一、Introduction（ch1）

1、体系结构定义：

基于 1993 年 D Garlan,Mshaw 的软件体系结构定义，可以认为软件体系结构由三大要素组成，分别为：组件，连接件，约束。其作用如下：

- 组件：具有某种功能的可重用的**软件模块单元**，表示了系统中主要的**计算单元**和**数据存储**。
- 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。复杂的连接件有：客户－服务器通信协议，数据库和应用之间 SQL 连接等。
- 约束：表示了组件和连接件的拓扑逻辑和约束。

2、什么是好的软件架构?

（1）、使软件处于稳定状态，可以处理故障和错误

（2）、使软件易于修改，同时保持软件的稳定性。

（3）、设计软件架构，使软件在未来易于扩展功能。

（4）、使软件易于操作，主要包括软件的组件的维护和管理。

（5）、尽可能低成本制作软件

（6）、不要使软件过于复杂，避免较高的实施及运作成本，以及保持软件的可维护性、可扩展性

3、软件工程

（1）、定义：软件工程是用工程、科学和数学的原则与方法研制、维护计算机软件的有关技术及管理方法。

（2）、软件工程的三个要素：方法、工具和过程。

- 方法：完成软件工程项目的技术手段
- 工具：工具为软件工程方法提供自动或半自动的软件制成环境
- 过程：过程是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的

4、软件开发人员组成

- 编程级别:程序员、开发人员、编码人员、数据库管理员
- 软件架构级:软件架构师、系统架构师
- 项目管理级别:项目/组/部门经理

二、Software Architecture Style （五个架构风格）

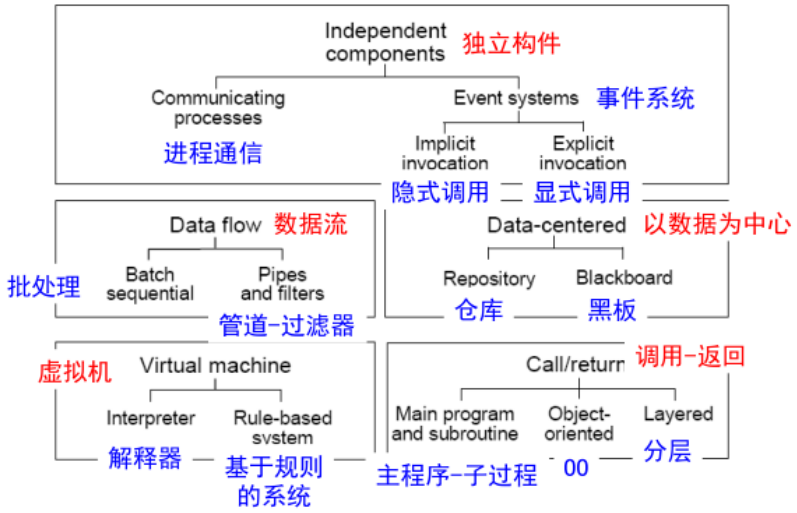
1、 软件体系架构风格定义：

- 描述一类体系结构
- 独立于实际问题，强调了软件系统中**通用的组织结构**
- 在**实践**中被多次设计、**应用**
- 是若干设计**思想的综合**
- 具有已经被熟知的特性，并且可以**复用**

2、 对于软件体系架构风格的知识点：

组件、连接件、拓扑结构、语义约束、优缺点

【一个系统可以由多种风格组成】



三、Software Architecture Modeling and Documentation(2)

描述质量属性，设定场景优先级->根据质量属性场景优先级，通过 ADD 方法设计软件架构风格，并通过策略实现质量属性响应的目标->进行文档化描述->进行评估

1、 文档编制的 7 个原则

- 从读者的角度来写：为作者方便而写的文件标志；让信息更容易找到!
- 避免不必要的重复：每种信息都应该准确地记录在一个地方；这使得文档更容易使用和更改；重复常常会使人混淆，因为重复的信息方式略有不同。
- 避免歧义：精确定义的符号/语言有助于避免所有类型的歧义。
- 使用标准的组织方式。
- 记录的基本原理
- 保持文档的更新，但不要太过更新
- 审查文件的适用性

2、结构和视图

结构：存在于软件或硬件中的架构元素的实际集合

视图：视图表示一组元素以及这些元素之间的关系。由系统涉众编写和读取。

3、视图

视图是一种管理复杂性的方法。

每个视图都可以用来回答关于体系结构的不同问题

4、Architectural Structures

（1）、“模块”结构——由被称为模块的实现单元元素组成。

- **Decomposition view** – shows modules that are related via the "is a sub-module of" relation
- **Uses view** – shows modules that are related via the "uses" relation (i.e., one module uses the services provided by another module)
- **Layered view** – shows modules that are partitioned into groups of related and coherent functionality. Each group represents one layer in the overall structure.
- **Class/generalization view** – shows modules called classes that are related via the "inherits from" or "is an instance" of relations

- 分解视图：通过"is a sub-module of"关系显示相关的模块
- 使用视图：显示通过"使用"关系相关的模块(例如，一个模块使用另一个模块提供的服务)
- 分层视图：显示被划分为相关和一致功能组的模块。每一组代表整体结构中的一层。
- 类/泛化视图：显示被称为类的模块，这些类通过关系的"继承自"或"是一个实例"进行关联

（2）、"组件和连接器"结构——由运行时组件(计算单元)和它们之间的连接器(通信路径)组成。

- 进程视图：显示通过通信、同步和/或排除操作连接的进程或线程
- 并发视图：显示组件和连接器，其中连接器表示"逻辑线程"
- 共享数据(存储库)视图：显示创建、存储和访问持久数据的组件和连接器
- 客户端-服务器视图：显示合作的客户端和服务器以及它们之间的连接器(例如，它们共享的协议和消息)

（3）、"分配"结构——由软件元素及其与创建和执行软件的外部环境中的元素之间的关系组成

- 部署视图：显示软件元素及其对硬件和通信元素的分配
- 实现视图：显示软件元素及其在开发、集成和配置控制环境中的文件结构的映射
- 工作分配视图：显示模块，以及它们如何分配给负责实现和集成它们的开发团队

（4）、视图 IEEE

- 一个架构描述是由一个或多个视图组织起来的。
- 视图由一个或多个模型组成，并符合一个视点。
- 架构描述选择一个或多个视点
- 视点覆盖一个或多个涉众关注点。一个视点为一个或多个模型建立方法

（5）、Kruchten 的 4+1 观点

逻辑视图:支持行为需求。关键抽象，是对象或对象类

进程视图:处理并发性和分布。将线程映射到对象。

开发视图:软件模块、库、子系统、开发单元的组织。

物理视图:将其他元素映射到处理节点和通信节点上。

"加一个"视图:将其他视图映射到重要的用例上，以显示它们如何工作。

四、质量属性

五、Software Architecture Design (ch5)

1、架构师主要使用以前尝试过的解决方案来创建新的架构

2、实现质量属性的策略（6 个属性的策略选择）

3、Design Operators

（1)、定义：设计操作符是创建架构设计的基本设计工具

– Decomposition 分解
– Replication 复制
– Compression 压缩
– Abstraction 抽象
– Resource Sharing 资源共享

（2)、分解:

定义:

- 分解是将不同的功能划分为具有良好定义的接口的不同组件的操作
- 分解将一个系统分成两个或多个系统(称为子系统)。
- 分解用于实现各种质量属性，包括可修改性、可移植性、性能和可构建性

类型:

- 部分/整体分解：将系统划分为一组在功能上不重叠的子组件。部分/整体组成可分为均匀分解和非均匀分解。
- 统一分解将系统划分为一组本身结构相似的组件
- 泛化-专门化分解，组件之间存在潜在的重叠功能

（3)、复制

定义:

- 复制一个部件以提高可靠性和性能的操作。
- 冗余，一个组件有几个相同的副本同时执行。
- n 版本编程，其中相同功能有几种不同的实现。

冗余:

冗余：一个组件有几个相同的副本同时执行。

- 静态复制:备份组件在需要时才执行任何工作(例如当主组件发生故障时)，对于正常操作而言，其性能比运行时复制更好，但在响应故障时较慢
- 运行时复制:复制的组件同时执行相同的功能，以性能为代价提高可靠性

N 版本编程:

- 同时运行冗余组件来执行一个函数，并使用一些算法来确定多个结果中哪个是正确的(有时使用投票，有时使用收到的第一个结果)

（4)、压缩

定义：是分解的反义词

- 压缩涉及到将组件合并为单个组件或删除组件之间的层或接口
- 压缩的目的是通过消除某种间接级别来提高性能

（5)、抽象

- 抽象通过引入语义丰富的服务层来隐藏信息，同时隐藏实现细节
- 也称为创建虚拟机。虚拟机不一定是编程或脚本语言的复杂解释器。一般来说，任何隐藏某些系统实现的东西都被认为是虚拟机。这包括 JDBC 或 ODBC 等接口

- 抽象：尤其是虚拟机——也可以用来提高适应性

（6）、资源共享

- 封装数据或服务，以便在多个独立的客户端组件之间共享它们
- 增强的可集成性、可移植性和可修改性。

4、Architectural Drivers

（1）、体系结构是由一些功能、质量和业务需求的集合“塑造”的。我们称这些塑造需求为架构驱动

（2）、要确定体系结构的驱动因素，请确定最高优先级的业务目标。将这些业务目标转化为质量场景或用例。从这个列表中，选择对体系结构影响最大的那些

5、Attribute-driven Design (ADD)

（1）、ADD 是一种循序渐进的方法，用于系统地生成系统的第一个架构设计。

（2）、ADD 结果：总体结构决策-互连和协调机制-模式和战术在体系结构的特定部分的应用-明确的质量属性需求的实现-不是详细的接口

（3）、ADD 要求作为输入:—质量属性要求—功能要求—约束

（4）、过程

第 1 步:确认有足够的需求信息

第 2 步:选择要分解的系统的一部分

第 3 步:优先考虑需求并确定架构驱动因素

第 4 步:选择设计概念——模式、风格、策略——满足与我们选择分解的系统部分相关联的架构驱动因素。

步骤 5:实例化架构元素并分配功能

第 6 步:合并到目前为止完成的设计

第 7 步:分配剩余的功能

第 8 步:为实例化的元素定义接口

第 9 步:验证和细化需求，并使它们成为实例化元素的约束

第 10 步:为希望分解的系统的下一部分重复第 2 步到第 9 步

六、Architecture Evaluation（ch6）

1、定义：架构评估是一种开发生命周期活动，其中几个涉众使用场景等评估技术，在一个正式或非正式的过程中一起分析软件架构

2、几种不同的评估方法

（1）、Scenario-based Architecture Analysis Method（SAAM）:

是第一个文档化的软件体系结构分析方法，最初开发它是为了分析体系结构的可修改性。但是，它对于分析体系结构的任何非功能方面都很有用。它建立在使用涉众生成的场景来评估架构的基础上。【SAAM 没有明确地处理质量属性之间的交互】

（2）、Architecture Trade-off Analysis Method (ATAM).

是 SAAM 的继承者。该方法将质量属性效用树和质量属性合并到体系结构的分析中。ATAM 处理质量属性之间的交互。因此，权衡是关于竞争质量属性的。ATAM 是 SAAM 的专门化，特别关注可修改性、性能、可用性和安全性。

（3）、SAAM Founded on Complex Scenarios (SAAMCS).

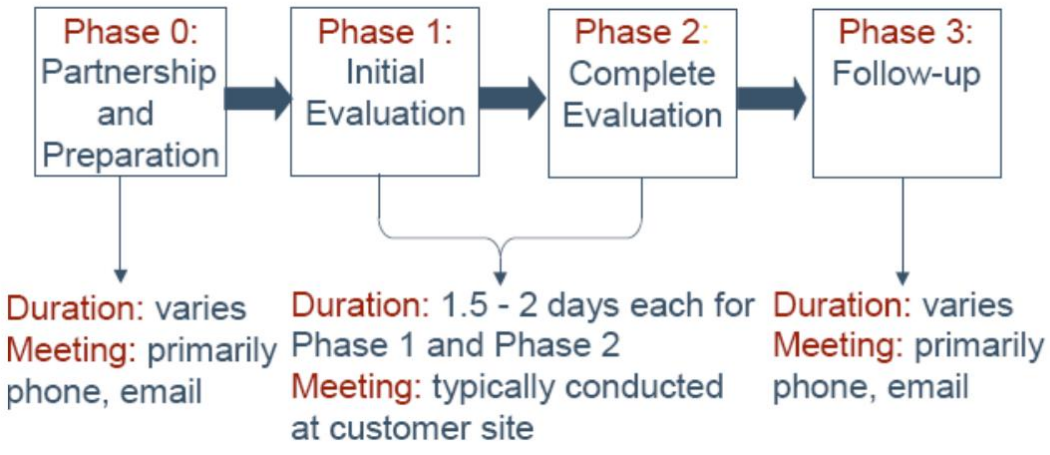
该方法将评估场景的复杂性作为最重要的风险评估因素

（4）、Extending SAAM by Integration in the Domain (ESAAMI)
该方法将 SAAM 与特定领域和基于重用的软件开发过程集成在一起。domain-specific and reuse-based
（5）、Software Architecture Analysis Method for Evolution and Reusability (SAAMER)
这种方法特别关注进化和可重用性的质量属性 evolution and reusability
（6）、Scenario-Based Architecture Reengineering (SBAR)
该方法利用场景、仿真、数学建模和基于经验的推理来评估质量属性。该方法还包含一个架构设计方法
（7）、Architecture Level Prediction of Software Maintenance (ALPSM)
这是使用场景分析可维护性的另一种方法，称为更改场景，表示维护任务
（8）、软件架构评估模型(SAEM)
This method is based on formal and rigorous quality requirements.

3、 ATAM 目的
（1）、ATAM 是一种帮助涉众提出正确问题以发现潜在问题的体系结构决策的方法。
（2）、ATAM 的目的不是提供精确的分析…其目的是发现由架构决策产生的风险。
（3）、希望找到趋势:体系结构决策和系统属性预测之间的相关性

4、 ATAM 优点
- 识别风险
- 澄清质量属性需求
- 改进架构文档
- 为架构决策编制文档基础
- 增加涉众之间的沟通

5、 ATAM 过程



（1）、phase0：在技术评估之前。
- 客户和评估团队的一部分交换对要评估架构的方法和系统的理解。
- 达成执行评估的协议。
- 派出核心评估小组。

（2）、phase1：涉及一小群主要面向技术的参与者
阶段 1 以体系结构为中心，对体系结构的信息进行自上而下的分析

1. Present the ATAM
2. Present business drivers
3. Present architecture
4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches

1、Present the ATAM：**评估团队展示了 ATAM 的概述**，包括 ATAM 步骤，技术（效用树生成；场景头脑风暴/映射），输出（架构方法，效用树和场景，风险、非风险、敏感点和权衡）

2、Present Business Drivers：ATAM **客户代表介绍系统的业务驱动**，包括：系统的业务上下文，**高级功能需求，高级质量属性需求**

3、Present Architecture：**架构师展示了架构的概述**，包括：技术约束，如指定使用的操作系统、硬件或中间件，系统交互的其他系统，用于处理质量属性需求的体系结构方法，评估团队开始探测和捕获风险。

4、Identify Architectural Approaches：确定主要的架构方法；评估者开始确定体系结构中对实现质量属性目标至关重要的位置

5、Generate Quality Attribute Utility Tree：通过构建一个实用工具树，识别、优先排序，并细化最重要的质量属性目标。工具树是一种自上而下的工具，用于描述“驱动”特定属性需求，并对其进行优先级排序。驱动质量属性的是高级节点(通常是性能、可修改性、安全性和可用性)。场景是实用程序树的叶子。输出:对特定质量属性需求的描述和优先级排序。

场景应涵盖以下范围:用例场景:预期的系统使用;增长场景:预期的系统更改;探索性场景:系统未预料到的压力

- Use case scenario
  - *Remote user requests a database report via the Web during peak period and receives it within 5 seconds.*
- Growth scenario
  - *Add a new data server to reduce latency in scenario 1 to 2.5 seconds within 1 person-week.*
- Exploratory scenario
  - *Half of the servers go down during normal operation without affecting overall system availability.*

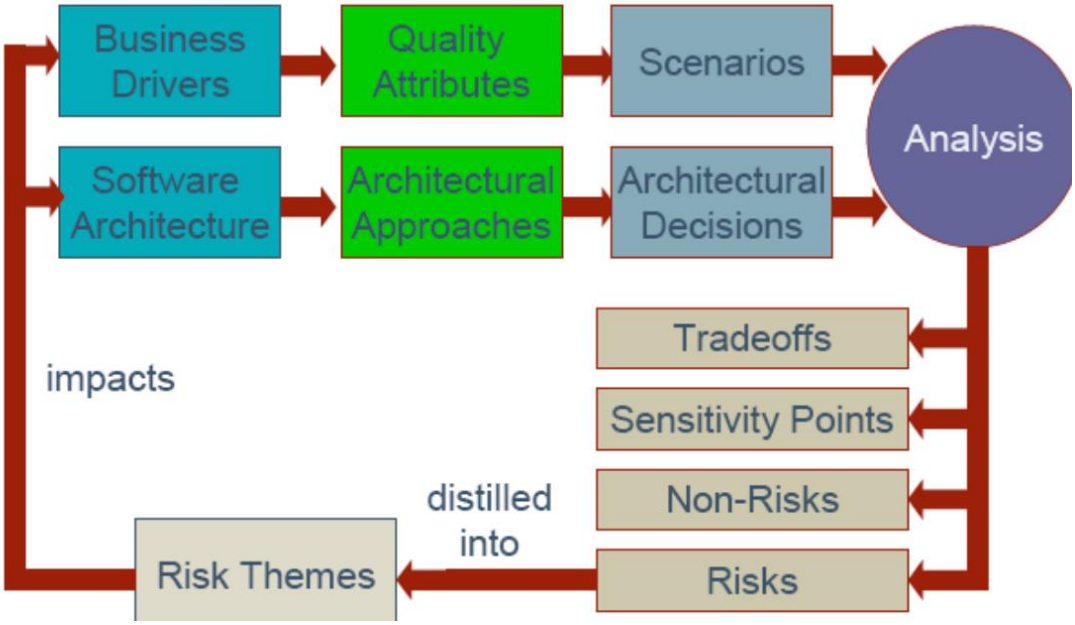
6. Analyze Architectural Approaches：评估团队从特定质量属性的角度探索体系结构方法，以识别风险。确定体系结构方法；为最高优先级的场景提出质量属性特定问题；确定并记录风险和非风险、敏感点和权衡

（3）、阶段 2:涉及到更大的利益相关者群体

第二阶段以涉众为中心，着重于引出涉众的不同观点，并验证第一阶段的结果

7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results

8、确定受上一步生成的场景影响的体系结构方法。继续识别风险和非风险
9、概括 ATAM 的所有步骤并展示 ATAM 的输出，包括—架构方法—实用工具树—场景—风险和非风险—敏感性点和权衡—风险主题



（4）phase3:主要是为客户生成最终报告，以及对评估和 ATAM 材料的质量进行反思

ATAM 是一种根据多种质量属性评估体系结构的方法；一种发现体系结构决策结果的有效策略；一种确定趋势的方法，而不是执行精确分析的方法