**实验十 软件体系结构设计（二）**

实验目的：

1. 给项目添加提高质量属性的设计
2. 体系结构风格和视图特点
3. 练习故障树分析
4. 研究经典软件体系结构案例
5. 继续补充和修改自己项目的SAD

实验内容：

1. 学习、检索课本5.17参考文献及以下推荐的参考书或网上检索新的有关软件体系结构的资料。

小组分工，每位成员选择自己关注的部分专题学习并写出学习报告（笔记）（附到最终提交的SAD）。

Software Architecture in Practice ，3rd Edition

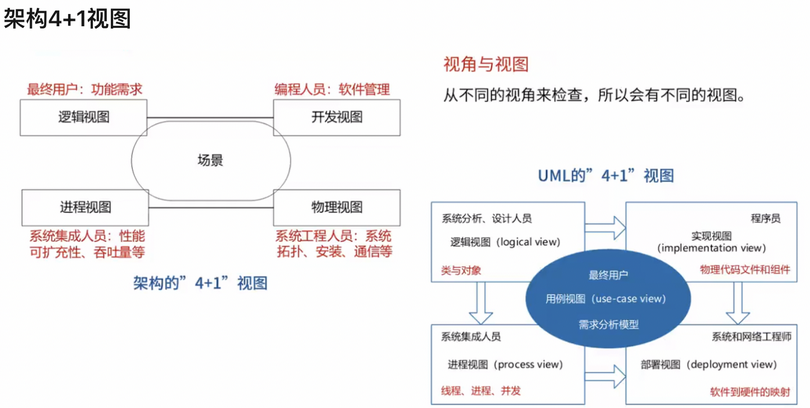
Documenting Software Architectures ，2nd，Paul Clements https://wiki.sei.cmu.edu/sad

Software Architecture: a Roadmap

特别注意会议INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE 的相关文章 (2023年的会议网址<https://icsa-conferences.org/2023/>)

1. 对比书上各种软件体系结构风格和视图特点，思考自己项目属于哪种设计风格？

网上搜索最新的软件体系结构资料，如MVC、Kruchten 4+1视图等。



|  |  |
| --- | --- |
| **五大架构风格** | **子风格** |
| 数据流风格 | 批处理序列（数据一个为整体）、管道-过滤器 |
| 调用/返回风格 | 主程序/子程序、面向对象、层次结构 |
| 独立架构风格 | 进程通信、事件驱动（隐式调用） |
| 虚拟机风格 | 解释器、（基于）规则的系统 |
| 仓库风格(也叫数据共享风格) | 数据库系统、超文本系统、黑板系统 |

1. 第五章课后习题14，故障树转割集树练习。

针对自己项目分析、描绘故障树，分解为割集树 （附到最终提交的SAD）。

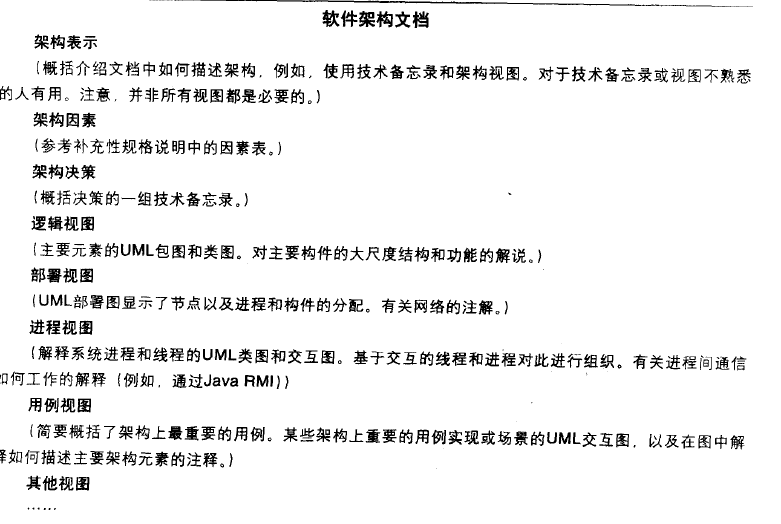
1. 参阅课本和网上资料，研究经典软件体系结构案例KWIC。

An Introduction to Software Architecture，4.1节

针对KWIC和自己项目，参考课本ch5 表5-3，小组成员每人给几种不同的体系结构风格设计打分，评最佳。

1. 补充和修改自己项目的SAD

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。



架构是一组重要决策，涉及软件系统的组织，对结构元素及其组成系统所用接口的选择，从这些结构和行为元素到规模更大的子系统的组成，以及指导该组织结构的架构风格。共同主题涉及动机、约束、模式、职责和系统连接。  
• 软件架构文档SAD文档的结构：  
架构表示  
（概括介绍文档中如何描述架构，例如：使用技术备忘录和架构视图，对于技术备忘录或视图不熟悉的人有用，注意并非所有视图都是必要的）  
架构因素  
（参考补充性规格说明）  
架构决策  
（概括决策的一组技术备忘录）  
逻辑视图  
（主要元素的UML包图和类图，对主要构件的大尺度结构和功能的解说）  
部署视图  
（UML部署图显示了节点以及进程和构件的分配。有关网络的注解）  
进程视图  
（解释系统进程和线程的UML类图和交互图，基于交互的线程和进程对此进行组织，有关进程间的通讯如何工作的解释）  
用例视图  
（简要概括了构架上最重要的用例，某些构架上重要的用例实现或场景的UML交互图，以及在图中解释如何描述主要构架元素的注释）  
其他视图

学习笔记：

（一）软件体系结构的定义

在《Software Architecture in Practice》一书中，对于软件体系结构的定义是“The software architecture of a system is the set of structures needed to reason about the system. These structures comprise software elements, relations among them, and properties of both.”即软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束构成。

软件体系结构概念的提出最初是为了解决从软件需求向软件实现的平坦过渡问题，但是现在已经逐渐扩展到软件开发的整个生命周期。其包括包括有需求阶段的软件体系结构、设计阶段的软件体系结构、实现阶段的软件体系结构、部署阶段的软件体系结构，后开发阶段的软件体系结构。

（二）软件体系结构的主要组成

软件体系结构主要包括构件、连接件、约束三个部分。他们之间的关系可以简单的表述为：按照系统的性能约束或者功能约束，用连接件将构件组装成软件系统。

①构件：构件是具有某种功能的可复用的软件结构单元，表示系统中主要的计算元素和数据存储。构件是一个抽象的概念，在程序中可以指程序函数、模块、对象、类等等。

②连接件:连接件：表示了组件之间的交互。简单的连接件有：管道，过程调用，事件广播等等。复杂的连接件有：客户-服务器通信协议，数据库和应用之间的SQL连接等等。

（三）软件体系结构的重要性

软件体系结构的作用犹如建筑工程中的“施工图纸”，没有图纸，不能开工。同样的，软件体系结构是软件开发过程中的蓝图，没有软件体系结构设计，就无法构造复杂的软件系统。

软件的体系结构影响着软件质量属性。软件的质量属性是外部可见的，例如，安全性、可用性、延迟时间。不同的软件体系结构在对相同的数据对象处理时，有时会有明显的优劣之分，如KWIC中几种不同的风格使得对于相同文本的处理最后所耗费的时间和资源是不相同的。因此，挑选恰当的软件体系结构会更容易满足软件的质量属性。通过SA分析能够及早预知系统的质量属性，从而允许架构师和项目经理对软件开发成本和进度进行提前的推理与预测。

通过软件体系结构的设计极大地促进设计的重用性和代码的重用性，并且使得系统的组织结构易被理解，从而使得开发人员将注意力集中在组建的组装上，而不是简单的、重复的在各个项目上创建组件。因此，通过软件体系结构设计，提升了开发人员的创造力，同时也降低了设计系统与系统的复杂性。

1.逻辑视图（The Logical Architecture View）

逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。逻辑视图常用的风格是面向对象风格，通过抽象、封装和继承等方法，可以用对象模型来代表逻辑视图、用类图来描述，与此同时，还可以使用状态图、协作图来描述逻辑视图。

2.开发视图（The Development Architecture View）

开发视图要考虑软件内部的需求，如软件开发的容易性、软件的重用和软件的通用性，要充分考虑由于具体开发工具的不同而带来的局限性。开发视图所用的风格通常是层次结构风格，其通过系统的输入输出关系的模块图和子系统图来进行描述。

3.进程视图（The Process Architecture View）

进程视图描述系统中的进程，其侧重于系统的运行特性，强调并发性、分布性、系统集成性和容错能力。进程视图可以描述成多层的抽象，每一级别关注不同的方面。

最高层可以看做一组独立可执行的通信程序的逻辑网络，第二层可以看做单个进程，第三层则是任务级的。

4.物理视图（The Physical Architecture View）

物理视图考虑如何将软件映射到硬件上，同时还需要考虑软件的非功能性需求，例如说可用性、可靠性（容错程度）、性能等等，通常包含部署图等。

5.场景（Scenarios）

“4+1”中的“1”，用于刻画构件之间的相互关系，可以看作是那些重要系统活动的抽象，将四个视图有机地联系起来。场景可以描述一个特定的视图内的构件关系，也可以描述不同视图间的构件关系。场景采用文本、图形表示皆可。

**一、什么是软件体系架构**  
 软件体系架构，我们首先用简单的望文生义法来进行想象。第一，“软件”，我们知道这个东西和软件有关。软件是一系列按照特定顺序组织的计算机数据和指令的集合，其与软件有关，我们猜想可能是对这个数据和指令的集合的一种排列。“体系”，告诉我们这个东西不只是单单的某一个事物，既然要成体系，起码要有一个相对大的基数和一定的层次结构，我们所熟知的体系，如人体的免疫体系，一个公司的体系，一个国家的体系，整个自然界这个体系。这些又有一些共同的特点可以让我们去总结，他们分工明确，他们各自之间达成了一种相对的平衡，保持着整个体系的活力和正常运行。“架构”，并不是那么容易理解，我们每个人都对架构有着自己的理解，却又都不够全面，换句话说，我们都对架构这个概念有着理解，却又很难用自己的一句话来进行描述。到目前看来，软件体系架构，就是一个充满着计算机数据和指令的体系，这是我们没有加入架构的概念，这样会使这个定义或者说结论看起来很奇怪，我们就要从架构的这个点进行入手，因为看起来它很关键，甚至足以影响到软件体系架构的定义的合理性。  
**二、软件体系架构是如何产生的**  
 要说软件体系架构是如何产生的，首先要从架构入手，看架构是如何产生的，因为软件体系看起来是用来修饰架构的。架构最早的出现就是因为问题规模的不断扩大，导致问题不好解决，由架构来对问题进行分解。王概凯先生先生在他的架构漫谈中如是说到：架构就是根据要解决的问题，对目标系统的边界进行界定，并对目标系统按某个原则的进行切分。切分的原则，要便于不同的角色，对切分出来的部分，并行或串行开展工作，一般并行才能减少时间，再对这些切分出来的部分，设立沟通机制，使这些部分之间能够进行有机的联系，合并组装成为一个整体，完成目标系统的所有工作。我们在此基础上做一些简单的分析，首先，架构是用来解决问题的，而且，第一步就是对问题的边界进行确定，第二步就是对问题的边界内部进行划分，第三步是对这些切分好的部分建立起关联，最后一步是将各个部分合并到一起，重新组装成为一个整体。其中，最为关键的一步，是第二步，如何对架构进行切分，如果切分的不合理，就会为后续的工作带来压力，因为切分的不合理，导致各部分之间无法形成联系。分解将大问题拆解成一个个小问题，这样解决的难度自然就大幅降低了。  
**三、软件体系架构的作用**  
 软件体系架构的存在就是为了解决大规模的问题，将大规模的问题切分一个个小的功能模块，有各个开发小组完成功能，在最后进行组装拼接成一个完整的系统。这正是我们在开发过程中所常用的模式，切合软件公司的开发流程和体系。分工明确，整个开发过程基本是并发执行，大大缩减的开发所需时间，降低了开发难度，提高了开发效率。分工明确，各个小组在按照架构师的设计进行实现，达到了平衡。这正和我们最一开始所分析的不谋而合，完全诠释了软件体系架构的定义。  
**四、软件体系架构的优化**  
 了解过一些算法的人都知道，算法的优化，要么是在设计思路上进行改变，要么是想办法降低实现过程中的时间复杂度。就像循环嵌套和双指针之间的优化，如果需要循环多次的话效率太低，我们能不能只用一次循环就解决问题，也就是说，我们可以对问题再进行切分，再分配到各个人，再次将串行变为并行，就像是一个递归，递归终止条件就是问题规模无法再切分，这样的优化思路很简单，但也存在问题，当问题切分到一定程度后，无法再切分，我们应该怎么办。我们就要从其他的角度来想办法，比如，某两个功能模块的功能相似，我们可不可以把其中相同的部分单独封装，实现代码的复用。又或者从设计的角度来进行改进，也就是方向的选择上，包括实现系统所用的技术选型，哪种技术开发起来效率更高、或者哪些技术是我们用的比较熟。或者该技术之前开发中使用过，有一些现成的组件可以拿来使用，这些都是优化的思路和方向。  
 总结一下，软件体系架构，就是通过对问题规模的切分来降低开发难度，改串行为并行，提高开发效率，最后完成组装。