Software Architecture

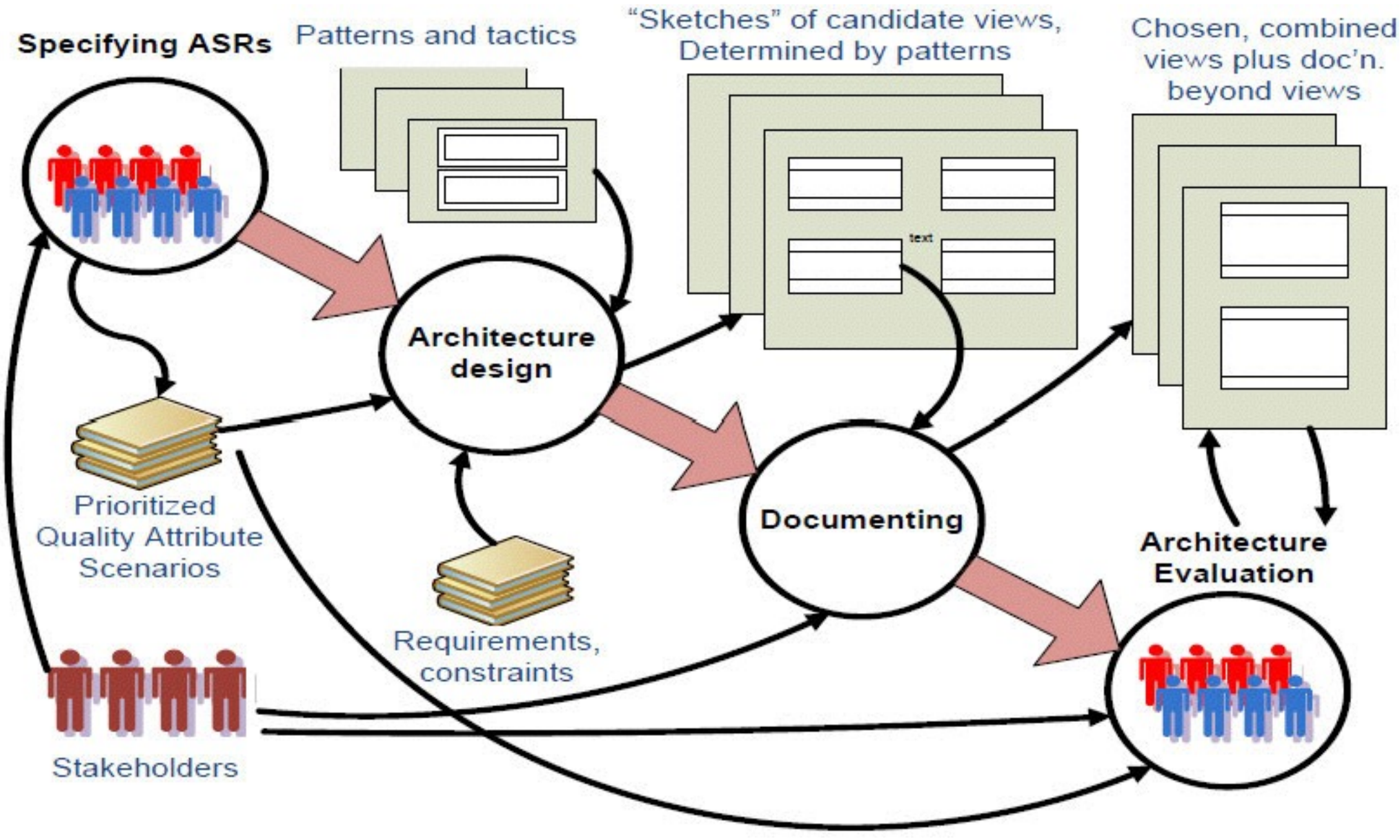
Outline

- Software Architecture
- Quality Attributes
- Designing Software Architecture
- Documenting Software Architecture
- Evaluating Software Architecture
- Software Product Lines
- Model Driven Architecture
- Service Oriented Architecture

Software Architecture in General

- What is software architecture?
  - Structure, Elements, Relationships, Design
- What does a software architect do?
- Where do architectures come from?
  - NFRs, ASRs, Quality Requirements; Stakeholders, Organisations, Technical Environments...
- Architecture Views
  - Logical view, Process view, Physical view, Development view...
- Architectural activities and process
- Software architecture knowledge areas

Architecture Process



Quality Attributes

- Software Requirements
  - Functional requirements, Quality requirements (NFRs), Constraints
- Quality Attributes
  - Modeling quality attribute scenarios: Source, Stimulus, Artefact, Environment, Response, Measure
  - external&internal
    Availability, Interoperability, Modifiability, Performance, Security, Testability, Usability, X-ability...
  - Tactics for quality attributes
- Architecturally Significant Requirements
  - How to gather and identify ASRs: Requirements, Interviews, Business goals, Utility tree
    QAW

架构模式

Architecture Patterns

Architecture Patterns

问题发生的情景　　对pattern的描述

Context, Problem, Solution: elements + relations + constraints

三类pattern

Module Patterns 考察静态的结构

Layered pattern

Component-Connector Patterns 系统/软件运行起来 架构内部相互之间的关系

Broker pattern, Model-view-controller pattern, Pipe-and-filter pattern, Client-server pattern, Peer-to-peer pattern, Service-oriented pattern, Publish-subscribe pattern, Share-data pattern

Allocation Patterns 部署/从软件系统和周围环境的关系出发来考察架构

分布计算的架构形式　　和上面的分层要分清（分层：和部署无关，是软件系统内部静态、逻辑上的结构）和运行平台相关，是软件系统和物理世界相互之间的关系

Map-reduce pattern, Multi-tier pattern

Patterns vs. Tactics

pattern可以包含其他的pattern 也可以包含其他的tactics
此处是基本的
可以创建自己的pattern （在讲过得基础之上添加其他的pattern/tactics进去）

比如：使用broken pattern提高其互操作性，但是其会在availablity security performance有隐患 风险，可能要对此pattern进行修改，修改之后形成自己的架构。

（我们把战术看作是设计的基本“ 构建块 ”，并根据该战术创建架构模式和策略）

Designing Architecture

Design Strategy

- Decomposition（分解） and iteration（迭代）, Generation（生成） and test（验证）

Attribute-Driven Design (ADD)

- Choose a part to design 找到一个部分进行设计
- Marshal all ASRs for that part 找到跟这个部分相关的所有ASR
- Create and test a design for that part
- Inputs to and outputs of ADD
- 更准确/详细的ADD方法（八个步骤）8-step process: 1. confirm requirements, 2. choose an element to decompose, 3. identify ASRs, 4. choose a design satisfying ASRs, 5. instantiate（实例化） elements & allocate（分配功能） responsibilities, 6. define interface, 7. verify（验证） & refine requirements, 8. repeat step 2-7 until all ASRs satisfied 此步骤可细分

Documenting Architecture

Views and Beyond

information beyond views

- Views:

  视图分为两大类
  结构视图（module c&c allocation视角产生的视图）对应产生的pattern上去
  从某个质量属性的角度（比如关注performance时，就会把与performance无关的设计tactics去除掉。关注某质量属性进行设计分析时产生的质量视图）

- Styles, patterns and views
- Structural views: module views, component-and-connector views, allocation views
- Quality views

质量视图和结构视图合在一起 就是我们要document的。但是这些视图如何去选择？（每一次ADD过程都会产生 对于一个复杂系统来说可能很庞大）——看哪些视图对于大的stakeholder来说都是共同关注的。以这些视图为基础，把其他的视图合并到其中去。再进行排序

- Documenting views: 1. build stakeholder/view table, 2. combine views, 3. prioritise & stage

视图之上的信息，一个主要的内容就是这些视图之间的关系。通常从一个单独的视图上并不能表现出来视图之间的联系。比如 一个element可能会出现在多个视图上——mapping
视图所表现出来的系统和周围环境（context）的关系？

- Beyond views: documentation info & architecture info (mapping between views)

视图&information beyond views合在一起，就是架构的归档。

- Documentation package: views + beyond

Evaluating Architecture

通用的架构评估的方法

ATAM: Architecture Tradeoff Analysis Method

Stakeholders involved in ATAM

都有哪些stakeholder会参与ATAM的不同阶段的不同步骤中

Inputs to and outputs of ATAM

不同阶段不同步骤的input 和output

Phase 0: Partnership & preparation

Phase 1: Evaluation - 1

1（架构设计人员 项目管理人员 评估人员）和2的主要参与人员基本一样，2的更多

六个步骤有不同的人做不同的事情 input 和output不一样

架构师实现架构

1. present ATAM, 2. present business drivers, 3. present architecture, 4. identify architectural approaches, 5. generate utility tree, 6. analyse architectural approaches

由评估团队/评估者告诉方法（评估过程中要做什么事情）

由项目管理人员说明整个项目的情况，对组织起到什么作用

Phase 2: Evaluation - 2

更多的stakeholder参与进来，目的是看第一个阶段的结果和第二个阶段相互之间看 是否有遗漏

主要通过头脑风暴

1. present ATAM & results, 7. brainstorm & prioritize, 8. analyse architectural approaches, 9. present results

对于不同stakeholder提出的senario进行排序。最终发现没有在第一个阶段被评估到的一些具体的质量情景。对遗漏的部分进行进一步的评价

Phase 3: Follow-up

前两阶段的结果进行归档

Software Product Lines

Software Product Lines (Engineering)

- Product = core assets + custom assets
- Reusability and Modifiability

关注的重点

Product Line Architecture

重用&为单独的SPL中某产品留有变化的余地——产品线开发中最重要的两个策略

架构设计中强调reuse 和 variation（对应上面两个属性）

- Reuse: find, understand, and use (invoke)

（具体到可以 做些挖掘 买 让第三方做——找到可以被reuse的部分）

六种不同形式的变化（YES/NO-是不是要使用这个element 数量不同 是否可以对其进行参数化-用参数来控制变化 通过定义一个统一的interface但是有不同的实现来体现不同的变化，使用hook机制，让系统能够根据运行时不断变化的产品来自己产生变化）变化发生在三个层次（架构层次 设计层次 最终实现的文件层次）

- Variation: forms of variation * software entity varied * binding time

6*3*5=90种 variation选择

5个时间点上的变化 binding time上（生成代码的时候 编译的时候 推到link的时候 在系统启动的时候根据注册表的设置情况来变化 在运行的时候动态改变）

- Architecture: variation points

SPL Practice Areas and Patterns

基于SEI产品线开发的方式

- 29 practice areas and 22 patterns

29种不同的practice 对应三个category ，22个pattern

Model Driven Architecture

模型驱动开发

关键的思想：把功能和实现进行分离。

Model Driven Development (MDD)

- Separates the specification of functionality and the specification of implementation
- Reusability, Interoperability, and Portability

Abstraction Levels 实现的时候 把产品分成了三个层次上的模型

- CIM (computation independent model), PIM (platform independent model), PSM (platform specific model)

计算无关 平台无关 平台相关

三个层次上可以进行转换（加入更多的信息）

OMG Standards 和MDA相关的一些标准

- UML, MOF (meta-object facility) specifies any modelling language, XMI, QVT (query-view-transformation)...

在横向模型进行转换（比如建模语言都依据这个标准 可以实现互操作性）

实现纵向的（portability）比如从PIM-PSM

Service Oriented Architecture

reusability & interoperability&modifiability

Service Find-Bind-Execute Paradigm

主要的思想：先找服务，再绑定服务，最后使用服务

SOA vs. Web Service

区分概念（SOA是一种架构设计的思想，并不一定非要web service来实现，只是现在普遍用web service这样一个实现技术）

SOA vs. Enterprise Service Bus (ESB)

ESB是给SOA的实现提供基础设施。能够帮助SOA更快/更有效的实现——提供底层支持（协议上/通讯上）

Service-oriented applications and Infrastructure

某一个应用 和基础设施（比如ESB） 某一个应用组合了不同的service

SOA Implementation

实现

以前没有soa系统，从实现单独的service 再实现整个系统

Top-down: identify operations of services, then implement services

已经有系统，转成SOA架构——分别去重用一些component，开放出来，在这个基础之上去构造基于SOA的应用

Bottom-up: reuse components, then expose existing components as services

Final Exam

- 简答题、论述题、设计分析题
  需要做展开 加入自己的思考和理解
  做些设计（题目/系统）做初始设计
- 英文题目、中文或英文答题
- 个别题目可能需画图
  设计分析题（生成一个视图）
- 基础内容70%
- 高阶内容30%