实验九 UML，逻辑， 软件体系结构设计（一）

实验目的：

1. 深入理解UML

2. 了解计算机学科中的逻辑

3. 学习对比软件体系结构设计GB和IEEE最新SAD (Software Architecture Document)的标准

4. 研究经典软件体系结构案例

3. 完成自己项目的SRS

实验内容：

1. 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 （刘易）

UML (统一建模语言) 是一种通用的可视化建模语言，用于指定、可视化、构建和记录软件系统的工件。它可以帮助你从不同的角度理解系统，并为开发人员、架构师、项目经理等提供一个共同的语言。

1. UML 视图：

UML 将系统分解成不同的视图，每个视图从不同的角度描述系统：

结构分类视图：

静态视图 (类图): 描述系统的静态结构，包括类、接口、数据类型以及它们之间的关系（例如，关联、泛化、依赖和实现）。

用例视图 (用例图): 描述系统功能以及参与者与系统之间的交互。

实现视图 (组件图): 描述系统的物理实现，包括组件、接口和它们之间的依赖关系。

部署视图 (部署图): 描述系统的物理部署，包括节点、组件实例和对象实例的配置。

动态行为视图：

状态机视图 (状态图): 描述对象在生命周期中响应事件的状态变化和行为。

活动视图 (活动图): 描述系统中活动的流程，包括顺序和并发活动。

交互视图 (序列图和协作图): 描述对象之间发送消息的顺序和方式。

模型管理视图：

描述模型本身的组织方式，包括包、子系统和模型之间的依赖关系。

2. UML 元素：

每个视图都包含一组特定的元素，用于描述系统的不同方面：

类: 描述一组具有相同属性、操作、方法、关系和行为的对象。

接口: 描述一组操作，但没有给出它们的实现或状态。

数据类型: 描述一组没有身份的原始值。

用例: 描述系统执行的一系列动作，包括与参与者之间的交互。

参与者: 描述与系统交互的外部实体。

组件: 描述系统中可替换的物理实现单元。

节点: 描述运行时计算资源，例如计算机或设备。

状态: 描述对象生命周期中的一个特定条件或情况。

转换: 描述对象在响应事件时如何从一个状态转移到另一个状态。

消息: 描述对象之间传递的信息。

包: 用于组织元素的通用机制。

3. 建模步骤：

使用 UML 建模的过程通常遵循以下步骤：

确定系统范围和目标： 了解你要建模的系统及其预期功能。

识别参与者和用例： 确定与系统交互的外部实体以及他们想要执行的操作。

创建用例图： 使用用例和参与者来描述系统功能。

创建静态视图： 使用类图来描述系统的静态结构。

创建动态视图： 使用状态图、活动图、序列图和协作图来描述系统的动态行为。

创建实现视图和部署视图： 描述系统的物理实现和部署。

细化和迭代： 根据需要细化和修改模型，以确保它准确地反映系统。

4. 工具和资源：

可以使用 UML 建模工具来创建和管理模型，例如：

Visual Paradigm

Enterprise Architect

StarUML

2. 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 （丁弘扬）

命题逻辑：这是最基本的逻辑系统，涉及真值命题及其之间的逻辑关系，可以对逻辑表达式的真假进行推理的数学模型。在计算机科学中，命题逻辑常用于描述和验证程序的正确性和系统的性质。

谓词逻辑：研究的是包含谓词（表示性质或关系）和变量的句子的逻辑结构和推理规则。谓词逻辑能够表达原子单元的内部结构，并且具有强大的表达能力和灵活性，可以在不同的知识之间建立联系。在计算机学科中常用于描述数据结构，算法和程序的行为，此外，它也在数据库查询语言（如SQL）和人工智能领域（如知识表示和推理）中发挥重要作用。

模态逻辑：模态逻辑引入了模态算子，如“可能”和“必然”，以描述陈述的可能性和必然性。在计算机科学中，模态逻辑用于描述和推理关于系统状态，行为或属性的可能性和必然性。例如，在并发系统和分布式系统的建模中，模态逻辑可以帮助分析和推理系统状态之间的转换和性质。

时态逻辑：时态逻辑考虑了时间因素，允许我们描述和推理关于系统状态随时间变化的性质。在计算机学科中时态逻辑常用于描述和分析实时系统，并发系统和反应式系统的行为。通过时态逻辑，我们可以形式化表达系统的时态属性，如安全性，活性和稳定性，并进行验证。

3. 分工协作，参考国标“13 - 软件(结构)设计说明(SDD)”等资料，对比参考SAD最新标准IEEE-42010.pdf，针对自己的项目设计SAD初稿。 （王攀）

4. 分工协作，学习、检索研究经典软件体系结构案例。

On-the-Criteria-To-Be-Used-in-Decomposing-Systems-into-Modules.pdf

<http://www.cs.cmu.edu/~ModProb/index.html> （张勇）

软件模块化设计

模块化的目的：提高系统的灵活性和可理解性，同时缩短开发时间。

模块化的效果：依赖于划分系统为模块时所使用的标准。

系统设计问题：论文提出了一个系统设计问题，并描述了传统和非传统的分解方法，展示了非传统分解在实现目标上的优势。

模块化标准：讨论了划分系统时所使用的标准，并提出了一些可用于系统模块化分解的标准，包括信息隐藏（每个模块隐藏一个设计决策，不让其他模块知道）和独立开发等。

模块化实现：指出如果按照传统的假设（一个模块由一个或多个子程序组成）来实现非传统分解，那么在大多数情况下效率会较低。论文草拟了一种替代的实现方法，该方法不会有这种效率影响，例如通过使用汇编器将程序编写为看起来像是子程序的函数，但实际上根据需要进行适当的汇编。

模块化的好处：包括管理上的便利（开发时间缩短）、产品的灵活性（一个模块的大幅更改不必影响其他模块）、以及可理解性（可以逐个模块地研究系统，从而更好地理解并设计整个系统）。

KWIC索引生产系统：作为例子，论文详细描述了一个KWIC索引系统的模块化，包括输入、循环位移、排序、输出和主控制模块。

模块化比较：比较了两种不同的模块化方法，讨论了它们在可变性、独立开发和可理解性方面的差异。

效率与实现：讨论了如何通过非传统的模块化方法实现效率，包括使用汇编器将程序编写为看起来像是子程序的函数，但实际上根据需要进行适当的汇编。

编译器和解释器的共同分解：论文还讨论了如何将模块化规则应用于编译器和解释器的设计，发现早期的分解对于编译器和解释器都是有效的。

层次结构：论文提出了程序的层次结构概念，并讨论了如何通过模块化设计来实现层次结构，以及层次结构的好处。

结论：建议不要基于流程图开始系统模块化的分解，而是从可能变化或难以设计的决策列表开始，每个模块都设计为隐藏这些决策。为了实现高效，需要放弃模块是子程序集合的假设，而是允许子程序和程序成为来自不同模块的代码集合。

5. 完成软件需求规格说明SRS （李雨桐）

**下周五（含）前将软件需求规格说明提交给相应的助教**

项目跟踪，建立能反映项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。