第一章

* 软件危机是指落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致计算机软件的开发和维护过程中遇到的一系列严重问题。
* 软件体系结构层次：一般软件系统可分为系统、子系统、模块、类多个层次。其中，（系统——子系统——模块——类）层面一般称为体系结构层次。数据结构等称为编码层次。二进制代码称为可执行代码层次。
* 软件体系结构风格：管道过滤器风格、调用返回风格、正交与分层风格、共享数据风格。

第4章

* 质量属性包含：可用性、可靠性、性能、安全性、可修改性、易用性。

第6章

* 可用性战术根据其目的可分为：错误检测、错误恢复、错误预防
* 错误检测技术：ping 与 echo战术、heartbeat战术、Exception战术
* 错误恢复战术：Voting战术、主动冗余、被动冗余、备份、shadow操作、检查点/回滚
* 错误预防战术：从服务中删除战术、事务（一般概念）、事务（JTA）、事务（Spring）、进程监视器

第7章

* 可修改性战术可分为：局部化修改、防止连锁反映、推迟绑定时间
* 局部化修改是指在设计期间为模块分配任务，用以把预期的变更限制在一定的范围之内。一般具备以下几个子战术：维持语义的一致性、抽象通用的服务、预测期望的变更、泛化模块等。
* 推迟绑定时间战术：包括运行时注册、配置文件、多态、依赖反转（Ioc）、组件更换、遵守已定义的协议、面向切面编程（AOP）

-------------------------------------------------------------------------------

-----------------------------------------------------------------*下面是小道消息*

* 软件危机是指落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致计算机软件开发和维护过程中遇到的一系列严重的问题。一方面是软件成本的增长，……
* 慢工出细活在软件行业并不成立
* 软件工程是用工程、科学和数学的原则与方法研制、维护计算机相关技术及管理方法。
* 工程实践使得平凡的操作者创造出复杂的系统
* 软件工程三要素：方法、工具、过程
* 软件工程的方法，包括结构化方法、面向对象方法和形式化方法。
* 结构化方法具有两个特点：

1. 强调自顶向下顺序地完成软件开发的各阶段任务
2. 结构化的方法要么面向行为，要么面向数据，缺乏使两者有机结合的机制

* 软件工程过程的两方面内涵：

1. 软件工程过程是指为获得软件产品，在软件工具支持下由软件工程师完成的一系列软件工程活动。
2. 从软件开发观点看，它就是使用适当的资源为开发软件进行的一组开发活动。

* RUP初始阶段里程碑——生命周期目标里程碑、细化阶段——生命周期结构里程碑……
* 软件工程解决如何工程化开发软件项目，而软件体系结构则是在工程化开发项目之初规划、设计软件的架构，从而引导软件工程朝着正确的方向发展。
* 软件架构要满足：修改性、易用性、性能等多种非功能属性。
* 经典的软件体系结构，包含了管道过滤器、调用返回、层次等体系结构形态，经典软件体系结构产生于软件规模较小的时代。诸如P2P、C/S基于SOA架构成为现代软件体系结构的代表。
* 软件体系结构的理想效果是可完整的、高效地重用整个软件体系结构，将现有体系结构应用于新的项目中。
* 对于程序员新手而言，在一个框架中进行开发，更像是在一个半成品的积木模型上进行再创造。
* 迪杰斯特拉认为软件应该注意（分解与结构化）
* “4+1”视图模型从5个不同的视角包括：

逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件体系结构。

第三章

* 软件体系结构风格指的是某种特定的模式或者样例
* 经典软件体系结构风格是在早期、规模较小的系统中常被使用的一类软件体系结构风格的总成，包括管道过滤器风格、调用返回风格、正交与分层风格以及共享数据风格。
* ……经过处理将结果置于输出接口中，这样的部件称为过滤器。
* 管道过滤器由管道和过滤器两个部分组成，其中管道是用于传递数据流的通道，过滤器则是具体处理数据的相关计算单元。
* 批处理是串行的，管道过滤器是并行的。
* 若同一时刻所有的节点同时进行处理，且新的节点也不断进入，这种流水线似的处理方式称为管道过滤器。反之，只有等所有节点都……称为批处理。
* 在软件设计中，如果两个组件或者模块不存在互相调用的关系，可以将这两个模块引申为正交。这些互不交叠的功能应该处于同一层次。
* 正交体系结构包含层和线索两个部分。
* 系统被分配给多个小组，之间需要明确的接口主要是**数据模型**和**数据接口**。
* 正交模式的好处体现在现在将整个系统由上而下对功能进行纵向划分，通过纵向的分制使得每一部分相对独立，便于独立式的开发或者累加式的开发。正交模式的弊是因为各子系统之间相对独立，一旦需要数据共享则只能在最底层或者最上层共享。顶层共享效率不高。
* 共享数据的体系结构风格包括了数据库模式和黑板模式。
* 黑板模式：指系统根据中央数据单元当前的状态确定由哪个过程执行。
* 显式调用：调用者掌握了被调用者的引用，并直接进行调用。
* 隐式调用最大的区别在于新的调用方式可保障调用者与被调用者解隅，为程序提供灵活性和支持。
* 消息机制使得消息发送端无需了解消息最终接收端的状态。
* 消息机制的实现一般称为消息总线或者消息中间件，调用者也可称为消息发布者，被调用者称为消息订阅者。
* SOA是面向服务的架构，各服务之间是可互操作、独立、模块化、位置明确、松耦合的，并且可以相互调用，不依赖其它系统。
* 松耦合的好处：代码灵活、容易控制代码影响范围。
* Web服务分为基本服务和组合服务。服务组合必须具有高可靠性、高可用性和高适应行，以服务增值为主要目标。
* Web抽象层次分为：硬编码模式、模型驱动模式……
* 静态服务组合其实就是一个软件复用的过程，组合服务流程的开发人员根据应用的需求，将已有的web服务进行组合创建，利用现有模块实现新的功能。
* 动态服务组合是在执行过程中根据需求和所要调用的web服务功能来动态的选择服务构建。

======第四章P62

* 质量属性一般也称为非功能属性。
* 非功能属性（质量属性）则可认为是附加在相关功能之上。
* 质量属性按照分类，包括可用性，性能性，安全性，易用性，可修改性等。
* 性能简单的说是系统对请求的响应时间
* 安全性是系统对正常异常请求的防护能力
* 可修改性是系统应对需求变更请求所需的修改代价
* 衡量一个系统应对系统修改请求的修改代价，称为修改性
* 易用性是可用性的一个重要方面
* 性能是计算机系统及应用永恒的话题
* 可用性是系统的平均可使用时间，一般采用可用时间的比例进行描述。
* 可靠性是在给定的时间间隔和给定时间下，系统能正确执行其功能的概率。
* 可靠性的量化指标是周期内系统平均无故障运行时间，可用性的量化指标是周期内系统无故障运行的总时间
* 一般提高可靠性的同时，也同时提高了可用性，提高**可靠性**需要强调减少系统**中断**故障的次数，提高**可用性**，需要强调减少从灾难中**恢复**的时间。
* 一个硬盘的性能指标：

1. 主轴转速，
2. 寻道时间，
3. 硬盘表面温度，
4. 道至道时间，
5. 全程访问时间，
6. 最大内部数据传输率

* 非功能需求是指软件产品为满足用户业务需求及功能需求，而必须具有的且除功能需求以外的特性。
* 需求文档模板中的**用例图**，主要体现**功能需求**

第5章

* 质量属性的识别一般靠开发人员的个人经验
* 质量属性场景分为一般场景和具体场景
* 所谓的质量属性战术，正是为控制响应而提出的
* 系统对自己的反应时间可以从系统内部和系统外部元素进行分析
  + 系统内部因素可归结为系统的资源消耗程度及资源争用等
  + 外部因素，包括系统外部的请求特征，包括频度和强度
* 一般的性能战术可以分为资源需求类，战术，资源管理类战术，资源仲裁类战术
  + 资源需求类战术有三种：
    1. 减少所处理事件的数量
    2. 降低单个事件的处理资源，或减少处理事件所需的时间
    3. 控制现存资源的使用
* 在降低事件频度方面，分为主动降低和与被动降低两个方面
  + 主动降低事件频率，主要从更改设计角度考虑，通过分析原有事件产生来源，对可降低频率而不影响应用功能的事件源进行更改，通过事件源自身的频率下降，达到降低事件频度的目标
  + 被动降低事件频率的方法，一般是在无法进行主动降低时才使用
  + 降低事件的处理资源，一般采用提升计算效率以及减少计算开销的方法
  + 为降低事件处理资源，还可采用降低计算开销的方法，该方法主要体现在减少资源需求或者不必要的环节
* 除了上述两种方法之外，采用控制资源使用的方法，也同样可以达到降低资源需求的目标，控制资源的使用及通过强制限制相关执行时间或者，执行方式从而达到降低资源消耗的目的
* 资源管理类的战术，即通过对资源的管理，从而保障相关性能，一般包括并发战术，维持数据或计算多个副本，增加可用资源等
* 计算多个副本战术，主要体现为在多个地方保持同一个计算内容，或者将一个计算内容分配给多个分离的模块进行处理
* 在设计中，将原本属于服务器端的一部分计算能力转移到客户端进行计算，从而降低服务器端的请求量或者复杂度
* 数据副本战术及保持多份类似的数据，从而降低由于请求数据而带来的IO开销
* 使用缓存主要有以下两个原因，减少相应延迟，减少网络带宽消耗
* 数据副本战术需解决各个数据副本间同步的问题，缓存的数据被操作之后，若没有及时更新相应的原始数据，可能导致数据不一致
* 增加可用资源，包括增加相关的CPU，内存，网络等
* 资源仲裁类战术：先进先出方式是为了保障用户请求处理的有序性，可利用队列等方式保障用户请求的处理顺序
* 先进先出的队列方式，保障了请求的有序执行，然而却忽略了请求的重要性，无法保障重要请求的执行时效，固定优先级策略的提出，就是为了解决这个问题
* 动态优先级战术，动态优先级是在固定优先级策略之后提出的，是在固定优先级的基础上进行的改进
* 时限优先级最早优先战术，实现时间最早优先是另一种调度方法，重点考察请求的最后时限
* 静态调度，静态调度指的是在系统上线前，事先离线分配相关资源的调度方法

======第6章======待补充

* 可用性：系统正常执行时间的比例，等于“系统平均工作时间 /（系统工作时间+系统错误时间+系统维修时间”。
* 代码由于人为因素写错了或者考虑不周全，成为了**错误；故障**是系统出错后导致系统不正常工作的结果；**BUG**是被激发出来的错误，是故障的总称。
* 可修改性战术预测期望的变更，一个系统中可能进行修改的部分如下：

业务规则、对硬件、软件系统的依赖、非标准语言的特性、状态变量、模块间的交互、质量目标可能改变、运行环境。

* 各个模块都为某个功能而串接，表现为模块间的依赖关系过于紧密，模块失去了相关的独立性。
* 仲裁者模块根据其所在位置以及所起的作用可以分为数据仲裁者和服务仲裁者。
* 典型的数据依赖性包括存储库、消息中间件等。
* 典型的服务仲裁者则是用于隔离服务之间由于接口等产生的依赖关系。
* 静态调度，是指在系统上线前实现离线分配相关资源的调度方法。

待续。。。。。。。。。。。。。。。。。。。。。