软工-全部实验要求与学习记录-协作平台

2023春 软件工程与实践

特别要求到保存到协作平台的内容（其他学习进度附后）

EXP12 论述利斯科夫替换原则（里氏代换原则）、单一职责原则、开闭原则、德（迪）米特法则、依赖倒...

EXP13 给出4种设计模式的例子，并总结其特点 （保存到每个小组选定的协作开发平台上）

EXP14 深入理解白盒测试和黑盒测试，总结其特点 （保存到每个小组选定的协作开发平台上，以组为单位）

下列是实验全部的内容汇总，含学习笔记、进度记录

实验1-2

CASE工具调研

实验3

小组分工讨论传统软件开发过程模型与敏捷开发（中几种主要方法）的比较，分析各自的优缺点，以及如何...

并且分析自己项目中可能存在的风险，细化风险管理（做出风险分级及应对预案）。

阅读Scrum开发方法文档，理解Scrum过程工作模型

实验4

阅读XP开发方法文档，理解XP过程工作模型

阅读DevOps文档，了解DevOps

实验5

活动图练习：（预习）书上练习题2,3（p97-98) 的软件开发项目活动图，找出关键路径

小组讨论，针对自己项目中的工作进行工作活动分解，分工进行各自合理的工作进度估算，最后汇总绘出项...

下图是一个软件开发项目的活动图，边长代表天数。请分析在图上标出每一个活动的最早开始时间、最晚开...

练习项目跟踪工具的使用，如用甘特图记录跟踪项目过程。

调研国内外软件开发团队组织结构和工作方式对比。

实验6

工作量估算

风险管理

实验7

参照课本及PPT上例子，练习用静态建模 (E-R、UML）等工具对所负责的系统建模，用模型model与用户沟...

实验8

阅读“SYSTEM MODELLING WITH PETRI NETS”，进一步学习Petri网知识，了解如何应用Petri网对系统进...

针对各自负责项目的不同场景，练习用各种动态建模工具 (状态图、Petri网、数据流图、OCL逻辑等）建模...

实验9

阅读“The Unified Modeling Language Reference Manual”，进一步学习UML知识，理解如何应用UML对...

- 统一建模语言UML
- UML的应用范围
- 初识UML
- UML构造快
  - 事物
  - 关系
  - 图
  - UML2
- UML通用机制
- 规格说明
- 修饰
- 通用划分
- UML扩展机制
- 用例图
  - 用例图的组成元素
- ER图
  - ER图的核心要素：
- 功能结构图
  - 定义
- 系统流程图
  - 说明
- 组织结构图
- 顺序图

浏览“LOGIC IN COMPUTER SCIENCE--Modelling and Reasoning about Systems”，了解常用逻辑及其在...

分工协作，学习、检索研究经典软件体系结构案例。

实验10-11

对比书上各种软件体系结构风格和视图特点

研究经典软件体系结构案例KWIC

实验12

结合项目的进程和开发历程，从设计原则的几个方面，思考软件开发存在的问题和解决方案。

学习依赖注入技术

论述利斯科夫替换原则（里氏代换原则）、单一职责原则、开闭原则、德（迪）米特法则、依赖倒转原则、...

实验13

结合项目的进程和开发历程，分析软件采用那些设计模式

给出4种设计模式的例子，并总结其特点 （保存到每个小组选定的协作开发平台上）

实验14

阅读下面软件测试相关资料（或查阅其它相关资料）Software Testing-A Research Travelogue (2000–201...

深入理解白盒测试和黑盒测试，总结其特点 （保存到每个小组选定的协作开发平台上，以组为单位）

阅读下面符号测试 （Symbolic Testing）相关资料（或查阅其它相关资料），了解符号测试的基本概念、主...

阅读下面差分测试 （Differential Testing）相关资料（或查阅其它相关资料），了解差分测试的基本原理、...

2023春 软件工程与实践

小组信息：

组名：嗯嗯对对

项目地址：https://github.com/kwinderic/BubblingMap

小组成员：

计机20.1	李正华	202000130109	kwinderic
计机20.1	褚瑞凡	202022180199	RefunRefun
计机20.1	杨绍清	202000171124	Andy-2035
计机20.2	刘芋彤	202000130008	SDUyutou
计机20.2	李泽恺	202000130069	crjg-k@qq.com

特别要求到保存到协作平台的内容（其他学习进度附后）

EXP12 论述利斯科夫替换原则（里氏代换原则）、单一职责原则、开闭原则、德（迪）米特法则、依赖倒转原则、合成复用原则，结合自己的实践项目举例说明如何应用 （保存到每个小组选定的协作开发平台上，以组为单位）。

里氏代换原则（Liskov Substitution Principle）强调子类对象必须能够替换其父类对象，而不影响程序的正确性和预期行为。

在我们的Bubbling活动管理系统中，有一个抽象的活动类（Activity），并且派生出了多个具体的活动类型类，如演讲活动（SpeechActivity）、体育活动（SportsActivity）和美食活动（FoodActivity）等。这些具体的活动类型类都继承自活动类，并实现了其特定的行为和属性。

在遵循里氏代换原则的情况下，可以在代码中使用活动类的实例来替换具体的活动类型类的实例，并保持系统的正确性和预期行为。例如，假设系统中有一个活动管理器（ActivityManager）类，负责管理所有的活动。可以使用活动类的实例来处理各种具体的活动类型，而不需要知道具体的活动类型。

单一职责原则（Single Responsibility Principle）强调一个类应该有且只有一个引起它变化的原因，即一个类应该只负责一个单一的职责或功能。当一个类承担了过多的职责时，它变得复杂、难以理解和维护，容易引发代码的脆弱性和不稳定性。

在活动管理系统中，可以通过单一职责原则来确保各个类具有清晰的职责和功能，以提高代码的可读性、可维护性和可测试性。以下是一些示例：

活动类（Activity）：

职责：封装活动的基本信息，如活动名称、时间、地点等。

不负责：处理活动的具体业务逻辑，如活动报名、签到等。

活动管理器类（ActivityManager）：

职责：管理活动的整体流程，包括活动的创建、编辑、删除等操作。

不负责：处理活动的具体业务逻辑，如报名和签到功能。

活动报名类（Registration）：

职责：处理活动的报名流程，验证报名信息、记录报名人数等。

不负责：处理其他与活动无关的功能，如签到和支付等。

通过将不同的功能划分到不同的类中，遵循单一职责原则，可以使得每个类专注于自己的职责，代码结构更加清晰。

开闭原则（Open-Closed Principle）是面向对象设计的基本原则之一，强调软件实体（类、模块、函数等）应该对扩展开放，对修改关闭。简单来说，它要求系统的设计应该是可扩展的，而不需要修改已有的代码。

在活动管理系统中，可以通过应用开闭原则来支持系统的扩展和变化，而不需要修改已有的代码。以下是一些示例：

活动类（Activity）：

开放性：定义了抽象的活动类，作为活动类型的基类，允许派生出不同具体的活动类型类。

封闭性：不需要修改活动类的代码，即可通过派生类添加新的活动类型。

活动管理器类（ActivityManager）：

开放性：通过接口或抽象类定义活动管理器的基本功能和方法，允许扩展具体的活动管理器子类。

封闭性：不需要修改活动管理器类的代码，即可通过新增子类扩展活动管理器的功能。

活动报名类（Registration）：

开放性：使用接口或抽象类定义活动报名的基本流程和方法，允许扩展具体的活动报名子类。

封闭性：不需要修改活动报名类的代码，即可通过新增子类扩展活动报名的功能。

通过遵循开闭原则，系统的扩展性得到了保证，当需要添加新的活动类型、新增活动管理功能或扩展活动报名流程时，可以通过扩展现有的抽象类或接口，并新增具体的子类来实现，而无需修改已有的代码。

迪米特法则（Law of Demeter）是面向对象设计的原则之一，强调一个对象应该尽可能少地了解其他对象的细节。简而言之，一个对象应该与其他对象保持最少的直接交互，而是通过尽可能少的方法调用来完成需要的功能。

在活动管理系统中，可以通过应用迪米特法则来降低对象之间的耦合度，提高系统的灵活性和可维护性。以下是一些示例：

活动类（Activity）：

活动类只与活动管理器类（ActivityManager）进行直接交互，通过提供必要的接口方法来向活动管理器报告自身的状态或获取必要的信息。

活动管理器类（ActivityManager）：

活动管理器类作为系统的核心组件，负责管理活动的整体流程。它不需要了解每个具体活动的细节，只需要与活动类进行交互，通过活动类提供的接口方法进行必要的操作。

活动报名类（Registration）：

活动报名类通过活动管理器类进行与活动的交互，不直接与具体的活动类进行交互。它向活动管理器报名人数、验证报名信息等，而不需要知道具体活动的实现细节。

通过遵循迪米特法则，对象之间的耦合度降低，对象之间的关系更加松散。每个对象只需要与少数相关的对象进行交互，不需要了解其他对象的内部细节。

依赖倒转原则（Dependency Inversion Principle）是面向对象设计的原则之一，强调高层模块不应该依赖低层模块的具体实现细节，而应该依赖于抽象。简单来说，高层模块和低层模块都应该依赖于抽象，而不是依赖于具体的实现。

在活动管理系统中，可以通过应用依赖倒转原则来降低模块之间的耦合度，提高系统的灵活性和可维护性。以下是一些示例：

活动类（Activity）：

活动类定义了一个抽象的活动接口，高层模块（如活动管理器）依赖于该抽象接口，而不是依赖于具体的活动类实现。这样，具体的活动类可以通过实现该接口来进行扩展。

活动管理器类（ActivityManager）：

活动管理器类依赖于活动接口，通过依赖注入的方式，将具体的活动对象传递给活动管理器。这样，活动管理器不需要知道具体的活动类型，只需要通过活动接口来调用活动的方法。

活动报名类（Registration）：

活动报名类同样依赖于活动接口，通过依赖注入的方式，将具体的活动对象传递给活动报名类。这样，活动报名类不需要知道具体的活动类型，只需要通过活动接口来进行报名操作。

通过遵循依赖倒转原则，模块之间的依赖关系更加松散，高层模块和低层模块都依赖于抽象接口，而不直接依赖于具体的实现。

合成复用原则（Composite/Aggregate Reuse Principle）是面向对象设计的原则之一，强调通过组合（合成）关系来实现对象的复用，而不是通过继承。简而言之，合成复用原则鼓励使用对象组合来构建更复杂的对象，而不是通过继承来获得复用。

在活动管理系统中，可以通过应用合成复用原则来实现对象的复用和组合，提高系统的灵活性和可维护性。以下是一些示例：

活动管理器类（ActivityManager）：

活动管理器类可以使用合成复用原则来组合不同的功能组件，例如地图组件、活动报名组件、支付组件等，以实现更复杂的活动管理功能。

通过将这些组件作为活动管理器的成员变量，可以在运行时灵活地配置和替换不同的组件实现，从而实现功能的复用和扩展。

活动报名类（Registration）：

活动报名类可以使用合成复用原则来组合不同的验证组件、通知组件等，以实现灵活的报名流程。

通过将这些组件作为活动报名类的成员变量，并通过接口定义它们的行为，可以在运行时灵活地配置和替换不同的组件实现，以满足不同的需求。

通过遵循合成复用原则，可以将系统的功能划分为不同的组件，通过组合这些组件来实现更复杂的功能，而不是通过继承来获得复用。

EXP13 给出4种设计模式的例子，并总结其特点 （保存到每个小组选定的协作开发平台上）

创建型模式 – 工厂模式（Factory Pattern）：

例子：活动管理系统中的活动工厂，根据不同的活动类型创建相应的活动对象。

特点：封装对象的创建过程，通过一个共同的接口来创建对象的实例。提供了一种统一的方式来创建对象，避免了直接依赖具体类的问题，提高了代码的可扩展性和可维护性。

结构型模式 – 适配器模式（Adapter Pattern）：

例子：地图接口适配器，将不同地图服务提供商的接口适配为统一的地图接口。

特点：将一个类的接口转换为客户端所期望的另一个接口，使得原本不兼容的类能够协同工作。通过适配器，客户端可以统一调用适配后的接口，实现了类之间的解耦和互操作。

行为型模式 – 观察者模式（Observer Pattern）：

例子：活动报名组件注册为活动管理器的观察者，当活动报名人数变化时，更新界面显示。

特点：定义了对象之间的一对多依赖关系，当一个对象的状态发生变化时，其所有依赖对象将自动收到通知并作出相应的更新。实现了解耦和动态的对象间通信。

并发型模式 – 单例模式（Singleton Pattern）：

例子：数据库连接池，保证系统中只有一个数据库连接池的实例。

特点：确保一个类只有一个实例，并提供全局访问点来获取该实例。通过单例模式，可以实现对资源的统一管理和控制，确保线程安全和节省系统资源。

EXP14 深入理解白盒测试和黑盒测试，总结其特点 （保存到每个小组选定的协作开发平台上，以组为单位）

WhiteBox.pdf

BlackBox.pdf

白盒测试：

特点：白盒测试是基于内部结构和逻辑的测试方法，也称为结构化测试或透明盒测试。

目标：白盒测试关注于检查和评估软件系统的内部工作方式和实现细节，以发现可能存在的逻辑错误、路径覆盖不足、代码漏洞等问题。

实施：测试人员需要了解被测试软件的内部结构、算法和代码逻辑，使用这些知识来设计测试用例和执行测试。

方法：白盒测试使用各种技术，如语句覆盖、分支覆盖、路径覆盖、条件覆盖等，以确保测试用例能够覆盖系统中的各种执行路径和代码段。

黑盒测试：

特点：黑盒测试是基于系统功能和需求的测试方法，也称为功能性测试或不透明盒测试。

目标：黑盒测试关注于测试软件系统的功能是否符合规格和预期行为，而不关心其内部实现和结构。

实施：测试人员不需要了解被测试软件的内部结构，只需要根据需求和规格文档设计测试用例和执行测试。

方法：黑盒测试使用各种技术，如等价类划分、边界值分析、决策表、状态转换等，以确保测试用例能够覆盖不同的输入组合和系统行为。

小结

白盒测试关注于软件的内部结构和实现，以发现逻辑错误和代码漏洞。

黑盒测试关注于软件的功能和需求，以验证系统是否按照规格和预期行为进行操作。

白盒测试需要了解内部结构和逻辑，使用结构相关的覆盖准则进行测试设计。

黑盒测试不需要了解内部结构，根据需求和规格进行测试用例设计。

两种测试方法可以互补使用，以达到更全面和有效的测试覆盖。

下列是实验全部的内容汇总，含学习笔记、进度记录

实验1-2

CASE工具调研

本项目使用的CASE工具：

图标工具：draw.io、ProcessOn，用于流程图、项目活动图的创建和绘制；

文档工具：

l Microsoft Word，用于开发文档的撰写与共享；

l Microsoft Excel，用于构建甘特图，追踪项目进度；

配置管理工具：Git，用于项目版本管理和变更控制；

编程工具：Android Studio、IntelliJ IDEA，用于项目开发；

CASE工具类型

图标工具

这些工具用于以图形形式表示各种软件组件和系统结构之间的系统组件、数据和控制流。例如，用于创建流程图的流程图制作工具。

过程建模工具

过程建是创建软件过程模型的方法，用于软件开发。过程建模工具帮助管理者根据软件产品的需求选择或修改过程模型。例如，EPF Composer。

项目管理工具

这些工具用于项目计划、成本和工作量估算、项目调度和资源规划。管理者必须严格遵守软件项目中提到的每一个步骤。项目管理工具有助于在整个组织内实时存储和共享项目信息。例如，Creative Pro Office, Trac 项目, Basecamp。

文档工具

软件项目中的文档在软件过程之前启动，贯穿于SDLC的所有阶段和项目完成后。

文档工具为技术用户和最终用户生成文档。技术用户主要是开发团队的内部专业人员，他们参考系统手册，参考手册，培训手册，安装手册等。最终用户文档描述了系统的功能和操作方法，如用户手册。例如, Doxygen, DrExplain, Adobe RoboHelp 提供文档。

分析工具

这些工具有助于收集需求，自动检查图表中是否有任何不一致，不准确，数据冗余或错误遗漏。例如, 接受360, Accompa, CaseComplete进行需求分析, 可视分析师进行总体分析。

设计工具

这些工具帮助软件设计人员设计软件的块结构，这些块结构可以使用细化技术进一步分解为更小的模块。这些工具提供了每个模块的详细信息以及模块之间的互联。例如，动画软件设计。

配置管理工具

软件的一个实例在一个版本下发布。配置管理工具处理：

- 版本和修订管理
- 基线配置管理
- 变更控制管理

CASE工具通过自动跟踪，版本管理和发布管理在这方面提供帮助。例如, Fossil, Git, Accu REV。

变更控制工具

这些工具被认为是配置管理工具的一部分。他们处理基线固定后或软件首次发布时对软件所做的更改。 CASE工具自动更改跟踪，文件管理，代码管理等。它还有助于在实施组织的变更策略。

编程工具

这些工具包括编程环境，如IDE（集成开发环境），内置模块库和仿真工具。这些工具为构建软件产品提供了全面的帮助，其中包括模拟和测试的功能。例如, Cscope to search code in C, Eclipse。

原型开发工具

软件原型是预期软件产品的模拟版本。原型提供了产品的初始外观，并模拟产品的几个方面。原型CASE工具基本上是和图形库一起提供。他们可以创建独立于硬件的用户界面和设计。这些工具可以帮助我们根据现有的信息来构建快速原型。此外，他们还提供了软件原型的仿真。例如 Serenaprototype composer, Mockup Builder。

Web开发工具

这些工具帮助设计包含相关元素（如表单、文本，脚本，图形等）的网页。Web工具还提供了对正在开发以及完成后的效果的实时预览。例如, Fontello, Adobe Edge Inspect, Foundation 3, Brackets。

质量保证工具

软件组织中的质量保证是监控开发软件产品所采用的工程过程和方法，以确保质量符合组织标准。QA工具包括配置和变更控制工具以及软件测试工具。例如, SoapTest, AppsWatch, JMeter。

维护工具

软件维护包括软件产品交付后的修改。自动日志记录和错误报告技术，自动错误记录单生成和根本原因分析的几个CASE工具，它们可以在SDLC的维护阶段帮助软件组织。例如, Bugzilla用于缺陷跟踪，HP质量中心。

实验3

小组分工讨论传统软件开发过程模型与敏捷开发（中几种主要方法）的比较，分析各自的优缺点，以及如何应用于自己的项目中?

传统软件开发模型：

(1) **瀑布模型**：

在瀑布模型中软件开发的各项活动严格按照线性方式进行，当前活动接受上一活动的工作结果，实施完成所需的工作内容，当前活动的工作结果需要进行验证，如验证通过，则该结果作为下一项活动的输入，继续进行下一项活动，否则返回修改，所以说其本质是线性顺序模型。

> 优点：

1)保证质量，在每个阶段都要完成规定的文档，每个阶段都要对已完成的文档进行复审。

2) 严格遵循预先计划的步骤顺序进行，一切按部就班比较严谨。

> 缺点：

1)各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量；

2)由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险；

3)早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。

(2) **原型化模型**：

原型化模型允许开发人员快速构造整个系统或系统的一部分以理解或澄清问题，需要对需求或设计进行反复调查，以确保开发人员、用户和客户对需要什么和提交什么有一个共同的理解。

> 优点：

1)从认知论的角度看，原型方法遵循了人们认识事物的规律，因而更容易为人们所普遍接受；

2)原型方法将模拟的手段引入分析的初期阶段，沟通了人们的思想，缩短了用户和开发人员之间的距离。

> 缺点：

1)对于一个大型系统，如果不经过系统分析得到系统的整体划分，而直接用原型来模拟是很困难的；

2)对于原有应用的业务流程、信息流程混乱的情况，原型构造与使用有一定的困难；

3)对于一个批处理系统，由于大部分活动是内部处理的，因此应用原型方法会有一定的困难；

4)文档容易被忽略；

5)项目难以规划和管理。

(3) **增量模型**

在增量开发中，需求文档中指定的系统按功能划分为子系统。定义发布时首先定义一个小的功能子系统，然后在每一个新的发布中增加新功能。软件开发的不同阶段是按软件产品所具有的功能划分，先开发主要功能或用户最需要的功能，然后，随着时间推进，不断增加新的辅助功能或次要功能，最终开发出一个强大的、功能完善的、高质量的、稳定的产品。

> 优点：

1)有利于增加客户对系统的信心

2)降低系统失败风险

3)提高系统可靠性，稳定性和可维护性

> 缺点：

1)增量粒度难以选择

2)把每个新的增量构建集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品

3)容易退化为边做边改模型，从而是软件过程的控制失去整体性

(4) **螺旋模型**

螺旋模型，它将瀑布模型和快速原型模型结合起来，强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。其核心在于不需要在刚开始的时候就把所有事情都定义的清清楚楚，而是在定义最重要的功能时去实现它,然后听取客户的意见,之后再进入到下一个阶段.如此不断轮回重复,直到得到你满意的最终产品。

> 优点：

1)支持用户需求的动态变化, 具有良好的可扩充性和可修改性。也支持软件系统的可维护性，每次维护过程只是沿螺旋模型继续多走一两个周期

2)原型易于用户和开发人员共同理解需求，还可作为继续开发的基础，并为用户参与所有关键决策提供了方便。

3)螺旋模型为项目管理人员及时调整管理决策提供了方便，进而可降低开发风险

> 缺点：

1)很难让用户确信这种演化方法的结果是可以控制的

2)建设周期长，而软件技术发展比较快，所以经常出现软件开发完毕后，和当前的技术水平有了较大的差距，无法满足当前用户需求

敏捷开发主要方法：

(1) XP

XP（极限编程）的思想源自 Kent Beck和Ward Cunningham在软件项目中的合作经历。XP注重的核心是沟通、简明、反馈和勇气。因为知道计划永远赶不上变化，XP无需开发人员在软件开始初期做出很多的文档。XP提倡测试先行，为了将以后出现bug的几率降到最低。

(2) SCRUM

SCRUM是一种迭代的增量化过程，用于产品开发或工作管理。它是一种可以集合各种开发实践的经验化过程框架。SCRUM中发布产品的重要性高于一切。

该方法由Ken Schwaber和 Jeff Sutherland 提出，旨在寻求充分发挥面向对象和构件技术的开发方法，是对迭代式面向对象方法的改进。

(3) Crystal Methods

Crystal Methods（水晶方法族）由Alistair Cockburn在20实际90年代末提出。之所以是个系列，是因为他相信不同类型的项目需要不同的方法。虽然水晶系列不如XP那样的产出效率，但会有更多的人能够接受并遵循它。

(4) FDD

FDD （Feature-Driven Development，特性驱动开发）由Peter Coad、Jeff de Luca 、Eric Lefebvre共同开发，是一套针对中小型软件开发项目的开发模式。此外，FDD是一个模型驱动的快速迭代开发过程，它强调的是简化、实用、易于被开发团队接受，适用于需求经常变动的项目。

(5) ASD

ASD（Adaptive Software Development，自适应软件开发）由Jim Highsmith在1999年正式提出。ASD强调开发方法的适应性（Adaptive），这一思想来源于复杂系统的混沌理论。ASD不象其他方法那样有很多具体的实践做法，它更侧重为ASD的重要性提供最根本的基础，并从更高的组织和管理层次来阐述开发方法为什么要具备适应性。

(6) DSDM

DSDM（动态系统开发方法）是众多敏捷开发方法中的一种，它倡导以业务为核心，快速而有效地进行系统开发。实践证明DSDM是成功的敏捷开发方法之一。在英国，由于其在各种规模的软件组织中的成功，它已成为应用最为广泛的快速应用开发方法。

DSDM不但遵循了敏捷方法的原理，而且也适合那些成熟的传统开发方法有坚实基础的软件组织。

(7) 轻量型RUP

RUP其实是个过程的框架，它可以包容许多不同类型的过程，Craig Larman 极力主张以敏捷型方式来使用RUP。他的观点是:目前如此众多的努力以推进敏捷型方法，只不过是在接受能被视为 RUP 的主流OO开发方法而已。

对比分析

1、相对于瀑布开发模型

敏捷开发区别于瀑布式的特征很明显，敏捷开发是以一种迭代的方式推进的，而瀑布模型是最典型的预见性的方法，严格遵循遇见计划的步骤顺序进行，步骤成为衡量进度的方法。敏捷开发过程中，软件一直处于可使用状态，它将项目分成若干相互联系、可以独立运行的子程序，因此，每个阶段软件都是可见的。

2、相对于增量式开发

两者的相同之处在于都强调在较短的开发周期提交软件。不同之处在于敏捷开发的周期可能更短，并且更加强调队伍中的高度协作。需求时刻在变，人们对于需求的理解也时刻在变，项目环境也在不停的变化，因此开发方法必须要能反映这种现实，敏捷开发方法就是属于这种适应性的开发方法，而非预设性。

另外重视交互也是一明显区别，敏捷开发更适用于小团队，比如在一个办公室工作，这样更有利于团队成员之间的交互，它也强调用户要与开发团队在一起工作，这样便于及时沟通交流，同时，“人与交互”也被列为敏捷开发价值观之首，可见其重要性。

3、相对于螺旋模型

和螺旋开发方法相比，敏捷方法强调适应性而非预见性。“螺旋模型”的核心就在于不需要在刚开始的时候就把所有事情都定义的清清楚楚，只需要定义最重要的功能并且实现它，然后听取客户的意见，之后再进入到下一个阶段。如此不轮回重复，直到得到您满意的最终产品。螺旋开发很大程度上是一种风险驱动的方法体系，因为在每个阶段之前及经常发生的循环之前，都必须首先进行风险评估。而敏捷开发，针对软件开发过程中诸多的不可预见性，强调的是适应性，适应性的方法集中在快速适应现实的变化。当项目的需求发生了变化，团队应该迅速适应，这个团队可能很难确切描述未来将会如何变化。

将敏捷开发到我们的项目中，需要做到：

团队竞赛

第一点，也是最重要的一点，敏捷建模者总是积极的寻求协作，因为他们意识到他们不是万事通，他们需要不同的观点，这样才能做到最好。软件开发可不是游泳，单干是非常危险的。在敏捷的字典中没有“我”这个词。

畅所欲言

敏捷建模者都有良好的沟通技巧——他们能够表达出他们想法，能够倾听，能够主动获取反馈，并且能够把需要的写出来。

脚踏实地

敏捷建模者应当脚踏实地他们的精力都集中在满足用户的需求上，他们不会在模型上画蛇添足，即便那双足是多么的好看。他们满足于提供可能的方案中最简单的一种，当然，前提是要能够完成工作。

好奇

敏捷建模者乐衷于研究问题，解决问题。

凡是都问个为什么

敏捷建模者看问题从不会至于表面，而是会打破沙锅问到底。他们从不会就想当然的认为一个产品或一项技术和它们的广告上说的那样，他们会自己试一试。

实事求是

敏捷建模者都非常的谦逊，他们从不认为自己是个万事通，所以他们会在建立好模型之后，用代码来小心的证明模型的正确。

根据实验

敏捷建模者应当愿意尝试新的方法，例如一项新的（或是已有的）建模技术。一般而言，他们也会接受敏捷建模开发技术，必要时，为了验证想法，他们愿意同传统的思想做斗争，例如在一个项目中减少文档数量。

有纪律

要坚持不懈的遵循敏捷建模的实践。对你来说，你可能会在不经意间说，“加上这个功能吧！无伤大雅。”或是，“我比project stakeholder更了解。”在AM的道路上要想不偏离方向，是需要一定的纪律性的。

并且分析自己项目中可能存在的风险，细化风险管理（做出风险分级及应对预案）。

风险评估

由于内外环境的变化，公司在运营过程中会面临各种风险，如果处理不当，就可能使公司陷入困境，甚至会引发重大突发事件，导致危机的发生。因此，树立风险意识，完善风险管理机制，有效应对各种风险和突发的事件，以确保公司战略目标的实现，就成为公司管理工作的一项重要内容。风险的成因是错综复杂的，正确地识别风险、恰当地评估风险、有效地应对风险是风险管理的核心内容。

1、经营管理风险

企业经营风险管理是企业以合理的风险成本投人，通过对风险的确认、选择和控制，以期达到最大的经营安全度。本公司成立创办之初，主要面临的经营管理风险有：创业风险，现金风险，授权风险，筹资风险等，就狭义来说，公司的经营管理风险主要来自于本公司内部的经营与管理。

1.1 应对措施

首先，企业必须全面分析经营中可能存在的风险，并对风险进行评估容忍度的确认，建立规范的风险监控体系。其次，建立风险管理制度，从制度上控制和规范风险的发生。再次，建立风险应急机制，一旦风险发生，应采取最有力的措施，将风险控制在最低限度内，减少损失，保证企业的根本利益。

2、市场风险

市场风险是指由于市场及相关的外部环境的不确定性而导致客户流失、品牌受损、市场萎缩、达不到链接预期效果乃至影响公司生存与发展的一种可能性。市场风险可导致企业投资活动失败，引发投资风险等一系列问题。消费需求变动、竞争对手行为、信息不确定或不对称等可能引发市场风险。除此以外，也可能是通过对其竞争者、供应商或者消费者间接对企业产生影响。本公司主要分析了当前我们所面临的两个主要风险：竞争对手风险和需求变化风险。

2.1 竞争对手风险

考虑到我们公司的产品及服务主要针对电子游戏领域，利用设备发现和融合，让普通人随时随地就能便携地享受高质量的游戏体验及相关附属功能。就此而言，本公司的竞争对手主要来自于同行业提供相同服务的机构，作为一个新兴技术创新公司，我们的竞争对手也面临着传统电子游戏行业的冲击。

2.2 需求变化风险

目前大众对于游戏这种运动还是抱有积极的态度，即有着很大的市场需求。但是随着时间的推移，用户可能对于游戏失去兴趣，这也导致了这种需求产生了一定的变化。

2.3 应对措施

对于竞争对手，我们要时刻关注竞争对手的动态及不断变化的市场情况，及时变动完善营销策略，在不超过消费者承受能力的情况下，制定出最佳的方案。在产品变化需求方面，我们在产品进入前期需做好市场调研工作，全面了解消费者情况，如消费者的承受力等确定当前发展方向，选定一个目标市场为突破口，制定相应的营销策略，由此打开市场，然后再逐步推入整个市场领域。除此以外，公司还会加大宣传，在不断完善营销策略的基础上，加大宣传力度，要使广大的消费者从观念上接受我们的产品，提高我们的服务质量。

3、技术风险

技术风险的种类很多，其主要类型是技术不足风险、技术开发风险、技术保护风险等。技术风险主要来自于两方面：

一、技术创新所需要的相关技术不配套、不成熟，技术创新所需要的相应设施、设备不够完善。由于这些因素的存在，影响到创新技术的适用性、先进性、完整性、可行性和可靠性，从而产生技术性风险。

二、对技术创新的市场预测不够充分。任何一项新技术、新产品最终都要接受市场的检验。如果不能对技术的市场适应性、先进性和收益性做出比较科学的预测，就使得创新的技术在初始阶段就存在风险。这种风险产生于技术本身，因而是技术风险。这种风险来自于新产品不一定被市场接受，或投放市场后被其他同类产品取代。就针对本公司的产品而言，主要存在以下几种风险：研发风险，被替代风险，被模仿风险。

3.1 研发风险

本公司的产品研发主要利用了设备发现和互联技术，主要包括视频的编解码和串流技术等。随着当今科学技术的发展，多设备融合技术也在不断进步，取得了一定的阶段性成果。正是在这个趋势之下，我们研发出来了该项产品服务，相比其他同行业的竞争者，也许设备融合已经得到了一定的运用，但是，如何让我们的产品服务满足用户的需求和使用习惯，这就是当下我们面临的研发风险。

3.2 模仿替代风险

如今市场上对于新产品的模仿现象依然存在。市场具有一定的复杂性，经常会出现竞争对手对新产品的仿造，而且仿造的产品由于在创新方面没有很大的投入，一般都比全新的新产品廉价，这使得本公司新产品的市场由于模仿产品的进入而出现竞争更加激烈的局面，这就容易让本公司获得较少的利润而不足以弥补开发投入。例如，某些生产商看到本产品服务的前景市场很好，就生产出类似的产品进行销售，这样就会为我公司的新产品带来一定的市场风险和竞争风险。

3.3 应对措施

消费者需求和欲望是开发新产品的起点，因此，公司在进行产品研发之前，必须对市场进行有效的调研分析，注重市场的调查及预测，在研发过程中不断对产品进行改良，通过对目标市场在产品的性能，体验感等方面的调查，准确把握消费者的偏好和需求，设计研发出更好的产品，从而形成产品优势。总之，企业要健康发展，在竞争中获得成功，占有有利的地位，必须不断创新，注重产品的研究和开发，不断推出满足消费者需求的产品与服务，从而实现企业的稳定增长。

4、政策风险

在市场经济条件下，由于受价值规律和竞争机制的影响，各企业争夺市场资源，都希望获得更大的活动自由，因而可能会触犯国家的有关政策，而国家政策又对企业的行为具有强制约束力。另外，国家在不同时期可以根据宏观环境的变化而改变政策，这必然会影响到企业的经济利益。因此，国家与企业之间由于政策的存在和调整，在经济利益上会产生矛盾，从而产生政策风险。

4.1 国家对相关服务业税收政策

在当今社会，国家对于服务行业采取优惠促进政策。对符合条件的小型微利服务业企业适用20%的优惠税率。按照我国社会经济的发展走势，未来国家政策对于服务类行业的支力度将会不断加大，对于服务企业也会给予一定的优惠政策。在这个宏观条件下，本公司作为新兴的小型服务企业，在一定程度上能够享受到国家政策的税收优惠待遇，但在某个特殊经济时段，也可能受到国家宏观政策的影响，因此，宏观政策风险对于本公司的可持续发展是不可避免的。

4.2 国家对新兴技术产业的支持政策变化

自2009年我国为应对国际金融危机的挑战、抢占全球新一轮技术革命的机遇以来，国家出台了一系列扶持战略性新兴产业发展的政策。国家把战略性新兴产业置于国家战略的高度，各地方也纷纷出台了支持新兴产业发展的决定，并在高新技术企业认定、促进科技成果转化和相关产业化政策中给予重点支持。在这样的宏观背景之下，作为本公司的新兴产业产品的“轻影”，深受国家政策的影响，政策风险依然存在。

4.3 应对措施

政策风险防范主要取决于市场参与者对国家宏观政策的理解和把握，取决于投资者对市场趋势的正确判断。由于政策风险防范的主要对象是政府管理当局，因而有其特殊性。企业在对政策风险进行管理时，首先要提高对政策风险的认识。对资产重组过程中面临的政策风险应及时地观察分析和研究，以提高对政策风险客观性和预见性的认识，充分掌握资产重组政策风险管理的主动权。其次要对政策风险进行预测和决策。为防止政策风险的发生，应事先确定资产重组的风险度，并对可能的损失有充分的估计，通过认真分析，及时发现潜在的政策风险并力求避免。对政策性风险管理应侧重于对潜在的政策风险因素进行分析，并采用科学的风险分析方法。通过对政策风险的有效管理，可以使企业避免或减少各种不必要的损失。

5、财务风险

企业的财务活动贯穿于生产经营的整个过程，财务风险是一种信号，通过它能够全面综合反映企业的经营状况，因此要求企业经营者要进行经常性财务分析，树立风险意识，建立有效的风险防范处理机制，加强企业财务风险控制，防范财务危机，建立预警分析指标体系，进行适当的财务风险决策。按财务活动的主要环节，可以分为流动性风险、信用风险、筹资风险、投资风险。按可控程度分类，可分为可控风险和不可控风险。

5.1 投资风险

投资决策对企业未来的发展起至关重要的作用，正确的投资决策可以降低企业风险，增加企业盈利;错误的投资决策可能会给企业带来灾难性的损失。错误的投资决策往往没有充分认识到投资的风险，同时对企业自身承受风险的能力预估有误。

5.2 资金短缺风险

资金短缺风险指企业不能及时足额地筹集到资金来满足生产经营的需要，从而导致企业不得不放弃供应商提供的优惠的现金折扣，低价甚至亏本出售存货和项目，不能及时清偿债务导致信用等级恶化，被迫破产重组或被收购等的风险。

5.3 应对措施

1、树立风险意识，建立有效的风险防范处理机制。具体方法主要有：一是坚持谨慎性原则，建立风险基金。即在损失发生以前以预提方式建立用于防范风险损失的专项预备金。二是建立企业资金使用效益监督制度。有关部门应定期对资产治理比率进行考核。同时，加强流动资金的投放和治理，提高流动资产的周转率，进而提高企业的变现能力，增加企业的短期偿债能力。另外，还要盘活存量资产，加快闲置设备的处理，将收回的资金偿还债务。

2、加强企业财务风险控制，回避风险，分散风险。

3、提高财务决策水平，建立财务预警系统。编制现金流量预算。

阅读Scrum开发方法文档，理解Scrum过程工作模型

概述

Scrum是跨职能团队以迭代、增量的方式开发产品或项目的一种开发框架。它把开发组织成被称为Sprint的工作周期。这些迭代每个都不超过4周（最常见的是两周），并且无间歇地相继进行。Sprint是受时间箱限制的，无论工作完成与否它们都会在特定日期结束,并且从不延长。通常由Scrum团队来选定一个 Sprint的时长，并且对于他们所有的Sprint都使用这一时长，直到这个团队能力提高，可以使用较短周期。在每个Sprint的初始，跨职能团队（大约7名成员）从排好优先级的列表中选择事项（客户需求）。团队对于在Sprint结尾他们相信自己可以交付哪些目标集合达成一致意见，这些交付应该是有形的并且能被真正“完成”的。在Sprint过程中不可以增加新事项，

Scrum在下一Sprint时才接受变化，当前这么短的一个Sprint周期里只注重于短小、清晰、相对固定的目标。团队每天都进行简短会面来检验工作进程，并调整后续步骤以确保完成剩余工作。在Sprint结尾，团队与利益关系人一起回顾这个Sprint,并演示所构建的产品。团队成员从中获取可以结合到下一Sprint中的反馈。Scrum强调在Sprint结尾产生真正“完成”了的可工作产品。在软件领域是指已经集成的、完全测试过的、已经为最终用户生成文档的、潜在可交付的系统。

Scrum中的一大主题就是“审视并调整”。因为开发工作不可避免地包含学习、创新和意外事件，Scrum强调进行小步骤开发，同时检验最终产品和当前实践的功效，然后调整产品目标与过程实践。周而复始。

Scrum开发流程中的三大角色

1、产品负责人（Product Owner）

主要负责确定产品的功能和达到要求的标准，指定软件的发布日期和交付的内容，同时有权力接受或拒绝开发团队的工作成果。

2、流程管理员（Scrum Master）

主要负责整个Scrum流程在项目中的顺利实施和进行，以及清除挡在客户和开发工作之间的沟通障碍，使得客户可以直接驱动开发。

3、开发团队（Scrum Team）

主要负责软件产品在Scrum规定流程下进行开发工作，人数控制在5~10人左右， 每个成员可能负责不同的技术方面，但要求每成员必须要有很强的自我管理能力，同时具有一定的表达能力；成员可以采用任何工作方式，只要能达到Sprint的 目标。

有关Scrum的几个名词

- backlog：可以预知的所有任务， 包括功能性的和非功能性的所有任务。
- sprint：一次跌代开发的时间周期，一般最多以30天为一个周期。在这段时间内，开发团队需要完成一个制定的backlog，并且最终成果是一个增量的，可以交付的产品。
- sprint backlog：一个sprint周期内所需要完成的任务。
- scrumMaster：负责监督整个Scrum进程，修订计划的一个团队成员。
- time-box：一个用于开会时间段。比如每个daily scrum meeting的time-box为15分钟。
- sprint planning meeting：在启动每个sprint前召开。一般为一天时间（8小时）。该会议需要制定的任务是：产品Owner和团队成员将backlog分解成小的功能模块，决定在即将进行的sprint里需要完成多少小功能模块，确定好这个Product Backlog的任务优先级。另外，该会议还需详细地讨论如何能够按照需求完成这些小功能模块。制定的这些模块的工作量以小时计算。
- Daily Scrum meeting：开发团队成员召开，一般为15分钟。每个开发成员需要向ScrumMaster汇报

三个项目：今天完成了什么？是否遇到了障碍？即将要做什么？通过该会议，团队成员可以相互了解项目进度。

- Sprint review meeting：在每个Sprint结束后，这个Team将这个Sprint的工作成果演示给Product Owner和其他相关的人员。一般该会议为 4 小时。
- Sprint retrospective meeting：对刚结束的Sprint进行总结。会议的参与人员为团队开发的内部人员。一般该会议为 3 小时。

Scrum的特点及使用流程

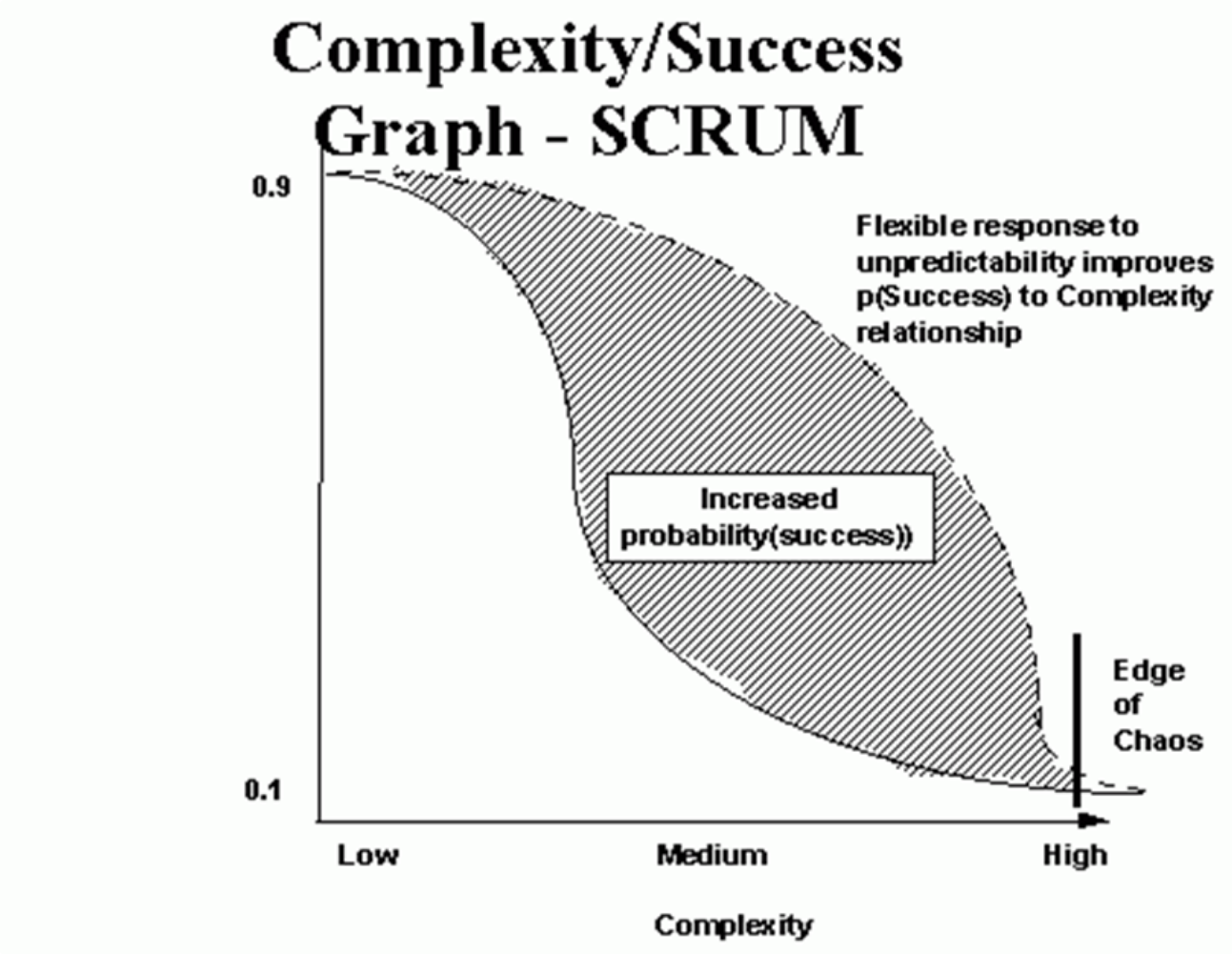
Scrum是一种敏捷开发的方式，它的特点是：灵活性、适应需求变化、更适合团队比较小的情况、每一个迭代均有产出、容易学习。

1) 将整个产品的backlog分解成Sprint Backlog,这个Sprint Backlog是按照目前的人力物力条件可以完成的。
2) 召开sprint planning meeting，划分，确定这个Sprint内需要完成的任务，标注任务的优先级并分配给每个成员。注意这里的任务是以小时计算的，并不是按人天计算。
3) 进入sprint开发周期，在这个周期内，每天需要召开Daily Scrum meeting。
4) 整个sprint周期结束，召开Sprint review meeting，将成果演示给Product Owner。
5) 团队成员最后召开Sprint retrospective meeting，总结问题和经验。
6) 这样周而复始，按照同样的步骤进行下一次Sprint。

Scrum较传统开发模型的优点

Scrum模型的一个显著特点就是响应变化，它能够尽快地响应变化。下面的图片使用传统的软件开发模型（瀑布模型、螺旋模型或迭代模型）。随着系统因素（内部和外部因素）的复杂度增加，项目成功的可能性就迅速降低。

下图是Scrum模型和传统模型的对比：



实验4

阅读XP开发方法文档，理解XP过程工作模型

XP（极限编程）是一种敏捷软件开发方法，强调团队合作、灵活性和快速迭代。

XP开发过程模型学习后的特征总结：

1. 用户故事（User Stories）：XP鼓励将软件需求表达为用户故事。用户故事是对用户需求的简短描述，包含标题、描述和验收标准。用户故事帮助团队理解用户期望，提供明确的目标和反馈机制。
2. 短期迭代（Short Iterations）：XP采用短期迭代的开发周期。通常，每个迭代持续一到两周。在每个迭代期间，团队成员共同努力完成一部分功能，并根据用户反馈进行迭代和改进。这种迭代的方式使得团队能够快速响应变化，并逐步构建出高质量的软件系统。
3. 测试驱动开发（Test-Driven Development，TDD）：XP倡导测试驱动开发。在编写代码之前，先编写自动化测试用例。测试用例定义了期望的功能和行为，开发人员通过编写代码使测试用例通过。TDD确保了代码的可测试性、可维护性和稳定性。

4. 集体所有权（Collective Ownership）：XP鼓励团队成员共同拥有代码的所有权。这意味着任何团队成员都可以修改和改进代码，从而促进知识共享和技术合作。集体所有权减少了对个人的依赖，提高了团队的灵活性和可靠性。
5. 持续集成（Continuous Integration）：XP强调持续集成的实践。团队成员频繁地将代码集成到共享的代码库中，并自动运行构建和测试过程。持续集成确保团队的代码始终保持一致且可部署，及时发现和解决集成问题。
6. 小型团队（Small Team）：XP鼓励小型团队合作。通常，一个XP团队由5到10名成员组成，包括开发人员、测试人员和客户代表。小型团队有助于提高沟通效率、团队凝聚力和协作性。
7. 持续改进（Continuous Improvement）：XP鼓励团队不断反思和改进工作方式。团队会定期举行回顾会议，回顾过去的工作，并识别出改进的机会。持续改进使团队能够逐步提高工作效率和软件质量。

阅读DevOps文档，了解DevOps

DevOps是一种软件开发和运维的方法论，旨在通过加强开发团队和运维团队之间的合作与沟通，实现软件交付的快速、高质量和可靠性。

DevOps学习特征概括：

1. 文化变革（Cultural Change）：DevOps强调团队间的协作和文化变革。开发和运维团队之间的壁垒被打破，取而代之的是合作、共享和共同承担责任的文化。团队成员需要培养相互信任、沟通和合作的习惯，以实现高效的软件交付和运维流程。
2. 自动化（Automation）：DevOps倡导自动化各个阶段的软件交付和运维过程。通过自动化，可以减少人工干预和错误，提高交付速度和质量。自动化包括构建、测试、部署、监控等方面，旨在实现可重复、可靠和可伸缩的软件交付流程。
3. 持续集成和持续交付（Continuous Integration and Continuous Delivery）：DevOps推崇持续集成和持续交付的实践。持续集成确保开发团队频繁地将代码集成到共享的代码库中，并通过自动化测试和构建验证代码的正确性。持续交付则通过自动化部署和发布，使得软件能够快速、可靠地交付给用户。
4. 基础设施即代码（Infrastructure as Code）：DevOps引入了基础设施即代码的概念，即将基础设施的配置和管理通过代码进行定义和自动化。通过基础设施即代码，可以实现基础设施的版本控制、自动化部署和可重复性，提高基础设施的可靠性和可管理性。
5. 频繁的反馈与改进（Frequent Feedback and Iteration）：DevOps鼓励频繁地获取用户和运维团队的反馈，并将反馈转化为改进的机会。通过持续监控和日志分析，可以及时发现和解决问题，提高系统的稳定性和性能。
6. 容器化和微服务架构（Containerization and Microservices Architecture）：DevOps推崇使用容器化技术（如Docker）和微服务架构来构建和管理应用程序。容器化提供了隔离、可移植和可扩展

的环境，使得应用程序能够在不同的环境中一致地运行。微服务架构将应用程序拆分为小型、自治的服务，提供灵活性和可伸缩性。

实验5

活动图练习：（预习）书上练习题2,3（p97–98）的软件开发项目活动图，找出关键路径

1. 习题2

节点	前驱
A	
B	A
C	A
D	A、B
E	A
F	A、C
G	A、E
H	A、C、E、F、G
I	A、B、D
J	A、B、D、E、G、I
K	A~J
L	A~K

活动	最早开始时间	最晚开始时间	时差
A–>B	1	1	0
B–>D	4	4	0
B–>I	4	5	1

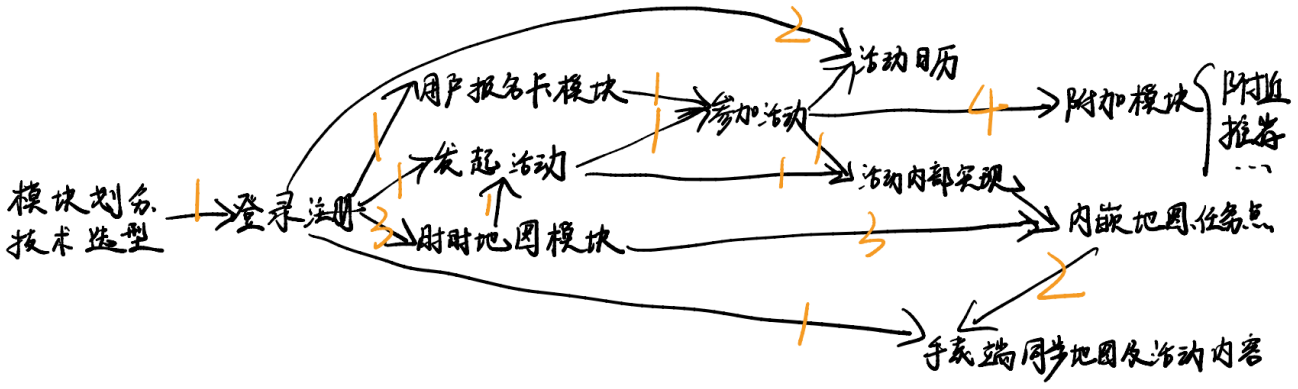
D–>I	9	9	0
I–>J	11	11	0
A–>C	1	5	4
C–>F	6	10	4
F–>H	9	13	4
A–>E	1	4	3
E–>G	5	8	3
G–>H	8	14	6
G–>J	8	11	3
H–>K	10	14	4
J–>K	13	16	3
J–>L	13	13	0
K–>L	15	18	3
L	20		

关键路径A–>B–>D–>I–>J–>L，共耗时20天。

2. 习题3

关键路径A–>B–>C–>E–>D–>I–>K–>L，共耗时24天。

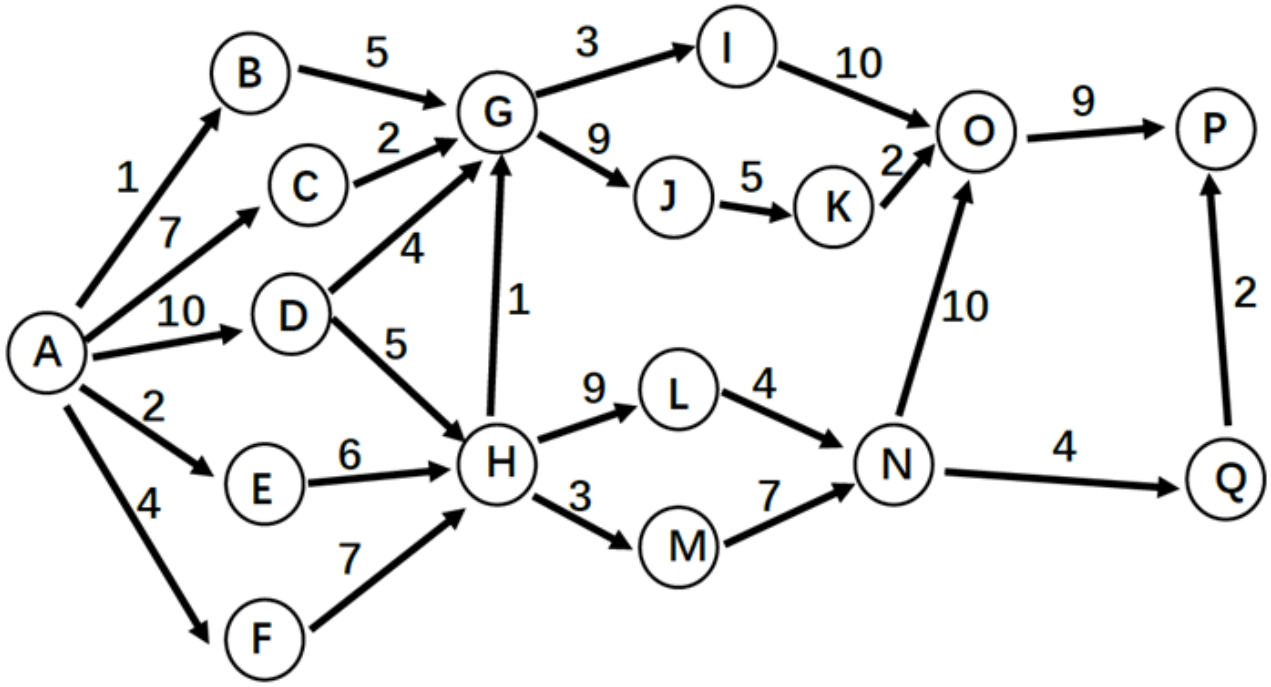
小组讨论，针对自己项目中的工作进行工作活动分解，分工进行各自合理的工作进度估算，最后汇总绘出项目活动图，找出关键路径。



关键路径：

模型划分、技术选型–>登陆注册–>实时地图模块–>内嵌地图、任务点–>手表端同步地图及活动内容

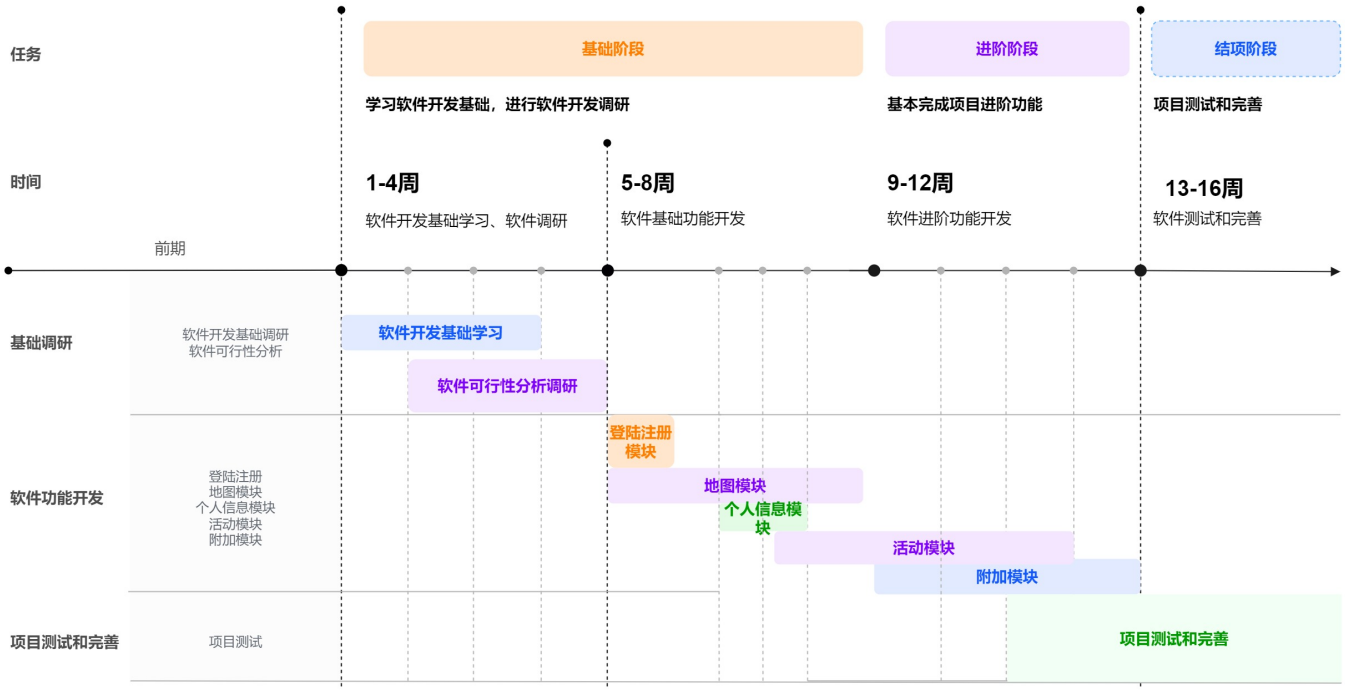
下图是一个软件开发项目的活动图，边长代表天数。请分析在图上标出每一个活动的最早开始时间、最晚开始时间和时差。然后找出关键路径和其总长度。



活动	最早开始时间	最晚开始时间	时差
AB	1	1	0

AC	1	1	0
AD	1	1	0
AE	1	1	0
AF	1	1	0
BG	2	2	0
CG	8	8	0
DG	11	11	0
DH	11	11	0
EH	3	3	0
FH	5	6	1
HG	9	11	2
GI	12	14	2
GJ	12	16	4
JK	21	24	3
KO	23	25	2
HL	16	18	2
HM	16	18	2
LN	25	30	5
MN	19	23	4
NO	26	28	2
OP	36	39	3
NQ	26	28	2
QP	30	32	2

练习项目跟踪工具的使用，如用甘特图记录跟踪项目过程。



调研国内外软件开发团队组织结构和工作方式对比。

分工调研国内与国外软件开发团队的管理方式对比（如：996工作制）。

从个人角度，你最喜欢的工作方式、工作环境条件、可接受的约束等是什么?

从团队项目管理角度，你认为最有效的项目组工作管理方式是什么?

一、国内与国外软件开发团队的管理方式对比主要涉及工作时间和工作文化等方面。以下是一些常见对比：

1. 工作时间：国内一些软件开发团队存在长时间工作的情况，如996工作制（早9点工作到晚9点，每周工作6天）。而在国外，普遍遵循标准工时，如每周40小时，注重工作与生活的平衡。
2. 工作文化：国内一些软件开发团队注重加班文化，对于加班有一定的社会认可。而在国外，强调效率和工作质量，倾向于避免长时间加班，并鼓励员工自我管理和个人发展。
3. 管理风格：国内软件开发团队通常采用较为集中的管理风格，强调上级权威和指令执行。而在国外，更加注重团队合作、沟通和共识，鼓励员工主动参与决策和发挥创造力。

二、从个人角度来看，我更喜欢以下工作方式、工作环境条件和约束：

1. 弹性工作时间：我倾向于有一定的弹性工作时间，可以根据任务和个人需求自主安排工作时间，倾

向于国外的工作环境。

2. 良好的工作环境：我比较看重工作环境，如舒适的办公空间、良好的设备和工具，就像微软和Facebook一样。
3. 平衡工作与生活：我认为工作与生活的平衡非常重要。合理的工作时间和强调休假制度能够保持工作的积极性和创造力，并提高工作效率。
4. 开放的沟通与合作：我喜欢能够自由地沟通和合作的工作环境。团队成员之间的互相支持、共享知识和合作是实现项目成功的关键。

三、从团队项目管理角度来看，我认为最有效的项目组工作管理方式包括以下几个方面：

1. 清晰的项目目标和角色定义：确保整个团队对项目目标的理解一致，并明确定义各个成员的角色和责任。
2. 敏捷开发方法：采用敏捷开发方法，如Scrum或Kanban，强调迭代和快速交付，保持团队的灵活性和适应性。
3. 有效的沟通与协作：建立良好的沟通渠道，促进团队成员之间的有效沟通和信息共享，包括定期会议、即时通讯工具等。
4. 追踪与监控：建立适当的项目追踪和监控机制，及时了解项目进展和问题，以便及时调整计划和解决挑战。
5. 鼓励创新和持续改进：鼓励团队成员提出新的想法和创新，同时持续改进团队的工作流程和实践，以提高工作效率和质量。

实验6

工作量估算

ch3 习题12（小组讨论）。

这种方法测量生产率并不完全合适。代码行数或应用点数并不能完全反映程序员的生产力和质量，因为不同的语言实现相同的功能可能产生不同数量的代码行数。此外，在实现开始之前无法测量代码行数，因此这种方法不能用于项目计划。最后，程序员可能会为了达到生产率目标而堆积代码，这会导致代码质量下降。因此，应该使用更全面的方法来测量生产率，例如使用功能点或实现的业务价值等指标。

具体来说，为了更准确地测量生产率，可以采用以下方法：

1. 使用功能点或实现的业务价值等指标来测量生产率，这些指标更能反映程序员的生产力和质量，而不受语言的影响。
2. 对于同样的设计，可以使用代码行数的平均值来测量生产率，但需要考虑语言的影响，并进行适当的调整。

3. 在项目开始之前，可以根据过去项目的经验和历史数据来估计生产率，但需要考虑项目的具体情况和特点，并进行适当的调整。
4. 在项目进行过程中，应该对程序员的生产力和质量进行监控和评估，并及时进行调整，以确保项目的进度和质量。
5. 应该鼓励程序员注重代码质量和可维护性，而不是单纯追求生产率，以提高项目的长期价值和可持续性。

参考书3.7（P94)皮卡地里电视广告销售系统按COCOMOII的工作量模型进行工作量估算的例子（结合P79–80表），估算自己项目的初始工作量。

根据以下步骤估算：

1. 确定项目的规模：根据项目的功能需求和规模，确定项目的规模。可以使用功能点或源代码行数等指标来衡量。
2. 选择适当的COCOMOII模型：根据项目的特点和规模，选择适当的COCOMOII模型。例如，可以选择基本模型、中间模型或详细模型等。
3. 确定模型参数：根据所选模型，确定相应的模型参数。例如，基本模型需要确定项目的规模因子和人月生产率等参数。
4. 计算初始工作量：根据所选模型和参数，计算项目的初始工作量。例如，对于基本模型，可以使用以下公式计算初始工作量：

初始工作量 = 规模因子 × (源代码行数/1000) ^ 人月生产率

其中，规模因子、人月生产率和源代码行数需要根据具体项目进行确定。

进行调整：根据实际情况和经验，对初始工作量进行适当的调整。例如，可以考虑项目的复杂性、开发环境的特点、团队的技能水平等因素，并进行相应的调整。

风险管理

ch3 习题11 （小组讨论）

对于学生软件开发项目，存在以下一些潜在的风险：

1. 时间管理风险：学生可能会在项目的时间管理方面出现问题，导致项目延期或无法按时完成。
2. 技术能力风险：学生可能缺乏足够的技术能力，无法完成项目所需的技术开发和实现。
3. 需求变更风险：学生可能无法完全理解和满足项目的需求，导致需求变更和项目延期。
4. 团队协作风险：学生可能在团队协作方面出现问题，导致沟通不畅、合作不力，影响项目进展和质

量。

5. 资源限制风险：学生可能受到资源限制，如时间、设备、经费等，影响项目的开发和实现。

风险暴露是指潜在风险变成实际问题的过程。为了减轻各种风险，可以使用以下技术：

1. 风险管理计划：建立风险管理计划，明确风险的分类、评估、控制和监控等措施。
2. 风险评估：对项目中的潜在风险进行评估和分析，确定风险的优先级和影响程度。
3. 风险控制：采取措施来控制和减轻各种风险，如制定时间进度表、技术培训、需求管理、团队建设等。
4. 风险监控：对项目中的风险进行监控和跟踪，及时发现和处理潜在的风险。
5. 团队协作：加强团队协作，建立有效的沟通机制，促进团队合作和协调。

总之，对于学生软件开发项目，需要认真分析和评估各种潜在风险，并采取相应的措施来减轻和控制风险。同时，加强团队协作和沟通，建立风险管理计划和监控机制，有助于确保项目的顺利完成。

分析自己项目中可能存在的风险，并进一步细化风险管理（做出风险分级及应对预案）。

1. 技术风险：涉及到技术实现方面的问题，如技术难点、技术选型不当、技术能力不足等。
2. 时间风险：涉及到项目进度方面的问题，如时间安排不合理、时间延误等。
3. 人员风险：涉及到人员方面的问题，如人员流失、人员纠纷、人员能力不足等。
4. 需求风险：涉及到需求方面的问题，如需求不明确、需求变更、需求不合理等。
5. 质量风险：涉及到产品质量方面的问题，如质量控制不当、质量缺陷、质量不符合标准等。

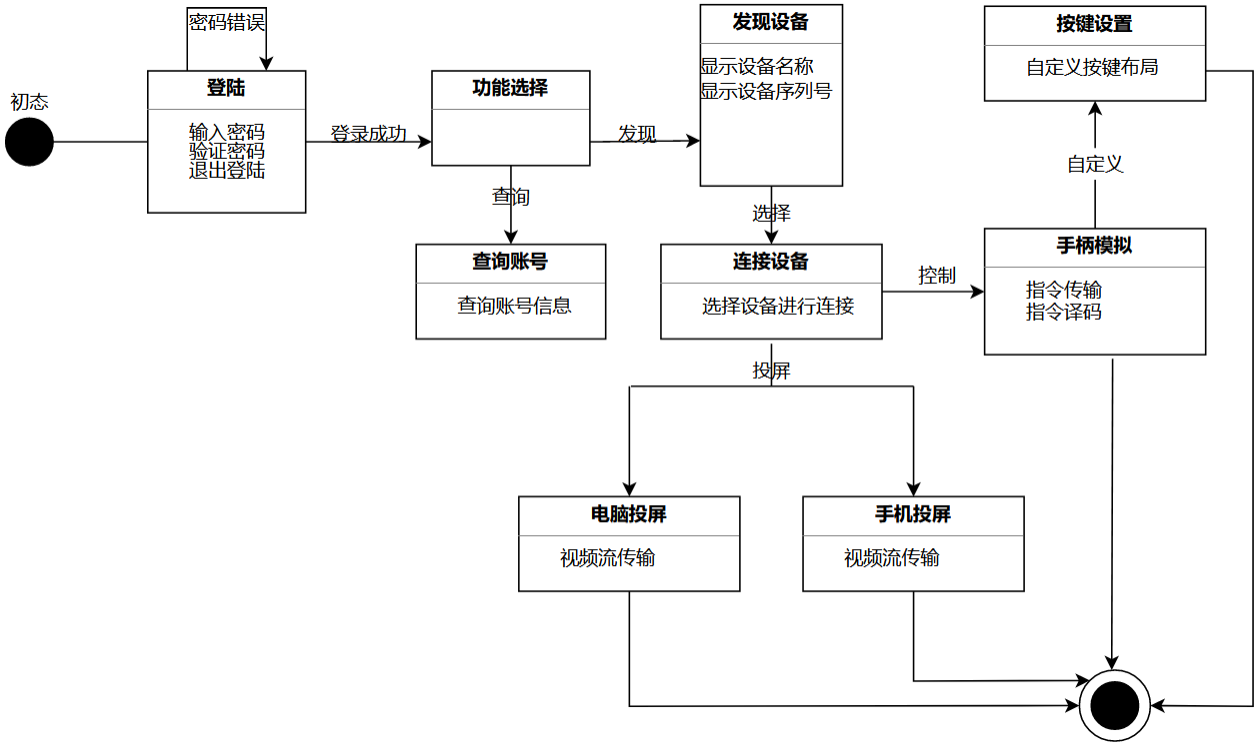
为了有效管理这些风险，可以采取以下措施：

1. 风险分级：对各种风险进行分类和评估，确定风险的优先级和影响程度，以便更有效地管理和控制风险。
2. 应对预案：对各种风险制定相应的应对预案，包括风险的控制和缓解措施、风险的应急处理和恢复措施等。
3. 风险监控：对各种风险进行监控和跟踪，及时发现和处理潜在的风险，以确保项目的顺利进行。
4. 团队建设：加强团队的协作和沟通，提高团队的技术能力和项目管理能力，以应对各种可能出现的风险。
5. 持续改进：在项目实施过程中，不断总结经验和教训，及时调整和改进项目管理和实施的方法和策略，以提高项目的质量和效率。

实验7

参照课本及PPT上例子，练习用静态建模（E-R、UML）等工具对所负责的系统建模，用模型model与用户沟通。分析、归纳、总结出符合实际的需求规格。

UML状态图：



1. 确定系统的功能和范围：首先要明确系统的功能和范围，包括系统的输入和输出、所涉及的业务流程、使用场景等。这有助于确定系统的主要需求和功能点。
2. 确定系统的实体和关系：根据系统的功能和范围，确定系统所涉及的实体和实体之间的关系。可以使用E-R图或UML类图等工具来表示实体和关系。
3. 确定系统的用例和场景：根据系统的功能和范围，确定系统的用例和场景。用例描述了系统与用户之间的交互，场景描述了用户如何使用系统完成特定的任务。
4. 确定系统的流程和状态：根据系统的业务流程和功能点，确定系统的流程和状态。可以使用UML活动图或状态图等工具来表示系统的流程和状态。
5. 确定系统的性能和安全需求：根据系统的功能和使用场景，确定系统的性能和安全需求。例如，系统的响应时间、并发处理能力、数据安全性等。
6. 与用户沟通并反复迭代：在建模过程中，需要与用户进行沟通，了解用户的需求和反馈，并根据用户的反馈进行相应的调整和迭代。这有助于确保建模结果符合实际需求。

总之，系统建模是一个迭代的过程，需要不断地与用户沟通、分析、归纳和总结，以确保建模结果符合实际需求。使用E-R图、UML类图、活动图、状态图等工具可以帮助我们更清晰地表示系统的实体、关系、流程和状态等，从而更好地理解和分析系统的需求。

实验8

阅读“SYSTEM MODELLING WITH PETRI NETS”，进一步学习Petri网知识，了解如何应用Petri网对系统进行建模

Petri网是一种适合于系统描述和分析的数学模型，主要描述异步和并发关系。（或者Petri网是对离散并行系统的数学表示，适用于描述异步的，并发的计算机系统模型。）

Petri网模型自然，直观，简单易懂的描述了在分析并行系统的状态行为的技术。

Petri网主要用于：计算机协议模型、柔性系统模型、计算机集成制造、人工智能、系统分析等领域。

Petri网由位置（P），变迁（T），有向弧线，令牌（token）所构成，其中P表示状态元素，T为变化元素，有向弧线既可以由P到T,又可以由T到P，token表示一种属性。其中P,T平等。T由P来描述，P由T而变化，T引起P中资源流动，F联系P,T。Petri网可描述为：{S,T,F,M0} S：表示状态，T：表示变化条件，F：表示有向弧，M0表示令牌的初始位置。

Petri网的直观理解：

用Petri网描述的系统有一个共同的特征：系统的动态行为表现为资源（物质资源和信息资源）的流动。

为了便于理解，先通过分布式系统的几个基本行为模型，描述的例子对Petri网做一个直观的说明：一个Petri网的结构元素包括：库所（place）、变迁（transition）和弧（arc）。库用于描述可能的系统局部状态，例如：计算机和通信系统的队列、缓冲、资源等。变迁用于描述修改系统状态的事件。例如：计算机和通信系统的信息处理、发送、资源的存取等。弧通过指向来规定局部状态和事件之间的关系。

在Petri网模型中，托肯包含在库所中，他们在库所中的动态的变化表示系统的不同状态。如果一个库所描述一个条件，它可以包含或者不包含托肯，也可以包含多个托肯。当库所中包含托肯时，条件为真；否则条件为假。如果一个库所定义一个状态，在这个库所中的托肯个数用于数量化这个状态。例如：在计算机和通信系统中，托肯可以表示处理的信息单元，资源单元和顾客、用户等对象实体。

一个Petri网模型的动态行为是由它的实施规则规定的，当使用等于1的弧权时，如果一个变迁的所有输入库所（这个库所连接到这个变迁，弧的方向是从库所到变迁）至少包含一个托肯，那么这个变迁使能（相关联的时间发生）。一个使能的变迁的触发导致从它所有的输入库中清楚1个托肯，在它的每一个输出库所（这些位置连接到这个变迁，弧的方向从变迁到位置）中产生1个托肯。当使用大于1的弧权

时，在变迁的每一个输入库所中都要包含至少等于连接弧权的托肯个数，它才使能；这个变迁的触发将清除在该变迁的每一个输入库所中的相应的托肯个数，并在变迁的每一个输出库中所产生相应的托肯个数。变迁的触发是原子操作，清楚输入库所的托肯和在输出库所产生托肯时1个不可分割的完整操作。

针对各自负责项目的不同场景，练习用各种动态建模工具（状态图、Petri网、数据流图、OCL逻辑等）建模需求，与用户沟通

一、数据流程图的作用是什么?

数据流程图是用于描述系统数据流程关系的工具，方便系统分析员在调查业务时候，与用户交换思维的实用工具。可以将数据流清晰的表现出来，更加形象具体。能够很方便地为读者阐述整个过程，包含功能、数据与之的联系。

二、数据流程图的符号数据

流程图常用的符号类型，可以大致分为4种，分别是：外部实体、处理过程、数据流和数据存储。1、外部实体：一般用正方形框来表示，输入外部实体的具体名称。2、处理过程：用带有圆角的长方形来表示。3、数据流：用一个水平箭头或者垂直的箭头来表示。4、数据存储：可以使用带开口的长方条来表示。

三、数据流程图的特点

数据流程图有一些特点，用于区分和其他流程图之间的关系。它的相关特点总结如下：数据流程图不同于传统的流程图和线框图，因为数据流程图的绘制基础是在数据的角度，而线框图示需要工作人员对原始数据处理后的角度绘制的。数据流程图是抽象化的，可以将各种具体的组织机构。工作场所和物质内容都去除掉，只保留下信息数据。数据流程图是概括性的，可以将系统里的各个业务之间的处理关系联系起来组合为一个整体。

四、数据流程图的绘制方法

绘制数据流程图，一般可以用Word自带的流程图工具，或者使用相关的流程图绘制工具。简单教程总结如下：使用数据流程图模板，新建画布；利用相应的流程图符号，组合成想要的作品结果；输入相关文字，美化流程图；导出为图片或者其他格式，完成数据流程图的绘制。

实验9

阅读“The Unified Modeling Language Reference Manual”，进一步学习UML知识，理解如何应用UML对系统进行建模

统一建模语言UML

统一建模语言UML是一种通用的可视化建模语言，可以用来描述、可视化、构造和文档化软件密集型系统的各种工作。

记录了与被构建系统的有关的决策和理解，可用于对系统的理解、设计、浏览、配置、维护以及控制系统的信息。

UML的应用范围

UML以面向对象的方式来描述系统。最广泛的应用是对软件系统进行建模，但它同样适用于许多非软件系统领域的系统。从理论上说，任何具有静态结构和动态行为的系统都可以使用UML进行建模。当UML应用于大多数软件系统的开发过程时，它从需求分析阶段到系统完成后的测试阶段都能起到重要作用。

在测试阶段，可以用UML图作为测试依据：用类图指导单元测试，用组件图和协作图指导集成测试，用用例图指导系统测试等。

初识UML

想要理解和使用UML，需要掌握UML的概念模型。UML的概念模型主要包括基本构造快、运用于构造快的通用机制和用于组织UML试图的架构。

UML构造快

构造块（building block）指的是UML的基本建模元素，是UML中用于表达的语言元素，是来自现实世界中的概念的抽象描述方法。

构造块包括事物（thing）、关系（relationship）和图（diagram）三个方面的内容。

- 事物是对模型中关键元素的抽象体现；
- 关系是事物和事物间联系的方式；
- 图是相关的事物及其关系的聚合表现。

事物

在UML中，事物是构成模型图的主要构造块，它们代表了一些面向对象的基本概念，事物被分为以下四种类型。

1. 结构事物结构事物（structural thing）通常作为UML模型的静态部分，用于描述概念元素或物理元素。结

构事物总称为类元（classifier）。常见的结构事物有类、接口、用例、协作、组件、节点等。

- 类（class）是对具有相同属性、相同操作、相同关系和相同语义的一组对象的描述。在UML图中使用矩形表示类，核心内容包括类名、属性、方法（操作）。
- 接口（interface）是一组操作的集合，这些操作包括类或组件的动作，描述了元素的外部可见行为。接口仅仅定义操作的数量和特征，但不提供具体的实现方法。接口可以被类继承，继承了某接口的类必须提供该接口所有操作的实现。接口一般不需要属性。在UML中接口的声明也使用矩形描述，在接口名上方使用构造型<>与类做区分。[见书 p17]
- 协作（collaboration）定义了一个交互，它是在为实现某个目标而共同工作、相互配合的多个元素之间的交互动作。协作具有结构、行为和纬度，一个类或对象可以参与多个协作。在UML图中把协作画成虚线椭圆，仅包含名称。这种表示法允许从外部把一个协作视为一个整体，但我们通常不使用这种表示方式，而是对协作内部更感兴趣。因此，我们往往会放大一个协作，将其内部细节引导到一些图中——特别是类图（用来表示协作的结构部分）和交互图（用来表示协作的行为部分）。
- 用例（use case）描述了一组动作序列，这些动作序列将作为服务由特定的参与者触发或执行，在过程中产生有价值、可观察的结果，结果可反馈给参与者或作为其他用例的参数。在UML图中用例表示为实现椭圆，仅包含名称。
- 组件（component）是系统中封装好的模块化部件，仅将外部接口暴露出来，内部实现被隐藏。组件可以由部件和部件之间的连接表示，其中部件也可以包含更小的组件。接口相同的部件可以互相替换。在UML图中将组件表示称矩形框，左边框上附着两个小矩形，框内些组件名。
- 节点（node）是在软件部署时需要的物理元素，其本质为一种计算机资源。一组组件可以存在于一个节点内，也可以从一个节点迁移到另一个节点。在UML图中节点用一个立方体表示，仅包含名称。

2. 行为事物行为事物（behavioral thing）也称为动作事物，是UML模型的动态部分，用于描述UML模型中的动态元素，主要为静态元素之间产生的时间和空间上的行为动作，类似于句子中动词的作用。
   - 交互（interaction）描述一种行为，它产生于协作完成一个任务的多个元素之间。交互包含消息、状态和连接。在UML图中消息表示为实箭头，源自消息发出者，指向接收者，箭头上方写操作名。
   - 状态机（state machine）定义了对象或行为在生命周期内的状态转移规则。状态机中包含状态、转移、条件（事件）以及活动。UML图中的状态机表示为圆角矩形，包含状态名。
   - 活动（activity）描述了一个操作执行时的过程信息。一个活动包含在操作执行过程中每一个步骤（动作）之间的先后序列关系。UML图中的活动也表示为圆角矩形，它和状态机图例的区分依靠语义。
3. 分组事物分组事物（grouping thing）又称组织事物，是UML模型的组织部分，是用来组织系统设计的事物。主要的分组事物是包，另外，其他基于包的扩展事物（例如子系统、层等）也可作为分组事物。
4. 注释事物注释事物（annotation thing）又称辅助事物，是UML模型的解释部分。这些注释事物用来描述、说明和标注模型的任何元素，简言之就是对UML中元素的注释。最主要的注释事物就是注解（note），是依附于一个元素或一组元素之上对其进行约束或解释的简单符号，内容为对元素的进一步解释文本。这些解释文本在UML图中可以附加到任何模型的任意位置上，用虚线连接到被解释的元素，当不需要显示注解时

可以隐藏，也可以以链接形式放到外部文本中（如果注释很长）。几乎所有的UML图形元素都可以用注解来说明。

关系

关系是模型元素之间具体化的语义连接，负责联系UML的各类事物，构造出结构良好的UML模型。

在UML中有四种主要的关系：

- 关联（association）：描述不同类元的实例之间的连接。它是一种结构化的关系，指一种对象和另一种对象之间存在联系，即“从一个对象可以访问另一个对象”。更详细地，我们说两个对象之间互相可以访问，那么这是一个**双向联系**，否则称为**单项联系**。关联中还有一种特殊的情况，称作**聚合关系**，聚合表示两个类元的实例具有整体和部分的关系，表示整体的模型元素可能是多个表示部分的模型元素的聚合。
- 依赖（dependency）：描述一对模型元素之间的内在联系（语义关系），若一个元素的某些特性随某一个独立元素的特性的改变而改变，则这个不是独立的，它依赖于上文所给的那个独立元素。
- 泛化（generalization）：类似于面向对象方法中的继承关系，是特殊到一般的一种归纳和分类关系。泛化可以添加约束条件，说明该泛化关系的使用方法或扩充方法，称为受限泛化。
- 实现（realization）：描述规格说明和其实现的元素之间的连接的一种关系。其中规定说明定义了行为的说明，真正的实现由后一个模型元素来完成。实现关系一般用于两种情况下：接口和实现接口的类和组件之间与用例和实现它们的协作之间。

这四种关系是UML模型中包含的最基本的关系，它们可以扩展和变形。

图

图是一组模型元素的图形表示，是模型的展示效果。多数的UML图是由通过路径连接的图形构成的。信息主要通过拓扑结构表示，而不依赖于符号的大小或者位置（有一些例外，如顺序图）。

根据UML图的基本功能和作用，可以将其划分为两大类：结构图（structure diagrams）和行为图（behaviour diagrams）。
结构图捕获事物与事物之间的静态关系，用来描述系统的静态结构模型；
行为图则捕获事物的交互过程如何产生系统的行为，用来描述系统的动态行为模型。

UML2

规范中包含14种图：类图、对象图、组合结构图、组件图、部署图、包图、外廓图、用例图、活动图、状态机图、顺序图、通信图、时序图、交互概念图。

UML通用机制

UML提供了四种通用机制，它们被一直地应用到模型中，描述了达到面向对象建模目的的4种策略，并在UML的不同语境下被反复运用，是的UML更简单并易于使用。
这四种机制分别是：规格说明（specifications）、修饰（adornments）、通用划分（common divisions）和扩展机制（extensibility mechanisms）。

规格说明

UML不仅仅是一个图形化的语言，恰恰相反，在每个图形符号后面都有一段描述用来说明构建模块的语法和语义。

UML的规格说明用来对系统的细节进行描述，在增加模型的规格说明时可以确定系统的更多性质，细化对系统的描述。通过规格说明，我们可以利用UML构建出一个可增量的模型，即首先分析确定UML图形，然后不断对该元素添加规格说明来完善其语义。

修饰

UML中大多数的元素都有一个唯一的和直接的图形符号，用来给元素的最重要的方面提供一个可视的表达方式。
修饰是对规格说明的文字的或图形的表示。
在UML中的每个元素符号都以一个基本的符号开始，在其上添加一些具有独特性的修饰。

通用划分

在面向对象系统建模中，通常有几种划分方法，其中最常见的两种划分是类型–实例与接口–实例。

UML扩展机制

构造型（stereotype）

标记值（tagged value）

约束（constraint）

用例图

用例图（use case diagram）是表示一个系统中用例与参与者关系之间的图。它描述了系统中相关的用户和系统对不同用户提供的功能和服务。
用例图是UML中对系统的动态方面建模的五种图之一（其他四种是活动图、状态机图、顺序图和通信图），是对系统、子系统和类的行为进行建模的核心。

用例图的组成元素

- 参与者凡有用例，必存在参与者。一个不能被任何用户感知到的“功能”和“事物”，在系统中存在的意义又是什么呢？ 太精辟了！什么是参与者？参与者（actor，也被译为执行者），是与系统主体交互的外部实体的类元，描述了一个或一组与系统产生交互的外部用户或外部事物。参与者以某种方式参与系统中一个或一组用例的执行。参与者位于系统边界之外，而不是系统的一部分。也就是说，参与者是从现实世界中与系统有交互的事物中抽象出来的，而并非系统中的一个类。在UML中，参与者有两种表示法。一般情况下，习惯用图表表示法来代表人，用类符号表示法来表示事物。此外参与者可以分栏来表示它的属性和接收到的事件。如何确定参与者？确定参与者是构建用例图的第一步。一般可以从以下几个角度来考虑：
    - 为系统提供输入的人或事物。
    - 接收系统输出的人或事物。
    - 需要接入的第三方系统或设备。
    - 时间是否会触发某些事件。
    - 负责支持或维护系统中信息的人。
- 除了从以上角度考虑参与者，还可以参考参与者的分类来进行确定。系统中的参与者一般可以分为四类。
    - 主要业务参与者（primary business actor）：主要从用例的执行中获得好处的关联人员。主要业务参与者可能会发起一个业务事件。
    - 主要系统参与者（primary system actor）：直接同系统交互以发起或触发业务或系统事件的关联人员。主要系统参与者可能会与主要业务参与者进行交互，以便使用系统。
    - 外部服务参与者（external server actor）：响应来自用例的请求的关联人员。
    - 外部接收参与者（external receiver actor）：从用例中接收某些价值或输出的非主要的关联人员。
- 用例在UML建模中，用例无疑是最重要的元素之一。
  用例图是UML中最重要的元素之一，准确的用例定义是在软件开发过程中不可或缺的要素。什么是用例？用例（use case，又被译作用况）是类元（一般是系统、子系统或类）提供的一个内聚的功能单元，表明系统与一个或多个参与者之间信息交换的顺序，也表明了系统执行的动作。一个用例就是系统的一个目标，描述为实现此目标的活动和系统交互的一个序列。用例的目标是要定义系统或子系统的一个行为，但不揭示系统的内部结构。用例是一种理解和记录系统需求的出色的技术，用例所描述的场景实际上包含了系统的一个或多个需求，因此用例与用例图被广泛用于系统的需求建模阶段，并在系统的整个生命周期中被不断细化。在UML中，用例用一个包含名称的椭圆形来表示，其中用例的名称可以显示在椭圆内部或椭圆下方。
  用例与参与者？参与者与用例是用例图中最重要的两个元素，二者也存在着密不可分的关系。用例是参与者与系统主体的不同交互作用的量化，是参与者请求或触发的一系列行为。一个用例可以隶属一个或多个参与者，一个参与者也可以参与一个或多个用例。没有参与任何用例的参与者是无意义的。用例与参与者之间存在关联关系，即参与者实例通过与用例实例传递消息实例（信号与调用）来与系统进行通信。

ER图

ER图的核心要素：

实体，属性，关系；

功能结构图

一般情况下产品或系统的总功能可分解为若干分功能，各分功能又可进一步分解为若干二级分功能，如此继续，直至各分功能被分解为功能单元为止。这种由分功能或功能单元按照其逻辑关系连成的结构称为功能结构。分功能或功能单元的相互关系可以用图来描述，表达分功能或功能单元相互关系或从属关系的图称为功能结构图。

定义

功能结构图就是按照功能的从属关系画成的图表，图中的每一个框都称为一个功能模块。功能模块可以根据具体情况分的大一点或小一点，分解得最小功能模块可以是一个程序中的每个处理过程，而较大的功能模块则可能是完成某一个任务的一组程序。

系统流程图

系统流程图是概括的描绘系统物理模型的传统工具。它的基本思想是用图形符号以黑盒子形式描绘系统里面的每个具体部件(程序、文件、数据库、表格、人工过程等），表达数据在系统各个部件之间流动的情况。

说明

系统流程图表达的是系统各部件的流动情况，而不是表示对信息进行加工处理的控制过程。

系统流程图的作用表现在以下几个方面：

1. 制作系统流程图的过程是系统分析员全面了解系统业务处理概况的过程，它是系统分析员做进一步分析的依据。
2. 系统流程图是系统分析员、管理员、业务操作员相互交流的工具。
3. 系统分析员可直接在系统流程图上画出可以有计算机处理的部分。
4. 可利用系统流程图来分析业务流程的合理性。

组织结构图

1. 可以显示其职能的划分.
2. 可以知道其权责是否适当.
3. 可以看出该人员的工作负荷是否过重.
4. 可以看出是否有无关人员承担几种较松散,无关系的工作.
5. 可以看出是否有让有才干的人没有发挥出来的情形.
6. 可以看出有没有让不胜任此项工作的人担任的重要职位.

7. 可以看出晋升的渠道是否畅通.
8. 可以显示出下次升级时谁是最合适的人选.

顺序图

是描述动态交互过程图，以时间为顺序，强调时间。

1. 包图 –> 组织结构图
2. 用例模型 –> 岗位职责图

浏览“LOGIC IN COMPUTER SCIENCE--Modelling and Reasoning about Systems”，了解常用逻辑及其在计算机学科中的应用

1、数理逻辑简介

数理逻辑又称符号逻辑、理论逻辑。它既是数学的一个分支，也是逻辑学的一个分支。是用数学方法研究逻辑或形式逻辑的学科。简而言之，数理逻辑就是精确化、数学化的形式逻辑。

逻辑是指事物的因果关系，或者说条件和结果的关系，这些因果关系可以用逻辑运算来表示，也就是用逻辑代数来描述。事物往往存在两种对立的状态，在逻辑代数中可以抽象地表示为 0 和 1 ，称为逻辑0状态和逻辑1状态。逻辑代数中的变量称为逻辑变量，用大写字母表示。逻辑变量的取值只有两种，即逻辑0和逻辑1，0 和 1 称为逻辑常量，并不表示数量的大小，而是表示两种对立的逻辑状态。

逻辑代数又称为布尔代数，逻辑代数也叫做开关代数。他最初的产生思想是利用计算的方法来代替人们思维中的逻辑推理过程，直到1847年，英国数学家布尔发表了《逻辑的数学分析》，建立了“布尔代数”，并创造一套符号系统，利用符号来表示逻辑中的各种概念，初步奠定数理逻辑的基础。它的基本运算是逻辑加、逻辑乘和逻辑非，也就是命题演算中的“与”、“或”、“非”，运算对象只有两个数 0和 1，相当于命题演算中的“真”和“假”。逻辑代数的运算特点如同电路分析中的开和关、高电位和低电位，利用电子元件可以组成相当于逻辑加、逻辑乘和逻辑非的门电路，就是逻辑元件。还能把简单的逻辑元件组成各种逻辑网络，这样任何复杂的 逻辑关系都可以有逻辑元件经过适当的组合来实现，从而使电子元件具有逻辑判断的功能。

命题逻辑等值演算就是命题代数，命题代数是一种特殊的逻辑代数。我们把命题看作运算的对象，如同代数中的数字、字母或代数式，而把逻辑连接词看作运算符号，就像代数中的“加、减、乘、除”那样，那么由简单命题组成复合和命题的过程，就可以当作逻辑运算的过程，也就是命题的演算。这样的逻辑运算也同代数运算一样具有一定的性质，满足一定的运算规律。例如满足交换律、结合律、分配律，同时也满足逻辑上的同一律、吸收律、双否定律、狄摩根定律、三段论定律等等。利用这些定律，我们可以进行逻辑推理，可以简化复合命题，可以推证两个复合命题是不是等价，也就是它们的真值表

是不是完全相同等等。设W是某一语言中所有命题构成的集合，且设T与F分别为真、假命题，∨、∧、¬分别为命题的析取、合取、否定联结词，则布尔代数〈W，∨，∧，¬，T，F〉就是命题代数。

逻辑代数与命题代数有所不同。它可以把1和0分别解释为命题的真和假，令变元只取1和0为值，即令其为二值的真值变元，并把¬、∨和∧解释为真值运算，从而得到一种提供命题真值运算定律的真值代数。而且，在二值的真值代数中特别可以有定理“p=1或p=0”，但在一般的命题代数和类代数中却没有与此相应的定理。

类代数就是集合代数。从逻辑代数、命题代数和类代数让我想到了代数结构（代数系统），布尔代数、集合代数、命题代数都是带两个二元运算和一个一元运算的代数结构。这是抽象代数中的内容，以后有时间再进行总结。

2、程序中的命题逻辑

逻辑代数在程序中无处不在，而一阶逻辑在人工智能中也被广泛应用。

程序的结构主要有顺序结构，条件结构和循环结构。数理逻辑在程序中的应用主要是条件结构，体现在语言上就是if...else...和switch控制语句。if后面括号里面的表达式就是命题，当命题为真时，执行紧跟后面的语句；为假时，执行else后面的语句。在写程序时，一定要注意复合命题的真假对应了哪些情况，熟悉命题公式的真值表，从而使自己的程序逻辑精确无误。switch语句只是分支语句，进行多项选择，并没有对命题变量的真假做出判断。有时需要对一些复杂的复合命题进行等值演算之后（也就是化简公式），再将命题变项写入条件中进行判断，这是为了简化判断逻辑以及代码的复杂度。在写if判断语句时，主要是根据情况的真假和条件，构造判断语句。

3、重要的等值式：

德摩根律：¬ （A∧B）⇔¬ A∨¬ B，¬ （A∨B）⇔¬ A∧¬ B

蕴涵等值式A→B⇔¬ （A∨B）

等价等值式A↔B⇔（A→B）∧（B→A）

假言异位A→B⇔¬ B→¬ A

等价否定等值式A↔B⇔¬ A↔¬ B

归谬论（A→B）∧（A→¬ B）⇔¬ A

等值演算：由已知的等值式推算出新的等值式的过程

分工协作，学习、检索研究经典软件体系结构案例。

Intro

1. 论文展示了传统和非传统的模块化设计
2. 如果按照传统的模块化思想（一个模块包含一个或者多个子过程）去实现非传统的模块化分解，在很多场景下将会非常低效

模块化设计

定义

对工程进行良好的分割可以确保系统的模块化设计。每个任务各自形成一个独立的、隔离的程序模块。在对相应模块进行实际编码时，它的**输入和输出定义明确，它与其他模块交互的接口定义也没有歧义**。在模块验收时，对模块的完整性也可以**独立测试**；很少需要对各个模块验收的先后顺序进行安排就可以开始模块验收工作。最后，由于整体系统是以模块化形式组织，可以将系统错误和缺陷**定位于特定的系统模块**，从而可以限制错误排查的范围。

但是没有实际的标准进行模块化设计，因此论文分析实例提出模块化标准

效益

1. 不同模块的开发人员交流很少且很明确
2. 灵活性（调整一个模块不会牵一发动全身）
3. 解耦，降低系统复杂性

传统与非传统的区别

非传统设计中，模块被当作是**任务承担者**，而不是子程序

模块化设计案例

KWIC索引（接受一些行，每行有若干词，每个词由若干字符组成，每行都可以循环移位：重复的第一个词删除，然后接到行末，KWIC把所有行的各种移位情况按照字母表顺序输出。）

一行的情况： I like apples –> apples I like –> I like apples –> like apples I

设计一：面向过程

输入模块：输入数据 移位模块：输入模块完成数据输入后调用，输出移位后的多行数据 排序模块：在移位完成后，将不同的行进行排列 输出模块：输出数据 管理模块：负责错误处理，内存分配之类的工作

优点：模块之间的数据共享，不同的计算功能被隔离在不同的模块中 缺点：对数据存储格式的变化将会影响几乎所有模块，这种分解难以支持有效的复用。

设计二：面向对象

行数据存储模块：将对行数据的处理方法封装在一个对象内 其他的模块与设计一类似，相当于是把对数据的处理专门抽离了出来

优点：接口可以统一使用行存储数据模块的格式，提高代码的复用性，接口描述起来也会简洁很多

设计三：管道–过滤体系（非论文上的）

过滤器

1. 输入过滤器 负责接收外界信息并转化为系统所需的数据流。
2. 处理过滤器 处理过滤器是系统内变换数据流的部件,它有一个入口和一个出口,数据经入口流入,经过处理过滤器内部处理之后从出口流出。
3. 输出过滤器

过滤器按照以上三种情况可分为两类:主动过滤器和被动过滤器。满足前两种情况的过滤器称为被动过滤器,满足最后一种情况的过滤器称为主动过滤器。

管道

管道作为过滤器之间数据流动的通道的软件部件,它的主要功能是连接各个过滤器,充当过滤器之间数据流的通道。管道具有**数据缓冲**以及提高过滤器之间的**并行性操作**的作用。管道由数据缓冲区,向数据缓冲区读和写数据,判断管道为空或已满等操作定义组成

模块化设计标准

1. 将过程变为模块（可以理解为将程序主要的步骤拆分成不同的模块）
2. 信息隐藏 （行数据存储模块所有步骤都有使用到，是将数据复杂的细节隐藏起来）

实验10–11

对比书上各种软件体系结构风格和视图特点

分层体系结构（Layered Architecture）：

特点：将系统划分为多个层次，每个层次有不同的功能和责任。较低层次的模块提供基本服务，较高层次的模块依赖于低层次的模块。

视图特点：主要关注系统的功能层次和依赖关系。

客户端-服务器体系结构（Client-Server Architecture）：

特点：系统分为客户端和服务器两个部分，客户端发起请求并接收响应，服务器提供服务并响应请求。

视图特点：关注系统中的客户端和服务器之间的交互和通信方式。

基于消息传递的体系结构（Message-Passing Architecture）：

特点：系统中的组件通过消息进行通信和交互，每个组件都有自己的状态和消息队列。

视图特点：关注系统中的组件和它们之间的消息传递机制。

事件驱动体系结构（Event-Driven Architecture）：

特点：系统中的组件通过事件进行通信和交互，当某个事件发生时，相关的组件会接收到通知并作出响应。

视图特点：关注系统中的事件和组件之间的关系以及事件的传播机制。

选择性调用体系结构（Selective Invocation Architecture）：

特点：系统中的组件通过接口提供服务，其他组件可以选择性地调用这些服务。

视图特点：关注系统中的接口和服务的定义以及组件之间的调用关系。

MVC（Model-View-Controller）是一种常见的软件体系结构风格，广泛应用于软件开发中。它的目标是将应用程序的逻辑和用户界面分离，以便更好地组织和管理代码。

MVC体系结构由三个核心组件组成：

1.模型（Model）：

模型代表应用程序的数据和业务逻辑。它负责处理数据的读取、写入和更新，以及对数据进行计算和验证。模型通常独立于用户界面，因此可以被多个视图共享。模型还可以通过事件机制通知视图和控制器关于数据的变化。

2.视图（View）：

视图是用户界面的表示，负责向用户展示数据并接收用户的输入。它通常是被动的，根据模型的数据来呈现相应的信息。视图的主要职责是提供用户友好的界面，使用户能够与应用程序进行交互。在MVC中，可以有多个视图来呈现同一份数据，每个视图都可以有不同的呈现方式。

3.控制器（Controller）：

控制器充当模型和视图之间的中介，负责处理用户的输入并相应地更新模型和视图。它接收用户的请求，协调模型的更新和视图的刷新，以保持数据的一致性。控制器还可以根据业务逻辑进行决策，选择合适的模型和视图进行操作。

MVC的核心思想是将应用程序的逻辑分离到模型中，将用户界面的呈现和用户交互分离到视图和控制器中。这种分离提供了更好的代码组织和可维护性，使不同的组件可以独立进行开发和测试。此外，MVC还支持模块化和重用，可以方便地替换或扩展模型、视图和控制器的任何一个组件。

Kruchten 4+1视图是一种软件体系结构设计方法，由Philippe Kruchten于1995年提出。它强调通过不同视图来描述软件系统的多个关注点，以全面、综合地描述系统的不同方面。这个方法包括四个主要视图和一个场景视图，以提供对软件系统的全面理解。

1.逻辑视图（Logical View）：

逻辑视图描述系统的功能方面，包括系统的主要功能、模块之间的关系以及各个模块的职责和行为。逻辑视图通常用于描述系统的结构和行为，以便开发团队可以理解系统的核心功能和模块之间的相互作用。

2.开发视图（Development View）：

开发视图关注的是软件系统的组织和开发方面，包括模块、构建、部署和测试等开发活动。开发视图通常用于支持软件开发人员的工作，帮助他们理解软件系统的结构和组织，以及开发和测试的过程。

3.进程视图（Process View）：

进程视图关注的是系统的并发和分布方面，包括系统的多个并发进程、线程或分布式组件之间的交互和通信。进程视图通常用于描述系统的运行时行为和性能特性，以便开发团队可以评估系统的性能和可伸缩性。

4.物理视图（Physical View）：

物理视图描述系统的物理部署和组织，包括硬件设备、网络拓扑和系统组件之间的关系。物理视图通常用于描述系统的部署环境和硬件配置，以便开发团队可以评估系统的可靠性和可扩展性。

5.场景视图（Scenarios View）：

场景视图用于描述系统的用例和交互场景，包括用户如何使用系统、系统对外部环境的响应等。场景视图通常用于支持系统需求分析和用户交互设计，以便开发团队可以理解系统的使用方式和用户期望。

通过这四个主要视图和一个场景视图的组合，Kruchten 4+1视图提供了对软件系统的全面描述，使不同的利益相关者可以从不同的角度理解和参与系统开发。这种方法强调在设计过程中综合考虑不同的关注点，并促进开发团队之间的沟通和合作。

研究经典软件体系结构案例KWIC

KWIC（Key Word In Context）是一个经典的软件体系结构案例，它用于生成索引和排序文本中的关键词。下面是对KWIC系统的简要介绍和分析：

1.系统描述：

KWIC系统接收一组文本输入，对每个输入进行处理，并生成带有关键词的上下文索引。该索引可以根据关键词进行排序，以便用户可以更方便地查找特定的文本片段。

2.架构特点：

KWIC系统采用了一种基于过滤器的体系结构，具有以下组件：

输入组件：负责接收文本输入，并将其传递给处理组件。

处理组件：将文本进行分割和重组，生成包含关键词的上下文索引。

排序组件：根据关键词对索引进行排序，以便用户可以按关键词查找。

输出组件：将生成的索引输出给用户进行检索和浏览。

3.优点：

可扩展性：KWIC系统的体系结构具有模块化和分层特性，使得可以很容易地添加新的过滤器和处理规则，以适应不同的需求。

可维护性：系统的模块化结构使得各个组件可以独立开发、测试和维护，从而提高了系统的可维护性。

可重用性：由于系统的模块化特性，可以将过滤器和处理规则进行重用，以便在其他文本处理系统中使用。

4.缺点：

性能限制：KWIC系统的基本架构可能对大量文本输入的处理效率存在一定的限制。当处理大量文本时，可能需要考虑性能优化措施，如并行处理或采用更高效的算法。

交互性限制：KWIC系统主要用于生成索引和排序，可能缺乏直接的用户交互界面。在实际应用中，可能需要进一步扩展以支持用户查询和浏览的功能。

KWIC系统作为一个经典的软件体系结构案例，突出了模块化和分层的设计原则，并展示了如何将输入、处理和输出等功能组织起来以实现特定的目标。这种体系结构设计可以为其他类似的文本处理系统提供参考，并提供了一种灵活、可扩展的架构模式。

小组成员给软件体系结构打分

分层体系结构（Layered Architecture）：

灵活性：9/10

可维护性：8/10

性能：6/10

扩展性：7/10

可测试性：8/10

客户端-服务器体系结构（Client-Server Architecture）：

灵活性：7/10

可维护性：7/10

性能：8/10

扩展性：9/10

可测试性：6/10

基于消息传递的体系结构（Message-Passing Architecture）：

灵活性：8/10

可维护性：7/10

性能：7/10

扩展性：8/10

可测试性：7/10

事件驱动体系结构（Event-Driven Architecture）：

灵活性：9/10

可维护性：7/10

性能：7/10

扩展性：8/10

可测试性：8/10

实验12

结合项目的进程和开发历程，从设计原则的几个方面，思考软件开发存在的问题和解决方案。

（1）模块化和高内聚性：

问题：软件开发过程中可能存在模块之间的依赖性过高、模块功能不清晰或模块功能重叠等问题，导致代码难以维护和扩展。

解决方案：采用模块化设计原则，将系统拆分为独立的模块，每个模块具有清晰的职责和功能。确保模块之间的耦合度尽可能低，提高模块的内聚性，使得每个模块可以独立开发、测试和维护。

（2）低耦合：

问题：模块之间的耦合度过高会导致系统的灵活性和可维护性降低，难以进行单元测试或进行模块的替换和重用。

解决方案：采用低耦合设计原则，通过定义清晰的接口和抽象层来降低模块之间的依赖性。使用松散耦合的通信机制，如事件驱动或消息传递，来减少直接依赖。同时，保持高内聚性，确保每个模块只关注自身的职责和功能。

（3）单一责任原则：

问题：某些模块或类可能承担了过多的责任，导致代码复杂、难以理解和维护。

解决方案：应用单一责任原则，确保每个模块或类只有一个明确的职责。将功能划分为更小的模块或类，并遵循单一责任原则来提高代码的可读性、可维护性和可测试性。

（4）开闭原则：

问题：在软件开发过程中，可能会遇到需求变更或功能扩展的情况，导致现有代码需要频繁修改，破坏了代码的稳定性。

解决方案：采用开闭原则，通过抽象和多态性来实现代码的可扩展性。将可变的部分封装成抽象的接口或基类，并定义可扩展的插件机制，以便在不修改现有代码的情况下进行功能扩展。

学习依赖注入技术

依赖注入（Dependency Injection，简称DI）是一种软件设计模式和技术，用于管理和解决模块之间的依赖关系。在传统的软件开发中，一个模块可能直接创建和依赖其他模块的实例，这导致了紧耦合的代码结构，难以进行模块的替换和测试。而依赖注入通过将依赖关系的创建和管理从模块内部移到外部，从而解耦模块之间的依赖关系。

具体来说，依赖注入的主要思想是将依赖的创建和注入过程交给一个独立的容器或框架来处理。模块只需要声明自己所需要的依赖项，而不需要负责依赖项的创建和生命周期管理。依赖注入可以通过以下几种方式实现：

（1）构造函数注入（Constructor Injection）：通过模块的构造函数接收依赖项作为参数，容器负责创建依赖项的实例并将其传递给模块。

（2）属性注入（Property Injection）：模块通过公开的属性或接口，容器在创建模块实例后，通过属性设置或方法调用将依赖项注入到模块中。

（3）接口注入（Interface Injection）：模块通过实现特定接口来接收依赖项，容器通过在模块上调用接口方法来将依赖项注入到模块中。

依赖注入的主要优点包括：

（1）解耦性：依赖注入将模块的依赖关系移出模块本身，使得模块之间的耦合度降低，提高了代码的可维护性和可测试性。

（2）可替代性：通过依赖注入，可以轻松替换模块的依赖项，以满足不同的需求或测试目的，而无需修改模块的代码。

（3）可扩展性：依赖注入使得添加新的依赖项变得简单，可以通过配置容器来添加或更改依赖项。

（4）可测试性：依赖注入使得对模块进行单元测试更加方便，可以通过注入模拟的依赖项来隔离测试和外部依赖。

论述利斯科夫替换原则（里氏代换原则）、单一职责原则、开闭原则、德（迪）米特法则、依赖倒转原则、合成复用原则，结合自己的实践项目举例说明如何应用 （保存到每个小组选定的协作开发平台上，以组为单位）。

里氏代换原则（Liskov Substitution Principle）强调子类对象必须能够替换其父类对象，而不影响程序的正确性和预期行为。

在我们的Bubbling活动管理系统中，有一个抽象的活动类（Activity），并且派生出了多个具体的活动类型类，如演讲活动（SpeechActivity）、体育活动（SportsActivity）和美食活动（FoodActivity）等。这些具体的活动类型类都继承自活动类，并实现了其特定的行为和属性。

在遵循里氏代换原则的情况下，可以在代码中使用活动类的实例来替换具体的活动类型类的实例，并保持系统的正确性和预期行为。例如，假设系统中有一个活动管理器（ActivityManager）类，负责管理所有的活动。可以使用活动类的实例来处理各种具体的活动类型，而不需要知道具体的活动类型。

单一职责原则（Single Responsibility Principle）强调一个类应该有且只有一个引起它变化的原因，即一个类应该只负责一个单一的职责或功能。当一个类承担了过多的职责时，它变得复杂、难以理解和维护，容易引发代码的脆弱性和不稳定性。

在活动管理系统中，可以通过单一职责原则来确保各个类具有清晰的职责和功能，以提高代码的可读性、可维护性和可测试性。以下是一些示例：

活动类（Activity）：

职责：封装活动的基本信息，如活动名称、时间、地点等。

不负责：处理活动的具体业务逻辑，如活动报名、签到等。

活动管理器类（ActivityManager）：

职责：管理活动的整体流程，包括活动的创建、编辑、删除等操作。

不负责：处理活动的具体业务逻辑，如报名和签到功能。

活动报名类（Registration）：

职责：处理活动的报名流程，验证报名信息、记录报名人数等。

不负责：处理其他与活动无关的功能，如签到和支付等。

通过将不同的功能划分到不同的类中，遵循单一职责原则，可以使得每个类专注于自己的职责，代码结构更加清晰。

开闭原则（Open-Closed Principle）是面向对象设计的基本原则之一，强调软件实体（类、模块、函数等）应该对扩展开放，对修改关闭。简单来说，它要求系统的设计应该是可扩展的，而不需要修改已有的代码。

在活动管理系统中，可以通过应用开闭原则来支持系统的扩展和变化，而不需要修改已有的代码。以下是一些示例：

活动类（Activity）：

开放性：定义了抽象的活动类，作为活动类型的基类，允许派生出不同具体的活动类型类。

封闭性：不需要修改活动类的代码，即可通过派生类添加新的活动类型。

活动管理器类（ActivityManager）：

开放性：通过接口或抽象类定义活动管理器的基本功能和方法，允许扩展具体的活动管理器子类。

封闭性：不需要修改活动管理器类的代码，即可通过新增子类扩展活动管理器的功能。

活动报名类（Registration）：

开放性：使用接口或抽象类定义活动报名的基本流程和方法，允许扩展具体的活动报名子类。

封闭性：不需要修改活动报名类的代码，即可通过新增子类扩展活动报名的功能。

通过遵循开闭原则，系统的扩展性得到了保证，当需要添加新的活动类型、新增活动管理功能或扩展活动报名流程时，可以通过扩展现有的抽象类或接口，并新增具体的子类来实现，而无需修改已有的代码。

迪米特法则（Law of Demeter）是面向对象设计的原则之一，强调一个对象应该尽可能少地了解其他对象的细节。简而言之，一个对象应该与其他对象保持最少的直接交互，而是通过尽可能少的方法调用来完成需要的功能。

在活动管理系统中，可以通过应用迪米特法则来降低对象之间的耦合度，提高系统的灵活性和可维护性。以下是一些示例：

活动类（Activity）：

活动类只与活动管理器类（ActivityManager）进行直接交互，通过提供必要的接口方法来向活动管理器报告自身的状态或获取必要的信息。

活动管理器类（ActivityManager）：

活动管理器类作为系统的核心组件，负责管理活动的整体流程。它不需要了解每个具体活动的细节，只需要与活动类进行交互，通过活动类提供的接口方法进行必要的操作。

活动报名类（Registration）：

活动报名类通过活动管理器类进行与活动的交互，不直接与具体的活动类进行交互。它向活动管理器报名人数、验证报名信息等，而不需要知道具体活动的实现细节。

通过遵循迪米特法则，对象之间的耦合度降低，对象之间的关系更加松散。每个对象只需要与少数相关的对象进行交互，不需要了解其他对象的内部细节。

依赖倒转原则（Dependency Inversion Principle）是面向对象设计的原则之一，强调高层模块不应该依赖低层模块的具体实现细节，而应该依赖于抽象。简单来说，高层模块和低层模块都应该依赖于抽象，而不是依赖于具体的实现。

在活动管理系统中，可以通过应用依赖倒转原则来降低模块之间的耦合度，提高系统的灵活性和可维护性。以下是一些示例：

活动类（Activity）：

活动类定义了一个抽象的活动接口，高层模块（如活动管理器）依赖于该抽象接口，而不是依赖于具体的活动类实现。这样，具体的活动类可以通过实现该接口来进行扩展。

活动管理器类（ActivityManager）：

活动管理器类依赖于活动接口，通过依赖注入的方式，将具体的活动对象传递给活动管理器。这样，活动管理器不需要知道具体的活动类型，只需要通过活动接口来调用活动的方法。

活动报名类（Registration）：

活动报名类同样依赖于活动接口，通过依赖注入的方式，将具体的活动对象传递给活动报名类。这样，活动报名类不需要知道具体的活动类型，只需要通过活动接口来进行报名操作。

通过遵循依赖倒转原则，模块之间的依赖关系更加松散，高层模块和低层模块都依赖于抽象接口，而不直接依赖于具体的实现。

合成复用原则（Composite/Aggregate Reuse Principle）是面向对象设计的原则之一，强调通过组合（合成）关系来实现对象的复用，而不是通过继承。简而言之，合成复用原则鼓励使用对象组合来构建更复杂的对象，而不是通过继承来获得复用。

在活动管理系统中，可以通过应用合成复用原则来实现对象的复用和组合，提高系统的灵活性和可维护性。以下是一些示例：

活动管理器类（ActivityManager）：

活动管理器类可以使用合成复用原则来组合不同的功能组件，例如地图组件、活动报名组件、支付组件等，以实现更复杂的活动管理功能。

通过将这些组件作为活动管理器的成员变量，可以在运行时灵活地配置和替换不同的组件实现，从而实现功能的复用和扩展。

活动报名类（Registration）：

活动报名类可以使用合成复用原则来组合不同的验证组件、通知组件等，以实现灵活的报名流程。

通过将这些组件作为活动报名类的成员变量，并通过接口定义它们的行为，可以在运行时灵活地配置和替换不同的组件实现，以满足不同的需求。

通过遵循合成复用原则，可以将系统的功能划分为不同的组件，通过组合这些组件来实现更复杂的功能，而不是通过继承来获得复用。

实验13

结合项目的进程和开发历程，分析软件采用那些设计模式

MVC（Model–View–Controller）模式：

Model–View–Controller（模型–视图–控制器）模式是一种常见的软件架构模式，它将应用程序划分为三个主要部分：模型、视图和控制器。

在项目中，模型表示活动的数据和业务逻辑，视图表示用户界面的展示，而控制器负责处理用户输入、更新模型数据和更新视图显示。这种分离使得系统更加灵活和可维护，例如，当活动数据发生变化时，控制器可以更新模型，并通知视图进行相应的更新，从而保持数据和界面的一致性。

依赖注入（Dependency Injection）模式：

依赖注入模式用于解耦组件之间的依赖关系，通过将依赖对象的创建和传递责任委托给外部的注入器或容器，以实现对象之间的松耦合。

在项目中，例如，活动管理器（Controller）可能依赖于活动服务（Activity Service）来处理活动的逻辑。使用依赖注入模式，您可以在活动管理器中声明活动服务的接口，并在运行时将具体的活动服务对象注入到活动管理器中。这样，活动管理器可以与不同的活动服务实现进行交互，而不需要直接依赖于具体的实现类。

工厂模式（Factory Pattern）：

工厂模式用于封装对象的创建过程，并通过一个共同的接口来创建对象的实例。

在项目中，可以使用工厂模式来根据不同的活动类型创建对应的活动对象。例如，活动工厂可以根据不同的活动类型（如讲座、比赛、展览等）创建相应的活动对象，通过一个统一的接口方法返回活动对象给活动管理器。这样，活动管理器可以使用统一的方式处理不同类型的活动，而不需要直接关注具体的活动对象的创建过程。

观察者模式（Observer Pattern）：

观察者模式用于实现对象之间的一对多依赖关系，当一个对象的状态发生变化时，它的所有依赖对象将自动收到通知并作出相应的更新。

在项目中，例如，活动报名组件（观察者）可以注册为活动管理器（被观察者）的观察者。当活动管理器的活动报名人数发生变化时，活动报名组件将收到通知，并更新界面显示相应的报名人数。这样，活动报名组件与活动管理器之间实现了解耦，当活动管理器的状态变化时，不需要直接操作活动报名组件的代码，而是通过观察者模式自动更新。

给出4种设计模式的例子，并总结其特点 （保存到每个小组选定的协作开发平台上）

创建型模式 – 工厂模式（Factory Pattern）：

例子：活动管理系统中的活动工厂，根据不同的活动类型创建相应的活动对象。

特点：封装对象的创建过程，通过一个共同的接口来创建对象的实例。提供了一种统一的方式来创建对象，避免了直接依赖具体类的问题，提高了代码的可扩展性和可维护性。

结构型模式 – 适配器模式（Adapter Pattern）：

例子：地图接口适配器，将不同地图服务提供商的接口适配为统一的地图接口。

特点：将一个类的接口转换为客户端所期望的另一个接口，使得原本不兼容的类能够协同工作。通过适配器，客户端可以统一调用适配后的接口，实现了类之间的解耦和互操作。

行为型模式 – 观察者模式（Observer Pattern）：

例子：活动报名组件注册为活动管理器的观察者，当活动报名人数变化时，更新界面显示。

特点：定义了对象之间的一对多依赖关系，当一个对象的状态发生变化时，其所有依赖对象将自动收到通知并作出相应的更新。实现了解耦和动态的对象间通信。

并发型模式 – 单例模式（Singleton Pattern）：

例子：数据库连接池，保证系统中只有一个数据库连接池的实例。

特点：确保一个类只有一个实例，并提供全局访问点来获取该实例。通过单例模式，可以实现对资源的统一管理和控制，确保线程安全和节省系统资源。

实验14

阅读下面软件测试相关资料（或查阅其它相关资料）Software Testing–A Research Travelogue (2000—2014).pdf，了解软件测试的基本概念、主要技术、重要挑战等

《Software Testing–A Research Travelogue (2000—2014)》是一篇研究综述，介绍了软件测试领域在2000年至2014年期间的研究进展和趋势。

（1）基本概念：

软件测试是一种验证和验证软件系统是否符合预期行为的活动。它涉及开发测试策略、设计测试用例、执行测试并分析测试结果的过程。

软件测试的目标是发现软件中的错误、验证系统是否满足需求、提供可靠的软件质量信息以及改善软件开发过程。

（2）主要技术：

黑盒测试（Black Box Testing）：基于输入和输出的测试方法，不考虑内部实现细节，关注系统的功能和规格。

白盒测试（White Box Testing）：基于内部结构的测试方法，关注程序的逻辑路径和代码覆盖率。

自动化测试（Automated Testing）：使用测试工具和脚本自动执行测试用例，提高测试效率和准确性。

基于模型的测试（Model-based Testing）：使用模型来表示系统行为，生成测试用例以覆盖系统的各种情况。

演化测试（Mutation Testing）：通过人为引入错误的方式来评估测试用例的质量和系统的容错能力。

（3）重要挑战：

测试覆盖率问题：如何设计测试用例以覆盖系统的各种情况，以及如何衡量测试的完整性和有效性。

自动化测试问题：自动化测试需要投资于工具和脚本的开发和维护，并且某些功能和性能测试仍然需要人工干预。

多样性和复杂性问题：软件系统日益复杂，涉及多个平台、多个设备和多个用户，如何测试这种多样性和复杂性是一个挑战。

效率和成本问题：在时间和资源有限的情况下，如何设计测试策略和分配资源，以最大程度地发现错误并提供可靠的质量信息。

深入理解白盒测试和黑盒测试，总结其特点 （保存到每个小组选定的协作开发平台上，以组为单位）

WhiteBox.pdf

BlackBox.pdf

白盒测试：

特点：白盒测试是基于内部结构和逻辑的测试方法，也称为结构化测试或透明盒测试。

目标：白盒测试关注于检查和评估软件系统的内部工作方式和实现细节，以发现可能存在的逻辑错误、路径覆盖不足、代码漏洞等问题。

实施：测试人员需要了解被测试软件的内部结构、算法和代码逻辑，使用这些知识来设计测试用例和执行测试。

方法：白盒测试使用各种技术，如语句覆盖、分支覆盖、路径覆盖、条件覆盖等，以确保测试用例能够覆盖系统中的各种执行路径和代码段。

黑盒测试：

特点：黑盒测试是基于系统功能和需求的测试方法，也称为功能性测试或不透明盒测试。

目标：黑盒测试关注于测试软件系统的功能是否符合规格和预期行为，而不关心其内部实现和结构。

实施：测试人员不需要了解被测试软件的内部结构，只需要根据需求和规格文档设计测试用例和执行测试。

方法：黑盒测试使用各种技术，如等价类划分、边界值分析、决策表、状态转换等，以确保测试用例能够覆盖不同的输入组合和系统行为。

小结

白盒测试关注于软件的内部结构和实现，以发现逻辑错误和代码漏洞。

黑盒测试关注于软件的功能和需求，以验证系统是否按照规格和预期行为进行操作。

白盒测试需要了解内部结构和逻辑，使用结构相关的覆盖准则进行测试设计。

黑盒测试不需要了解内部结构，根据需求和规格进行测试用例设计。

两种测试方法可以互补使用，以达到更全面和有效的测试覆盖。

阅读下面符号测试 （Symbolic Testing）相关资料（或查阅其它相关资料），了解符号测试的基本概念、主要技术、重要挑战等

A Survey of Symbolic Execution Techniques.pdf

Symbolic Execution and Program Testing.pdf

Symbolic Execution for Software Testing–Three Decades Later.pdf

基本概念：

符号测试是一种基于符号执行的软件测试方法，它通过使用符号值而不是具体的输入值来分析程序的路径和约束条件。

符号值代表了输入变量的未具体化的值，允许测试程序的不同路径和边界情况，以发现潜在的错误和漏洞。

主要技术：

符号执行（Symbolic Execution）：符号执行通过在程序中跟踪符号值的符号操作，而不是具体值的实际操作，来推导程序的执行路径和约束条件。

符号约束求解（Symbolic Constraint Solving）：符号约束求解器用于解析程序中的符号约束条件，以确定满足约束的具体输入值。

路径选择（Path Selection）：路径选择算法根据特定的目标（如代码覆盖率、错误探索等）来选择最有可能暴露错误的路径进行符号执行。

符号执行的扩展技术：包括符号执行的组合、符号执行的优化、符号执行的并行化等技术，用于提高符号执行的效率和可扩展性。

重要挑战：

路径爆炸问题：符号执行可能导致路径组合的爆炸增长，从而导致无法有效地覆盖所有可能的路径。

符号约束求解问题：符号约束求解器的性能和可扩展性是一个挑战，特别是处理复杂的约束条件时。

外部环境和系统交互问题：符号执行通常无法处理涉及外部环境和系统交互的情况，如文件操作、网络通信等。

精确性和误报问题：符号执行可能产生大量的路径和约束，需要仔细分析和筛选，以区分真正的错误和误报。

阅读下面差分测试 （Differential Testing）相关资料（或查阅其它相关资料），了解差分测试的基本原理、主要应用等

Differential Testing for Software.pdf

Feedback-Directed Differential Testing of Interactive Debuggers.pdf

基本原理：

差分测试是一种软件测试方法，通过比较多个软件实现的输出或行为差异来发现潜在的错误和漏洞。

差分测试基于两个或多个不同的实现（如不同版本、不同算法或不同编译器）之间的差异，通过输入相同的测试用例来检测这些差异。

主要应用：

软件版本测试：差分测试可以用于比较不同版本的软件实现，以发现在新版本中引入的错误或功能变更。

编译器测试：差分测试可以用于比较不同编译器生成的目标代码，以检测编译器的错误或优化差异。

漏洞发现：差分测试可以用于发现具有相同功能但实现不同的软件实体中的漏洞，如不同的浏览器或操作系统之间的差异。

安全性评估：差分测试可以用于评估软件系统在不同环境或配置下的安全性，如测试不同的网络协议栈对攻击的抵抗能力。

差分测试的流程：

选择测试集：根据差分测试的目标和应用场景，选择一组合适的测试用例。

执行测试：运行被比较的软件实现，并记录其输出或行为。

比较差异：对比不同实现之间的输出或行为，找出差异和潜在的错误。

分析结果：分析差异并进行错误跟踪、优化或修复。