-软件架构概念

-模型与建模方法 （4+1视图模型：逻辑、进程、物理、开发）

-***风格***与模式

-***基于架构***的软件开发

-架构演化与维护

***软件架构***

定义：软件架构包括构件（component），连接件，约束（基本元素），端口和角色

**构建和交互关系的集合（组成派）、重要设计决策的集合（决策派）**

思想：注重软件系统设计图，包含软件设计决策（**组织结构设计**）

**特征**：**重用性**、利益相关者多、关注点分离（**模块化**，分治）、质量驱动（**非功能性需求**）、概念完整性、循环风格

发展阶段：

1、基础研究阶段：Modular（模块化，逻辑切分）

2、概念体系核心技术形成：Software architecture，component（**组件化**，功能分隔）

3、理论体系丰富：基于组件软件架构（架构的通用描述方法，演化和重用）

4、软件架构理论完善与实践

架构描述：ADL、形式化方法、4+1架构、UML、IEEE软件架构描述

架构分析：结构分析（开发风格选择），功能分析，非功能分析

设计方法：工作，用例，模式，领域，属性驱动

测试：仿真系统测试

***架构模型***

对架构设计决策具象化和文档化

建模方法：非规范图形（盒线图），UML，形式化建模（符号化），基于UML形式化

基于图形的可视化建模：树形结构，树地图（Tree map），旭日图，双曲树

UML：表示法统一，支持多视图，**建模能力不强，语义精确性不足**，软件架构风格无法体现 建模阶段：当作架构语言直接建模，扩展机制约束丰富元模型，扩充元模型

形式化建模：Z语言，Petri网，B语言，CSP（新语言，难学难用）

UML形式化：用符号语言对UML进行规约（类，用例，状态，顺序）

文本语言建模：文档描述整体架构，无图形结构不易理解

Model Driven Architecture（模型驱动架构建模）：构造平台无关模型（PIM）验证可行并移植至平台相关模型（PSM）

**发展趋势**

***风格与模式（style & pattern）***

架构风格：**描述特定领域中系统组织方式的惯用模式，独立于语言**

通用，总体结构框架

**设计元素**，**配置规则**，**语义解释**

设计代码重用性，组织结构易理解，内部互操作性，需求和风格分析

**风格分类**：**数据流**、**调用返回**、**独立组件**（进程通信）、**虚拟机**（解释器）、**仓库**

1. 管道过滤器：输入-管道-过滤器(功能模块)-管道-输出（高内聚，低耦合）
2. 主，子程序风格：逐步分解细化，调用返回（数据访问效率高，可复用性低）
3. 面向对象：对象维护自身完整性，对象之间**相互隐蔽**，类标识调用**独立性低**
4. 层次化：系统组织为若干层次，层间存在接口，**通过接口形成服务**（**问题分解**，支持扩展，重用 OS）
5. 事件驱动：监听广播事件触发过程（独立，可复用性强 监听响应组件无控制权，无数据交换，正确性验证困难）
6. 解释器风格（Interpreter）：逐条边翻译边执行（程序可移植，**平台无关性，**效率低 JVM）
7. 基于规则的系统：提取频繁变化的业务逻辑形成**规则库**（跨场景）
8. 仓库风格：两种组件：**中央数据存储**组件，相对独立组件（可扩展性好 需有**同步机制**防止数据死锁）
9. 黑板系统：**知识源**-**控制器**-**黑板**（数据结构）-控制器-知识源（知识源之间相互独立，仅与黑板交互）
10. C2风格：通过**连接件绑定**在一起的**组件网**，通信和处理分开（分层处理和消息同步，**多用户** 大规模流式风格时不适用）
11. 客户机/服务器：两层CS（客户机-服务器） 三层CS（客户机-**应用服务器**-数据库服务器）Client-Sever
12. 浏览器/服务器：BS风格是对三层CS风格的改进方式之一，通过**浏览器**通过访问web服务器实现对web，数据服务器访问（客户端占用资源少，操作简单，开发成本低 个性化程度低，交互性不强，服务器负担重，可扩展性差安全性低，数据库查询速度低）Browser-sever
13. 平台/插件： platform平台-平台插件接口-plug-in插件（降低模块依赖性，模块独立开发与维护，动态组装/分离 可重用性差）
14. 面向Agent风格： Agent组件/连接件 具有自主，智能，交互等特性，根据所处环境智能进行行为（分布开放结构 缺乏自身社会性结构描述和环境交互）
15. 面向方面Aspect：系统中一些需求横切于系统，考虑**横向关注点（方面）**形成组件与连接件（定义交叉关系，层次化，面向对象 将一类事封装为一个方面）
16. 面向服务：**服务提供者-注册中心(UDDI)-服务请求者**（灵活性，复用性，以业务为核心 服务划分与编排困难，接口标准难以统一）
17. 正交架构：层（Layer），线索（Thread），组件（Component）组件通过线索调用不同层的组件（正交），线索间互不相关（完全正交）
18. 异构风格：多种架构通过层次组合，允许单一组件有复合连接件（遗留代码重用，解决解释习惯上差异 不同风格兼容问题解决困难BS-CS内外/查改有别）
19. 层次消息总线**H**ierarch **M**essage **B**us：基于层次的消息总线，支持构件的**分布**和**并发**，构件（单一/复合）通过网络与消息总线连接（构件耦合，灵活性高，支持动态演化 **重用要求高**）
20. **MVC：视图**（用户界面）**-控制器**（控制交互）**-模型**（核心数据，算法），主要应用于用户交互程序设计（良好的移植性，便于维护 增加复杂性，系统效率低）

架构模式：根据系统的结构组织定义系统族以及组件件关系

领域，平台相关性强，反应**最佳解决方案**

***架构描述语言***

ADL架构模式：任何用于软件架构的表示形式**（**组件，连接件，架构配置，约束条件）

C2：用户界面密集系统，按照特定形式描述架构，方便机器读取

**为何有多种描述语言**

***架构与敏捷开发***

**软件架构与敏捷开发均为了提高开发效率，软件质量，降低软件成本最大化价值**

软件开发：松散开发->软件工程思想->敏捷开发->智能化开发

敏捷开发(宣言)：需求，设计，编码，测试（可运行软件，客户合作，响应变化 迭代增量，持续集成）

**敏捷架构设计：种子设计（开发前设计，可扩展性，骨架与轮廓），详细架构设计（分解架构设计过程，尽早集成，降低开发风险）**

初始设计：得到原始架构（全局抽象层次设计，达成关于项目周期目标协议，确定业务和需求风险）

迭代过程：**迭代设计**（根据工作任务进行需求分析，设计，编码），**重构**（梳理架构，进行持续改进），**确定架构**（测试软件），**客户交流**（交流反馈，完成新需求），应对需求**不可预测性**，长期计划不稳定，单次迭代短期计划是稳定的

**设计思想：**

团队设计：**群体决策**，**避免理论完美**但难以实现的架构设计（原始架构 结论周密，避免个人遗漏 沟通成本，决策效率，责任不明确）

简单设计：**表达方案和现实抽象的简单化**（降低开发成本，提高沟通效率，适应与稳定性）

**迭代过程：**

XP：简单设计（发布计划，分配任务），结对编程（根据任务编码）,测试驱动开发（debugs），发布（确认需求与反馈，文档化）

Scrum：计划会议（Sprint backlog）->每日例会->燃尽图（预期进度与实际进度）->回顾会议（发布）

FDD：开发设计，设计包（并发独立开发），完成（根据特征构造）

**敏捷开发如何改变软件设计方式**

**架构驱动软件开发：**

**需求获取，架构设计，文档化，架构评估，实现，维护**

架构需求（功能/非功能）：

需求来源(讨论架构层次需求，不需直接深入功能性需求)，需求获取（用户群，环境与架构经验->需求获取->**特定质量场景**），**质量模型描述**（对**质量场景**进行描述，**评估**与预测质量属性 满足用户期望）

基本架构设计：**功能->架构需求列表**，开发子系统，发布（从0开始软件开发）

架构文档化：架构设计**完整且可追溯**的，利益相关者间联系，制定满足需求架构

架构结构：对架构的描述，系统可靠，可用，安全性分析（基于**功能**（功能需求分解），**代码**（关键性代码抽象描述 组件：包，类…），**并发**（线程与进程控制描述），**物理**（物理设备描述，系统可用性与性能），**开发**（文件目录与团队分工））

架构评估：分析架构识别**架构风险**，验证已满足质量需求（架构权衡分析法：质量属性会受其他质量属性影响而变差，综合评估）

实现与维护：架构与开发团队组织结构一致性，文档维护

***架构设计与实现***

**架构品质：模块化，适应需求/技术变化，系统动态运行与数据规划，明确灵活的部署规划**

**需求分析到架构设计：包含架构概念的需求规约，系统架构高层表述**

难点：描述语言转换，非功能需求难以规约，一致性难复杂度高，需求庞大

需求与架构相互影响形成螺旋式开发

详细设计：架构表达式细化，数据结构与算法选择（语言相关 接近源码的软件表示）

**Model Driven Architecture模型驱动架构**：计算无关模型（CIM业务模型，描述系统外部行为），平台无关模型（PIM高抽象，实现技术无关），平台特定模型（确定实现技术）

设计原则：一般（数据，应用程序，技术），关键原则（关注点分离，单一职责，松耦合，功能分离）

**架构设计威胁**：忽略非功能需求（全面认识需求），频繁变化需求（**关键需求**决定架构），非全面架构设计（多方面视角），不及时架构验证，创造性架构比重（加强经验架构使用），低可执行性（避免过度设计）

示例：

项目背景，需求分析（功能（use-case细化）/非功能（质量场景）），架构设计（整体架构）

**MDA基本思想与其优点**

***体系结构评估：***

了解体系结构的质量属性，屏蔽风险

**质量属性**：**可修改**（功能，容量扩展），**可用**（可正常运行时间占比），**可测试**（缺陷易被查出），**易用**（用户体验与效率），**安全性**（对非法操作防范），**性能**（响应速度）

评估作用：降低开销，挖掘隐性需求

评估方式：调查问卷（评估灵活自由 主观性强），**基于场景**（相对主观 领域依赖性高，需丰富领域知识），基于度量（通过度量值判断架构质量 较为客观，对体系具体情况精确了解）

Architecture Tradeoff Analysis Method：体系结构权衡分析（敏感点：构件特征（质量目标注意点），权衡点：多个质量属性的敏感点）（基于场景）

**质量场景描述**：刺激，环境，响应

ATAM评估步骤：ATAM方法陈述，业务动机，体系结构，体系结构方法，效用树（系统整体质量（效用-质量属性-属性求精-场景（优先级 场景重要性，实现难度））），分析体系结构，集体讨论确定优先级，陈述结果

Software Architecture Analysis Method：分析体系结构可修改性，系统属性与功能

评估步骤：场景形成（集体讨论），体系结构描述（静态/动态特征），场景分类优先级划分（直接/间接场景），单个场景评估（间接场景评估，提出修改），场景交互评估，总体评估

**ATAM与SAAM概念优缺点**

***架构演化与维护：***

架构生命周期：初始设计，实际使用，修改完善，弃用

架构演化：维持架构自身有用性（**组件，连接件，约束的增删改**）

**静态演化**：设计时与运行前（非运行态），软件架构设计，实现，维护

步骤：软件理解（提取系统抽象表示），需求变更分析，演化计划（分析原系统，选择演化计划），系统重构，系统测试

**动态演化**：软件内部执行或外部请求导致的结构改变与重配置

演化类型：交互->结构->架构动态性

属性改名，行为变化，拓扑结构变化

**演化原则**：成本控制（演化/重新开发），进度可控，风险可控，主体维持（软件主体行为稳定，变更信息类型/量稳定），平滑演进（变更率稳定），总体结构优化（可靠性，性能），目标一致（阶段/最终目标 技术债），模块独立（修改局部化），复杂性可控，利于重构，利于重用，适应新技术，质量向好，适应新需求

软件架构维护：架构知识管理（架构知识图谱）