definitions of SA

architecture = components + connectors + constraints；

Architecture business cycle ABC

ABC是一个影响的循环，从环境影响到架构，架构亦反作用于环境；

1.业务环境影响架构设计决定；

2.软甲架构反过来影响业务环境；

1.影响组织结构； 软件单元（元素）--》团队组织

2.影响组织的业务目标； 新市场--》新机遇--》适合的架构

3.影响顾客需求； 架构如生产线--》更快投入市场--》改变需求

4.影响软件将来是如何设计的；

3.架构设计关系到开发过程中的每一步；

1.软件架构是软件开发过程中重要的一步；

2.软件架构知识便于业务用例；

3.需求管理；

4.软甲架构需要确保实现忠实于架构设计；

4.密切影响组织的底线；

1.与利益相关者交流的关键；

2.架构决策影响开发循环；

3.构架限制软件细节；

4.构架影响计划表和预算；

5.合适的构架有助于保证质量；

架构师的职责

1.引导技术方向；

1.将客户需求转换为技术设计；

2.确保技术设计满足质量需求；

3.与客户达成一致；

4.回顾代码，确保架构与代码实现一致；

5.开发者的指导；

2.管理冲突；

1.确定利益相关者；

2.想利益相关者征求需求和期望；

3.不可预见困难最小化；

1.成功案例学习帮助寻找最佳解决办法；

2.基于构架的开发便于发现问题；

3.构架评估确定最佳设计；

·架构风格的决定元素

·组建

·连接件

·组建拓扑结构

·拓扑或者行为的约束

·对风格的非正式描述

·很少有纯粹的风格，风格彼此重叠，一个系统通常表现出多种风格。

·数据流风格（DataFlowStyle）

·特点：

1.由数据是否可用控制着计算

2.系统结构有数据的移动决定

3.结构清晰

·组建：计算单元

1.接口是输入端口与输出端口

2.输入端口读数据，输出端口写数据

·连接件：数据流

1.单向

2.通常是异步的

3.通常需要缓冲

·拓扑：任意的

批处理（Batch sequential）

特点：1.配个处理步骤是一个独立程序2.每一步必须在前一部结束后3.数据整体方式传递

例子：早期编译器、CASE

管道（Pipes-and-Filter)

Filter：1.读入、产生数据流2.运行数据流对其进行增、删、转

Pipes：1.转移数据 2.单向流动

特点：1.数据到来便被处理，不用收集完在处理；

2.过滤器是独立实体

3.运行的流中无上下文关系

4.过滤器物上下文关系

批处理与管道风格对比

同：1.把任务分解成一系列固定计算单元2.组件间只通过数据流传递来交互

异：批：全部 延时 无交互

管：增加的 即时 经常交互

·控制 特点：1.控制点怎样在程序间移动2.数据跟着控制走，不起推动作用

·调用返回风格

·历史

1.主程序和子程序

2.功能模块

3.抽象数据类型

4.对象

5.面向对象架构

6.组建

·主程序和子程序

特点：1.层次分解；2.单线程控制；3.层次推理；4.子系统是隐藏式的

·功能模块

模块化优点：1.分而治之；2.降低耦合；3.复杂系统被分解成几个易于

理解的模块

·对象构架

优点：封装、继承、多态、交互

问题：大量对象带来管理问题（额外的结构，规则和操作）

大量功能单一接口式交互复杂

自责分配使系统难以理解

·KWIC

·事件风格 （通过系统之间的潜在调用完成）

优点：1.问题分解（对象更独立）2.系统维护和重用3.性能（并行）4.稳健性

缺点：1.调用无序，引起歧义

2.系统维护和重用（event的引入需要额外的控制）

3.性能（间接导致额外开销）

·数据共享风格

·数据共享系统（包括共享存储的数据及访问其的元素）

1.这种风格描绘很多系统（depend on共享数据本身的特种）

2.风格解决收集、操作、保存大存储数据

3.这种风格容易增加数据的生产者和消费者

·各种仓库

1.设计者预先设计好的

2.输入流的信息类型决定

3.系统其它部分的信息决定

4.随机：计算状态决定（黑板）

·黑板模型

·特点：知识在一些写作的进程之间共享和分布；机会主义的处理数据；

·三个主要的组建：知识源；黑板；控制；

·知识源：把问题分成几个部分，每个部分独立计算；响应黑板变化；

目标：提供解决问题的知识

代表：过程、规则、逻辑断言

行为：修改黑板

·黑板：全局数据库包含 解域 的全部状态；知识源交互的唯一媒介

目标：保存知识源要用的数据

代表：来自解空间的数据

组织：分层

·控制：让知识源机会地响应

目标：让知识源机会地响应

代表：了解知识源的能力，决策解决问题的步骤

·层次风格

·特点

使用下层服务，服务上层，只能见到自己的邻接层

逐步解决问题，隐藏复杂

变更影响小

上层指导下层身份，顺序不能颠倒

层层调用，影响性能

·C/S P2P

1.功能需求与非功能需求

2.功能性与SA

·功能可以通过多种不同的架构实现；

·功能可以用人一种结构实现；

·质量需求更依赖于系统架构；

3.功能压倒质量（比如系统被频繁更新并不是因为质量缺陷）；

以下重点

·三种主要质量属性：系统质量属性；业务质量属性；架构质量属性；

·质量属性场景：一种描述“质量属性”特征的方法，由六部分构成。

·刺激源

·刺激

·环境

·制品

·响应

·响应度量

·系统质量属性

1.可用性（availability）

·关注：系统故障及后果

·质量属性场景：刺激源；刺激；制品；环境；响应；响应度量；

·可用性策略（架构策略就是这些策略的集合）

1.错误检测（防止错误变为系统故障，进行可能的修复）

错误检测三种策略：

1.命令/响应：组建发出ping，从期待组建接收echo。

2.心跳：源组件周期性的发出信息，如不发出，则被认为出错。

3.异常：·几种错误类·发现错误类即导致异常·处理:继续、语义转换。

2.错误恢复

恢复策略三种：

1.shadow操作：

2.状态再同步：

3.检查点/回滚：·checkpoint ·回滚--根据--checkpoint和事务日志

3.错误预防

1.免除服务

2.事务汇报

3.process monitor

2.可修改性

·关注：变更代价

·质量属性场景：刺激源；刺激；制品；环境；响应；响应度量；

·可修改性策略：

1.局部化修改战术（将变更限制在一小部分模块上，通过合理分配职责达到）

1.维持语义一致：

语义就是模块中职责间相互关系。

目标：职责单一，不依赖其他模块。

2.预期期望变更

变更预测帮助职责分配。

使变更的影响最小化。

3.泛化模块：不解释

4.限制可能选择

2.防止连锁反应战术

1.添加新的接口

2.限制通信路径

3.中间件使用

3.推迟绑定时间

1.运行时注册

2.配置文件

3.多台

4.组建更换

5.遵守已定义协议

3.性能

·关注：响应时间的时间

·质量属性场景：

·性能策略：

1.资源需求

1.提高计算效率

2.减少计算开销

3.管理时间率

4.控制采样频率

5.限制执行时间

6.限制队列大小

2.引入并发

1.并发

2.维持多个副本

3.增加可用资源

3.资源仲裁

1.FIFO

2.固定优先级

3.动态优先级

4.静态调度

4.安全性

·实现安全性能策略

1.抵抗攻击

1.对用户身份验证

2.对用户授权

3.维护数据机密性

4.维护完整性

5.限制暴露信息

6.限制访问

2.检测攻击

将访问模式与已知攻击模式比较

3.从攻击中恢复

1.通过可用性中的策略来恢复

2.通过追踪确定入侵者

5.可测试性

6.易用性