软件架构概述

1. **定义与重要性**
   - 软件系统架构是关于系统主要设计决策的集合，是软件构建和演进的蓝图。
   - 架构涵盖结构、行为、交互和非功能属性，影响系统全面发展。

2. **"Principal"的含义**
   - "Principal"表示具有一定重要性，赋予设计决策“架构状态”。
   - 并非所有设计决策都是架构的，取决于利益相关者对系统目标的定义。

3. **其他定义**
   - Perry 和 Wolf: 软件架构 ={ 元素, 形式, 基本原理 }，强调"是什么"、"怎样"、"为什么"。
   - Shaw 和 Garlan: 描述了构建系统的元素、元素间的交互、指导组合的模式以及对这些模式的约束。
   - Kruchten: 软件架构处理软件高层结构的设计和实现，包括抽象、分解、组合、风格和美感。

4. **时间方面**
   - 架构具有时间方面，系统在任一时刻只有一种架构，但随时间推移会发生变化。
   - 设计决策在系统生命周期中制定，架构演进是不可避免的。

5. **规范性与描述性架构**
   - 规范性架构捕获系统构建前的设计决策，描述性架构描述系统的实际构建方式。
   - 架构演进时，理想情况下应修改规范性架构，但实践中可能直接修改描述性架构。

6. **架构退化与恢复**
   - 架构漂移是引入不在规范性架构中的设计决策到描述性架构中，而不违反规范性架构。
   - 架构侵蚀是引入违反规范性架构的设计决策到描述性架构中。
   - 架构恢复是从实施级别的工件中确定软件系统的架构的过程。

7. **部署与评估**
   - 软件系统在部署前无法履行其目的，部署视图对系统是否满足需求至关重要。
   - 部署评估维度包括可用内存、功耗、所需网络带宽等。

8. **软件架构的元素与组件**
   - 软件架构是多个元素的组合与相互作用，包括处理、数据（信息或状态）、交互等。
   - 软件组件是架构实体，封装系统功能和数据的子集，通过明确定义的接口限制访问，具有明确定义的依赖关系。

What is Software Architecture?

软件架构是什么？

Definition:
定义：
A software system’s architecture is the set of principal design decisions about the system.
软件系统的架构是关于系统的主要设计决策的集合。
Software architecture is the blueprint for a software system’s construction and evolution.
软件架构是软件系统构建和演进的蓝图。
Design decisions encompass every facet of the system under development.
设计决策涵盖了正在开发的系统的每个方面，包括：
Structure 结构
Behavior 行为
Interaction 交互
Non-functional properties 非功能属性

What is "Principal"?
“Principal”是什么？

“Principal” implies a degree of importance that grants a design decision “architectural status”.
“Principal”意味着赋予一个设计决策“架构状态”的一定重要程度。
It implies that not all design decisions are architectural.
这意味着并非所有的设计决策都属于架构领域。
That is, they do not necessarily impact a system's architecture.
也就是说，它们不一定会影响系统的架构。
How one defines "principal" will depend on what the stakeholders define as the system goals.
“Principal” 的定义将取决于利益相关者如何定义系统的目标。

其他关于软件架构的定义
Perry and Wolf
Perry 和 Wolf
Software Architecture = { Elements, Form, Rationale }
软件架构 ={ 元素，形式，基本原理 }
what how why
是什么 怎样 为什么

Shaw and Garlan
Shaw 和 Garlan
Software architecture [is a level of design that] involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns.
软件架构[是设计的一个层次]，涉及到系统构建的元素描述，这些元素之间的交互，指导它

们组合的模式以及这些模式的约束。

Kruchten
Kruchten
Software architecture deals with the design and implementation of the high-level structure of software.
软件架构涉及软件高层结构的设计和实现。
Architecture deals with abstraction, decomposition, composition, style, and aesthetics.
架构处理抽象、分解、组合、风格和美感。

时间方面
Temporal Aspect
时间方面
Design decisions are unmade over a system’s lifetime.
设计决策在系统的生命周期内没有被制定。
Architecture has a temporal aspect.
架构具有时间方面的因素。
At any given point in time, the system has only one architecture.
在任何给定的时间点，系统只有一种架构。
A system’s architecture will change over time.
系统的架构会随着时间的推移而发生变化。

规范性架构与描述性架构
Prescriptive vs. Descriptive Architecture
规范性架构与描述性架构
A system’s prescriptive architecture captures the design decisions made prior to the system’s construction.
系统的规范性架构捕获了在系统构建之前所做的设计决策。 这是按照预期或意图的架构。
A system’s descriptive architecture describes how the system has been built.
系统的描述性架构描述了系统的实际构建方式。 这是按照实际实现或实际实现的架构。

Architectural Evolution
架构演进
When a system evolves, ideally its prescriptive architecture is modified first.
当系统发生演进时，理想情况下应首先修改其规范性架构。
In practice, the system – and thus its descriptive architecture – is often directly modified.
实际上，系统通常直接修改，从而影响其描述性架构。
This happens because of 这种情况发生的原因包括：
Developer sloppiness
开发人员的粗心
Perception of short deadlines which prevent thinking through and documenting
认为时间紧迫，无法深思熟虑和记录
Lack of documented prescriptive architecture
缺乏规范性架构的记录

Need or desire for code optimizations
对代码优化的需求或愿望
Inadequate techniques or tool support
技术或工具支持不足

Architectural Degradation
架构退化
Two related concepts
两个相关概念
Architectural drift
架构漂移
Architectural drift is the introduction of principal design decisions into a system’s descriptive architecture that are not included in, encompassed by, or implied by the prescriptive architecture but which do not violate any of the prescriptive architecture’s design decisions.
架构漂移是将主要设计决策引入到系统的描述性架构中，这些设计决策不包括在规范性架构中，也不被规范性架构所包容或隐含，但它们不违反规范性架构的任何设计决策。
Architectural erosion
架构侵蚀
Architectural erosion is the introduction of architectural design decisions into a system’s descriptive architecture that violate its prescriptive architecture.
架构侵蚀是将架构设计决策引入系统的描述性架构中，这些决策违反了其规范性架构。

Architectural Recovery
架构恢复
If architectural degradation is allowed to occur, one will be forced to recover the system's architecture sooner or later.
如果允许架构退化发生，迟早会被迫恢复系统的架构。
Architectural recovery is the process of determining a software system's architecture from its implementation-level artifacts.
架构恢复是从实施级别的工件中确定软件系统的架构的过程。
Implementation-level artifacts can be
实施级别的工件可以是：
Source code
源代码
Executable files
可执行文件
Java .class files
Java .class 文件

Deployment
部署
A software system cannot fulfill its purpose until it is deployed.

在部署之前，软件系统无法履行其目的。
Executable modules are physically placed on the hardware devices on which they are supposed to run.
可执行模块被物理放置在它们应该运行的硬件设备上。
The deployment view of an architecture can be critical in assessing whether the system will be able to satisfy its requirements.
架构的部署视图对于评估系统是否能够满足其需求可能至关重要。
Possible assessment dimensions
可能的评估维度
Available memory
可用内存
Power consumption
功耗
Required network bandwidth
所需网络带宽

Software Architecture's Elements
软件架构的元素
A software system’s architecture typically is not (and should not be) a uniform monolith.
软件系统的架构通常不应该是统一的单体（单一块）， 也不会是单一块。
A software system’s architecture should be a composition and interplay of different elements.
软件系统的架构应该是不同元素的组合和相互作用。
Elements include:
元素包括：
Processing
处理
Data, also referred to as information or state
数据，也称为信息或状态
Interaction
交互

Components
组件
Elements that encapsulate processing and data in a system’s architecture are referred to as software components.
在系统的架构中，封装处理和数据的元素被称为软件组件。

Definition
定义
A software component is an architectural entity that
软件组件是一种架构实体，它：

encapsulates a subset of the system’s functionality and/or data
封装系统功能和/或数据的一个子集

restricts access to that subset via an explicitly defined interface
通过明确定义的接口限制对该子集的访问
has explicitly defined dependencies on its required execution context
具有明确定义的依赖关系，依赖于其所需的执行上下文
Components typically provide application-specific services.
组件通常提供特定于应用程序的服务。