电商系统中，购物车模块和订单模块之间通过“订单”实体对象来传递商品信息和订单信息，实现从购物车到订单的流转。

持久性对象

持久性对象，持久性对象是指在软件工程中，那些被存储在持久化存储介质中，可以在程序运行结束后仍然访问和使用的对象。

UI (用户接口层)

答：UI (用户接口层) 是软件系统架构最外层，充当用户与系统交互的桥梁，它

日期： 年 月 日

通过按钮、菜单等界面元素展示系统功能与信息，收集用户的点击、输入等操作指令，并传递给业务逻辑层处理。同时将处理结果经可视化转换后反馈给用户。在技术实现上，Web 应用常使用 HTML、CSS、JavaScript，移动端和桌面应用则依赖平台选择对应开发语言与框架。其设计以用户体验为核心，旨在降低操作难度、提升使用效率。

DAL为数据访问层

数据访问层，它是一种软件架构模式，用于将数据访问逻辑与业务逻辑分离。DAL通常用于将数据访问逻辑从业务逻辑中分离出来，以便更好地管理和维护应用程序。DAL通常包括数据访问对象、数据访问接口和数据访问服务等组件。

MVC

MVC 是一种软件体系结构风格，把软件系统分为三个基本的部分：模型(Model)、视图(View) 和控制器(Controller)。

仓库体系结构风格

仓库体系结构风格 (Repository Architecture style) 是一种以数据为中心的体系结构范式，其核心思想是将数据存储在一个中央仓库（或称为中央数据结构），系统中的其他组件（处理模块）通过访问和操作该仓库中的数据来完成整体功能。这种风格适用于需要集中管理数据、支持多模块共享数据或基于数据进行复杂处理的场景。

顺序图

答：顺序图是统一建模语言 (UML) 中的一行动态交互图，用于描述系统在特定场景下对象之间基于时间顺序的交互行为，强调消息传递的时序关系和对象间

的工作过程。其核心元素包括对系统生命线(表示对象的存活周期)消息(如同步和返回消息)、发送者(表示执行操作的线程以及线程所属的线程组)、接收者(表示条件、循环等逻辑控制结构)、顺序通过(从抽象到具体)、时间流向(检测操作是否结束，直视或反向时间流)、协议(表示成对操作的参数)、状态(适用于分析运行行为，但面对复杂系统时可能对系统过多而无法会聚)、类型(表示操作元数据结构或流精信息)。

状态图

状态图是UML建模语言中的一种图形，用于描述单个对象在其生命周期中可能经历的所有状态。一般地，状态转换的事件条件是动作。它通过可观察的状态之间的转移连接，帮助理解对象在响应外部事件时的行为变化。

标识符

控制对象负责协调对系统实体对象。控制对象通常在一个区间内存在，并在该区间内随时终止。控制对象负责从边界对象处收集信息将这些信息分发给实体对象。

标准边界对象

标准边界对象对系统实体对象、控制对象通常在一个区间内存在，并在该区间内随时终止。控制对象负责从边界对象处收集信息将这些信息分发给实体对象。

日期： 年 月 日

④当许多参与者被包含在一个国际性时，必须需要考虑用直接来标记参与者的名字 (Participants names)。

⑤如果使用边界对象连接的可视方面(国产及类型是最好的选择)。

⑥总是使用最通用的术语描述接口及使用来自领域或实现域的术语。

安全性

安全性 (Security) 是确保软件系统免受恶意攻击及拒绝未经授权访问其化安。威胁的关键属性包括：以设计到部署的全生命周期防护措施、保护系统的机密性 (confidentiality) 完整性 (integrity) 和可用性 (availability) (PCA三元)。

可用性

一个系统，构件的在需要使用时可操作和响应的程度。指用户在使用系统时，系统能正常运行和满意度。重点关注用户体验 (以人和人机交互 (HMI)) (连性)。

健壮性

健壮性：指一个对于规范之外的输入情况的处理能力。

保密性

保密性是指确保软件系统中的敏感信息不被未经授权的个人或实体读取或泄露的能力。保密性是信息安全的一个重要方面，特别是在数据存储和传输过程中。

可移植性

可移植性 (Portability) 是指软件系统能够在不同的硬件平台上操作。运行环境或其它技术环境变更需进行适当的移植能力，而无需进行大规模修改或仅需少量调整。它是衡量软件质量的重要指标之一，直接影响

日期： 年 月 日

第 页

软件的通用性维护和生命周期管理。将平台移植到多种操作系

统 (如 Windows, Linux, OS X) 上运行。 2. 可模块化
设计减少对特定硬件或外部组件的依赖。

3. 低功耗设计通过模块化设计来降低功耗并延长平台使用寿命。

4. 依赖关系矩阵第二步完成工具在同时使用十分困难。

在此基础上增加对系统架构的分析，单独识别成功的地

点或失败点 (失败点往往与质量、可靠性、成本等直接相关)

传递数据的数量 (以比特率为衡量单位)。在阅读中，吞吐量是指在

没有被丢弃的情况下每秒能接受的最大速率。软件工程师强调的是系

统在单位时间内的工作能力

可靠性

能防止因概念设计和结构等方面不完备造成的设计系统失效，具

有挽回损失、减少维修费用的能力

易维护性

各软件系统的有效性体现在其能否以合理的时空资源开销准确实现用

户需求与业务目标，并在响应时间和操作效率、可维护性等维度达成质

量要求。在实践时，系统优化的优先级是提升可维护性的重中之重，但需与其它质量属性 (如安全性和用户体验) 进行权衡，确保最终产

品的软件价值可持续升值。

可延展性

可延展性是指软件系统能够快速灵活地适应环境变化，需求变更或

外部环境的变化，其本质是通过抽象解耦和模块化设计，将系统的可

靠部分与不稳定部分分离，从而降低变更成本，延长软件生命周期。它是微

服务的一个重要组成部分，尤其在复杂多变的业

务场景中至关重要。

可移植性

可移植性指的是软件能够适应不同的软硬件环境，易于移植和易于

修改的程度。它是指一个软件系统能够经历史演化和维护的难易程度

的一个重要指标。可移植性包括以下几个方面的可移植性：可读性、可扩

展性、低耦合性。

经济 (economics) 是项目管理中可被分配的经济和组织的最小工作单元。它代

表一项具体的、可衡量的活动，通常具有明确的输入输出时间的预算和资源需

求。

方法学是解决某一问题的方法集合，它规定了在何时何地以何种

方式使用。

用例图

答定义用来描述用户的需要，以用例的角度来描述系统的功能，并给出各

种行为的执行者，强调用例在使用系统，系统为执行者完成哪些功能，且系统

的蓝图。

类图

表示在某一时刻一组对象以及它们之间关系的图形。(一个对象是类图的一

个实例，由于对象存在生命周期，因此对系统的需求在系统某一时间段存在。）

化系统

软件工程中的化系统是指用来描述和表示软件工程中各种元素和关系的符号系统。常见的软件工程化系统包括：统一建模语言（UML）、类体系关系图、数据流图等。在软件工程中，化系统（Model System）用于可视化地描述已设计好的模型、流程结构和所选的符号体系、沟通和连接的关键工具。

化子

简单来说，“软件工程”是用系统设计的符号，就像“图形上的乐章”，此乐章或图形用逻辑推导出一个类，一个表示公司/客户、需求、类和类继承关系。这些符号以单一的方式将描述系统组织成一个整体。这种和沟通更直观，还能被工具自动识别并生成相应的结果。

需求

相关的任务被合到一起时，活动是更大的工作单元。活动的持续时间通常和某一个开发阶段相匹配，需求分析和系统设计。

需求

需求说明一个系统必须具有的性质，功能性需求是系统必须支持的功能说明，非功能需求是系统操作的一种约束，与系统的互动

元直接关系。

分析

软件工程中的分析是对软件系统的需求、功能、性质、数据以及相关组件等进行深入研究和理解的过程。

日期： 年 月 日

单元测试

单元测试从较小的模块开始，最后一直到整个应用程序。开发者首先应识别出他们自己的代码可能的类型和缺陷的方法，这被称为单元测试。单元测试是软件设计师的一个基本概念，它涉及到软件的最小可测试单元进行测试。这里的单元“通常指的是代码中的函数、方法或类等。在不同的编程语言和环境下单元的含义可能会有所不同。例如，在C语言中，一个函数可能被视作一个单元而在Java中，一个类可能被视作一个单元。对于图形化软件，一个窗体或一个菜单也可能被视作一个单元。

面向工程

面向工程（Object-oriented engineering）通过运用面向对象的方法论和模型驱动的思维，又称之为“面向对象设计”，是一种系统化的方法。用于从高层次的概念或需求出发，将其逐步精细化为具体的系统设计、实现和部署；在软件开发、硬件设计以及系统集成等多个领域均有广泛的应用。它强调从抽象到具体、从整体到部分的推导过程，确保最终的产品或系统能够满足既定的需求和目标。

PDN

$PDN \rightarrow Database$

PDN \rightarrow ODM

二综合题

什么是软件工程？软件工程的原理有哪些？

软件工程中的分析是指对软件系统的需求、功能、性质、数据以及相关组件等进行深入研究和理解的过程。

日期： 年 月 日