**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

5. 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**

UML（Unified Modeling Language）是一种通用的、视觉化的建模语言，用于软件系统的规范、可视化、构造和文档化。它捕获关于必须构建的系统的设计决策和理解，适用于所有开发方法、生命周期阶段、应用领域和媒介。UML旨在统一过去的建模技术经验和将当前软件最佳实践融入标准方法中，包含语义概念、表示法和指导原则。其具有静态、动态、环境和组织部分，设计为与交互式视觉建模工具配合使用，这些工具带有代码生成器和报告编写器。

UML并不定义一个标准过程，但旨在与迭代开发过程兼容，支持大多数现有的面向对象开发过程。它捕获系统的静态结构和动态行为信息，将系统建模为一组离散的对象，这些对象通过互动完成工作，最终使外部用户受益。静态结构定义了对系统及其实施重要的对象类型及对象间的关系，而动态行为则定义了对象随时间的历史和对象间的通信以实现目标。

从几个分离但相关的视角对系统进行建模，使得系统能够因不同目的而被理解。UML还包括组织结构，用于将模型组织成包，允许软件团队将大型系统划分为可操作的部分，理解并控制包之间的依赖关系，并能组织工作。

此外，UML提供了扩展机制，如约束、标记值和刻板印象，允许建模者在不修改底层建模语言的情况下进行一些常见扩展。这些机制设计得让工具无需理解扩展的完整语义就能存储和操作它们，通常作为字符串处理。虽然这可能不满足所有需求，但可以简单地实现大多数建模者所需的定制。UML不强制要求特定的开发过程，但设计时考虑到了支持迭代、增量、用例驱动且具有强大架构焦点的过程，这种过程被认为最适合现代复杂系统的开发。

**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**

1. 命题逻辑（Propositional Logic）：

简介：命题逻辑关注的是没有内部结构的简单陈述句（称为命题）之间的逻辑关系，例如真与假、与、或、非等逻辑运算。

应用：在计算机科学中，命题逻辑被用于软件验证、自动定理证明、程序正确性证明、硬件设计验证以及数据库查询语言的逻辑表达式。

1. 谓词逻辑（Predicate Logic）：

简介：谓词逻辑扩展了命题逻辑，允许使用变量和谓词来描述对象集合的性质，从而表达更复杂的陈述。

应用：在形式化方法中，谓词逻辑用于表达和验证软件和硬件系统的性质。它也是人工智能、专家系统、知识表示和数据完整性约束的关键组成部分。

1. 一阶逻辑（First-Order Logic）：

简介：一阶逻辑是最常用的谓词逻辑形式，它支持量化（全称量词和存在量词），能够表达关于个体对象的性质和关系。

应用：在数据库查询语言SQL中，一阶逻辑被用来表达查询条件。此外，它还在形式语义学、自动推理系统和计算机辅助证明中扮演重要角色。

1. 模态逻辑（Modal Logic）：

简介：模态逻辑引入了额外的算子来表达可能性、必然性、时间或认识论上的模态概念。

应用：在计算领域，模态逻辑常用于时序逻辑、知识表示、多智能体系统、并发和分布式系统的规范语言，以及安全协议的形式分析。

1. 时态逻辑（Temporal Logic）：

简介：专注于时间的逻辑，特别关注状态随时间的变化，如LTL（线性时态逻辑）和CTL（计算树逻辑）。

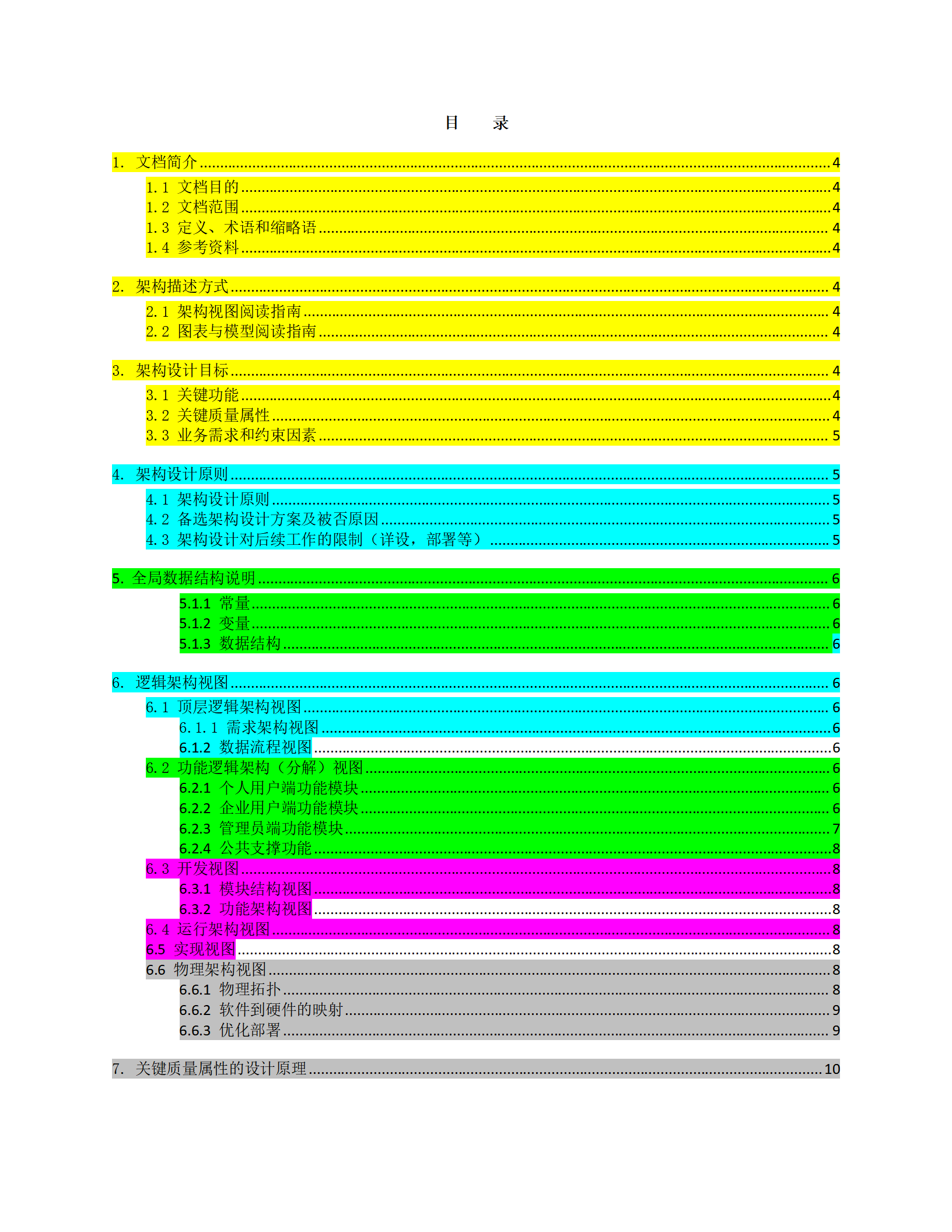
应用：在验证计算系统的时序行为、实时系统、并发系统和反应式系统中，时态逻辑是非常关键的工具。

1. 命题逻辑的公理系统与自然演绎：

简介：书中介绍了如何通过一系列规则进行命题逻辑的推导，包括自然演绎系统，它是无矛盾性、完备性和可靠性研究的基础。

应用：这些基础理论在自动化验证、算法设计和编程语言理论中有直接应用，帮助确保程序的正确性和安全性。

1. **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。**



**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**

研究经典软件体系结构时，关注点在于如何有效地将系统分解为模块，以提升软件的灵活性、可理解性和开发效率。根据D.L. Parnas的研究，一个关键观点是开始系统模块化不应该基于流程图，而是应该从一系列设计决策出发，特别是那些困难或可能发生变化的决策。每个模块应当围绕这样的决策进行设计，以隐藏这些决策细节，使得它们对外部模块不可见。由于设计决策往往不局限于程序执行的时间点，因此模块划分不会直接对应于处理步骤，而是更多地基于逻辑上的抽象和职责分离。

为了实现高效的模块化设计，传统观念中模块由一个或多个子程序组成的假设需要被放弃。取而代之的是，允许子程序和程序作为不同模块代码的集合体来组装。这样，即使采用非常规的模块分解方式，也可以通过合理的实现技术克服效率问题。

根据相关资料，我总结了一些模块化准则：

1. 数据结构与其访问和修改过程：作为一个整体包含在一个模块内，避免跨模块共享，这有助于封装和减少耦合。
2. 调用序列与函数：调用特定例程所需的指令序列与该例程本身应属于同一模块，便于管理和优化调用机制。
3. 控制块格式：在操作系统等程序中使用的队列控制块格式应封装在单独的“控制块模块”内，避免格式变动对多个模块造成影响。
4. 字符编码和排序规则：这类数据应被隐藏在一个模块内，以提供最大的灵活性。
5. 处理顺序：应尽可能在单个模块内部决定处理序列，以减少因外部条件变化导致的调整需求。

**5. 完成软件需求规格说明SRS**

已上交

**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。