概述：

**架构思想：关注软件系统设计图与设计决策**

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

**组成派：组件和关系的集合 决策派：重要设计决策的集合**

一般定义：组件，连接件，配置（约束）/基本元素，端口，角色

架构模型：

对架构设计决策具象化与文档化

建模方法：非规范图形，UML（建模能力不强，语义精确性不足），形式化方法（新语言 难掌握），基于UML的形式化方法（UML符号语言）

MDA模型驱动架构建模：构造平台无关模型PIM并移植到平台相关模型PSM

风格与模式：

模式：定义系统族及组件关系，反应最佳解决方案

风格：特定领域中系统组织方式惯用模式，独立于语言

设计元素，配置规则，语义解释（代码重用性，组织结构易懂性，内部互操作性，需求与风格分析）

风格分类：数据流，调用返回，独立组件，虚拟机，仓库

**二十种风格**

架构描述语言：

概念：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）

**为何有多种ADL**：每一种ADL都以独立的形式存在， 描述语法不同且互不兼容，同时又有许多共同的特征，这使设计人员很难选择一种合适的ADL

敏捷开发：

敏捷：需求，设计，编码，测试（个体与互动，可运行软件，客户合作，响应变化，迭代增量持续集成）

**敏捷开发与软件架构均是一个权衡的过程，为了提高开发效率、软件质量、降低开发成本、最大化价值**

敏捷架构：种子架构设计（可扩展性骨架与轮廓），详细架构设计（分解架构设计，尽早集成降低风险）

初始设计（规划式）：得到原始架构 迭代过程（演进式）：迭代设计，重构，确定架构，客户交流（需求不可预测性）

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），简单设计（表达方案与实现的抽象）

**敏捷对架构的影响**：

◦ 敏捷开发非常重视软件的架构设计，但是轻架构的详细设计

◦ 敏捷思想中将传统的架构设计分成：种子架构设计+详细架构设计。

◦ 种子架构设计关注软件系统的骨架或轮廓的设计

◦ 敏捷开发将详细架构设计转移到Code编码阶段、 重构阶段、单元测试阶段等

架构驱动开发：

步骤：需求获取，架构设计，文档化，架构评估，实现，维护

质量场景：抽象场景（软件需求描述），特定质量属性场景

质量模型：质量元模型的实例化，对质量属性定义、评估、预测，提高质量属性

质量元模型：质量模型中元素间关系

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，详细部署规划

**软件架构概念引入需求分析**：

问题：传统方法不考虑软件架构概念与原则，需求规约与架构映射困难

优势：确保需求与架构的一致性与可追踪性，提升需求完整性与结构化

协同演化：螺旋双峰过程，需求影响架构设计，架构设计使需求明确与细化

详细设计：架构表达式细化，数据结构与算法选择

映射问题：缺少架构视图，方案笼统无技术蓝图，层次间缺乏交互接口，过度设计

解决方法：重视架构视图，深入到技术相关，明确交互接口，保持扩展与可维护性基础上防止过度设计

MDA模型驱动架构：计算平台无关（CIM系统外部行为），平台无关模型（PIM高抽象，实现无关），平台特定模型（PSM具体技术实现）

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）

体系结构评估：

质量属性：可修改，可测试，可用性，易用性，安全性，性能

评估时期：前期（划分需求等级），后期（明确质量及功能需求）

评估方式：调查问卷，场景（对系统使用或修改的支持程度 需丰富的领域知识），度量（为软件属性赋予数值）

场景：刺激（系统交互的引发），环境（系统状态），响应（系统对刺激反应）

ATAM体系结构权衡分析：敏感点（构件特征），权衡点（多个质量属性敏感点）

评估步骤：陈述（信息交流），调查分析（关键质量属性需求 质量属性效用树（效用-质量属性-属性求精-场景）），测试（结果检验），形成报告（ATAM陈述结果）

SAAM体系结构分析方法：分析体系结构可修改性，系统属性及功能

评估步骤：场景形成（集体讨论），体系结构描述（动/静态特征），场景分类优先级划分（直/间接场景），单个场景，场景交互，总体

优缺点：

SAAM基于场景的评估需要大量专家知识，主观性强，没有对软件架构提供清晰的度量，适用于粗粒度的评估

ATAM从不同角度探究系统特性，对架构的完整性及风险决策提供支持，探讨质量属性权衡，但未对质量属性深入分析缺乏定量数据

架构演化与维护：

生命周期：初始设计，实际使用，修改完善，弃用

架构演化：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，需求变更分析，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动态重配置（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，风格变化）

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