Chapter0

抽象化程度越来越大

程序=数据结构+算法

程序=对象+对象

系统=组件+组件

系统=服务+服务

系统=元素+连接器

整体结构>技术细节

品质>功能

一个给定的功能，可以由实现许多不同的架构，具有不同的品质来实现。

质量要求取决于系统架构多于功能要求。

“系统经常重新设计，不是因为它们在功能上的缺陷，通常在功能上是相同的，但是因为它们很难保持，港口，或规模，或者太慢或是被网络黑客入侵。”

架构专注于更高层次的抽象，而不是技术细节

体系结构描述系统的所有视图

SA起源于SE的发展

SA专注于高层次的系统的抽象和系统质量，处理架构的复杂性。

SA文件是审查和详细设计的基础

Chapter1

系统的软件体系结构是系统所需要的结构的集合，包括软件元素，它们之间的关系，以及两者的性质。

结构是由关系在一起的元素的集合。

软件系统是由许多结构组成的。

Structures partition systems into implementation units are called modules.

结构划分系统为实现单元称为模块。

Modules are assigned specific responsibilities, and are the basis of work.

模块分配具体的职责，是工作的基础。

Structures focus on the way the elements interact with each other at runtime to carry out the system’s functions.

结构的重点是在运行时的元素相互作用的方式进行系统的功能。

We call runtime structures component-and-connector (C&C) structures.

我们调用运行时结构组件和连接器（C&C）结构。

In our use, a component is always a runtime entity.

在我们的使用中，组件始终是运行时实体。

Allocation structures describe the mapping from software structures to the system’s environments

分配结构描述从软件体系结构到系统环境的映射

organizational

组织

developmental

发育

installation

安装

Execution

执行

For example

例如，

Modules are assigned to teams to develop, and assigned to places in a file structure for implementation, integration, and testing.

这两个模块分配团队去开发和分配文件结构的位置去实施，集成和测试。

Components are deployed onto hardware in order to execute.

组件在硬件上的作用是为了执行

Decomposition Structure

分解结构

Use structure

使用结构

Layer structure

分层结构

Class (or generalization) structure

类结构

Data Model

数据模型

Service structure

服务结构

Concurrency structure

并发结构

Deployment structure

部署结构

Implementation structure

实施结构

Work assignment structure

工作分配结构

Common component-and-connector type patterns:

通用组件和连接器类型模式：

Shared-data pattern.

共享数据模式。

Client-server pattern.

客户端-服务器模式。

The components are the clients and the servers.

组件是客户端和服务器。

The connectors are protocols and messages they share among each other to carry out the system’s work.

该连接器是彼此共享的协议和消息，以进行系统的工作。

Common allocation patterns:

公共分配模式：

Multi-tier pattern.

多层模式。

Competence center pattern and platform pattern

能力中心模式与平台模式

There are three categories of structures: Module Structures, Component-and-connector Structures, Allocation Structures.

结构有三类：模块结构、组件和连接器结构、分配结构。

Structures represent the primary engineering leverage points of an architecture.

结构是一个建筑的主要工程杠杆点。

Every system has a software architecture, but this architecture may be documented and disseminated, or it may not be.

每一个系统都有一个软件架构，但这种架构可以记录和传播，也可能不。

There is no such thing as an inherently good or bad architecture. Architectures are either more or less fit for some purpose.

没有这样的东西作为一个天生的好或坏的架构。架构是多或少适合一些目的。

Chapter2

13个理由

1.Influence quality attributes.

1.影响质量属性。

2.Help reason about and manage change as the system evolves.

2.当系统的发展有助于改变和管理变化的原因。

3.Eearly prediction of a system’s qualities.

3.一个系统的质量的早期预测。

4.Enhances communication among stakeholders.

4.加强利益相关者之间的沟通。

5.Capture the earliest and hence most fundamental, hardest-to-change design decisions.

5.捕捉最早的，最根本的，最难改变的设计决策。

6.Defines a set of constraints on subsequent implementation.

6.定义了一组在随后的执行上的约束。

7.Dictates the structure of an organization, or vice versa.

7.决定一个组织的结构，反之亦然。

8.Provide the basis for evolutionary prototyping.

8.为进化的原型设计提供依据。

9.Allows the architect and project manager to reason about cost and schedule.

9.让建筑师和项目经理对成本和进度的原因。

10.As a transferable, reusable model that form the heart of a product line.

10.作为一种可重用的可重用模型，该模型是产品线的核心。

11.Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.

11.架构为基础的发展重点关注的组件组件，而不是简单地在他们的创作。

12.Reducing design and system complexity.

12.减少设计和系统复杂度。

13.Be the foundation for training a new team member

13.为培训新的团队成员奠定基础

Chapter3

Architectures exist in four different contexts.

体系结构存在于四个不同的环境中。

Technical. The technical context includes the achievement of quality attribute requirements.

技术。技术环境包括质量属性要求的实现。

Project life cycle.

项目生命周期。

Business.

业务。

Professional.

专业。

An architecture has influences that lead to its creation, and its existence has an impact on the architect, the organization, and, potentially, the industry.

一个架构会影响到它的创造，它的存在对架构师、组织和潜在的产业都有影响。

This is the Architecture Influence Cycle.

这是架构的影响周期。

Chapter4

Requirements for a system come in three categories.

系统要求三类。

Functional.

功能性。

Quality attribute.

质量属性。

Constraints.

约束。

Quality attribute scenario has 6 parts.

质量属性场景有6个部分。

Source of stimulus.

 源的刺激。

Stimulus

刺激

Environment.

 环境。

Artifact.

 工件。

Response.

 响应。

Response measure.

响应措施。

An architectural tactic is a design decision that affects a quality attribute response.

架构策略是影响质量属性响应的设计决策。

Architectural patterns can be seen as “packages” of tactics.

架构图案可以被看作是“包”的战术。

The seven categories of architectural design decisions are:

架构设计决策的七大类：

Allocation of responsibilities

责任分配

Coordination model

协调模式

Data model

数据模型

Management of resources

资源管理

Mapping among architectural elements

建筑元素之间的映射

Binding time decisions

绑定时间决定

Choice of technology

技术选择

Chapter5

Availability refers to the ability of the system to be available for use when a fault occurs.

可用性指的是当一个故障发生时，该系统可供使用的能力。

The fault must be recognized (or prevented) and then the system must respond.

故障必须被确认（或防止），然后系统必须响应。

The response will depend on the criticality of the application and the type of fault

该反应将取决于应用程序的临界点和故障类型

can range from “ignore it” to “keep on going as if it didn’t occur.”

可以从“忽略它”，“继续下去，如果它没有发生。”

Tactics for availability are categorized into detect faults, recover from faults and prevent faults.

可用性策略分类为故障检测，故障恢复和故障预防。

Detection tactics depend on detecting signs of life from various components.

检测策略依赖于各种组件的生命迹象。

Recovery tactics are retrying an operation or maintaining redundant data or computations.

恢复策略是重试操作或维护冗余数据或计算。

Prevention tactics depend on removing elements from service or limiting the scope of faults.

预防策略依赖于从服务中移除元素或限制故障范围。

All availability tactics involve the coordination model.

所有可用性策略涉及的协调模型。

Chapter6

Interoperability refers to the ability of systems to usefully exchange information.

互操作性是指系统有效地交换信息的能力。

Achieving interoperability involves the relevant systems locating each other and then managing the interfaces so that they can exchange information.

实现互操作性，涉及到相关的系统定位，然后管理接口，以便他们可以交换信息。

Chapter7

Modifiability deals with change and the cost in time or money of making a change, including the extent to which this modification affects other functions or quality attributes.

修正了在改变和改变的时间和金钱成本，包括范围，这样的修改会影响其他功能或质量属性。

Tactics to reduce the cost of making a change include making modules smaller, increasing cohesion, and reducing coupling. Deferring binding will also reduce the cost of making a change.

策略，以降低成本的变化，包括模块更小，增加凝聚力，减少耦合。延迟绑定也会降低制造成本变化。

Chapter8

Performance is about the management of system resources in the face of particular types of demand to achieve acceptable timing behavior.

性能是关于管理系统资源的特殊类型的需求，以实现可接受的时序行为。

Performance can be measured in terms of throughput and latency for both interactive and embedded real time systems.

性能可以衡量的吞吐量和延迟的互动和嵌入式实时系统。

Performance can be improved by reducing demand or by managing resources more appropriately.

通过减少需求或者更恰当地管理资源，可以提高性能。

Chapter9

Attacks against a system can be characterized as attacks against the confidentiality, integrity, or availability of a system or its data.

对一个系统的攻击可以被定性为对系统的机密性、完整性或可用性的攻击，或者它的数据。

This leads to many of the tactics used to achieve security. Identifying, authenticating, and authorizing actors are tactics intended to determine which users or systems are entitled to what kind of access to a system.

这导致了许多用于实现安全的策略。识别、认证和授权的演员，目的是确定哪些用户或系统策略有什么样的访问系统。

No security tactic is foolproof and systems will be compromised. Hence, tactics exist to detect an attack, limit the spread of any attack, and to react and recover from an attack.

没有安全策略是万无一失的系统会受到影响。因此，战术的存在，以检测攻击，限制任何攻击的蔓延，并作出反应和恢复攻击。

Chapter10

Ensuring that a system is easily testable has payoffs both in terms of the cost of testing and the reliability of the system.

确保系统容易测试中存在的测试和系统可靠性的成本收益。

Controlling and observing the system state are a major class of testability tactics.

控制和观察系统状态是一种主要的可测性策略。

Complex systems are difficult to test because of the large state space in which their computations take place, and because of the larger number of interconnections among the elements of the system. Consequently, keeping the system simple is another class of tactics that supports testability.

复杂的系统是很难测试，因为它们的计算发生的大的状态空间，并且由于系统中的元素之间的较大数量的互连。因此，保持系统的简单是另一种策略，支持可测性。

Chapter11

Architectural support for usability involves both allowing the user to take the initiative in circumstances such as cancelling a long running command, undoing a completed command, and aggregating data and commands.

可用性架构支持，包括允许用户主动的情况下，如取消长时间运行的命令，最终完成的命令，并汇总数据和命令。

To predict user or system response, the system must keep a model of the user, the system, and the task.

要预测用户或系统的响应，系统必须保持用户、系统和任务的一个模型。

Chapter12

There are many other quality attributes than the seven that we cover in detail.

有许多其他的质量属性比我们详细介绍的七。

Taxonomies of attributes may offer some help, but their disadvantages often outweigh their advantages.

分类的属性可能会提供一些帮助，但他们的缺点往往大于优点。

You may need to design or analyze a system for a “new” quality attribute. While this may be challenging, it is doable.

您可能需要设计或分析一个“新的”质量属性的系统。虽然这可能是具有挑战性的，它是可行的。

Chapter13

An architectural pattern

架构模式

is a package of design decisions that is found repeatedly in practice,

是一个包的设计决策，在实践中反复发现，

has known properties that permit reuse, and

已知道的属性，允许重用，和

describes a class of architectures.

描述一类体系结构。

Tactics are simpler than patterns

策略比模式更简单

Patterns are underspecified with respect to real systems so they have to be augmented with tactics.

模式就少于实际系统，所以他们必须与策略。

Augmentation ends when requirements for a specific system are satisfied.

当特定系统的要求满足时，增强结束。

Chapter14

Analysis is always a cost/benefit activity

分析总是一个成本/收益的活动

Cost is measure of creating and executing the analysis models and tools

创建和执行分析模型和工具的成本是衡量

Benefit depends on

效益取决于

Accuracy of analysis

分析精度

Importance of what is being analyzed

什么是被分析的重要性

Analysis can be done through

分析可以通过

Models for some attributes

某些属性模型

Measurement

测量

Thought experiments

思想实验

Simulations

模拟

Prototypes

原型

Chapter15

The Agile Manifesto and principles value close-knit teams, with continuous, frequent delivery of working software.

敏捷宣言和原则价值紧密的团队，连续的，频繁的交付工作软件。

Agile processes were initially employed on small- to medium-sized projects with short time frames. They were seldom used for larger projects, particularly with distributed development.

最初采用敏捷过程的小型项目，短时间框架。它们很少用于更大的项目，特别是与分布式开发。

Large-scale successful projects need a blend of agile and architecture.

大型成功的项目需要一个混合的灵活和架构。

Agile architects take a middle ground, proposing an initial architecture and running with that, until its technical debt becomes too great, at which point they need to refactor.

敏捷的建筑师采取中间立场，提出一个初步的体系结构和运行，直到它的技术债务变得太大，这时他们需要重构。

Boehm and Turner found that projects have a “sweet spot” where up-front architecture planning pays off.

Boehm和Turner发现项目有一个“甜蜜点”在前面的建筑规划的回报。

Chapter16

Architectures are driven by architecturally significant requirements.

架构的架构的重大需求驱动。

Architecturally significant requirements may be captured from requirements documents, by interviewing stakeholders, or by conducting a Quality Attribute Workshop.

重要的建筑要求可以从需求文档捕获，通过采访相关人士，或者通过质量属性车间。

Be mindful of the business goals of the organization.

注意组织的业务目标。

Business goals can be expressed in a common, structured form and represented as scenarios.

商业目标可以以一种常见的、结构化的形式表示，并表示为情景。

Business goals may be elicited and documented using a structured facilitation method called PALM.

业务目标可能引起和记录使用结构化的便利方法称为手掌。

A useful representation of quality attribute requirements is in a utility tree.

质量属性需求的一个有用的表示形式是在实用程序树中。

The utility tree helps to capture these requirements in a structured form.

本实用程序树有助于捕捉这些要求，在一个结构化的形式。

Scenarios are prioritized.

方案优先。

Chapter17

Designing the architecture is a matter of

设计架构是一个问题

Determining the ASRs

确定地

Performing generate and test one an element to decompose it to satisfy the ASRs

执行生成和测试一个元素的分解，满足地

Iterating until requirements are satisfied.

迭代直到满足要求。

Chapter18

Understand the uses and the audience.

了解使用和观众。

Architectural documentation serves as a means for communication among various stakeholders.

架构文档是各种利益相关者之间沟通的一种手段。

An architecture is a complicated artifact, best expressed by focusing on views.

一个架构是一个复杂的工件，最好的表达，侧重于意见。

You must choose the views to document, must choose the notation to document these views, and must choose a set of views that is both minimal and adequate.

您必须选择文档的视图，必须选择符号来记录这些视图，并且必须选择一组最小且足够的视图。

You must document not only the structure of the architecture but also the behavior.

你不仅必须记录结构的结构，而且还必须记录行为。

Chapter19

Implementation

实施

Implementation activities can embed architecture knowledge in the code

实现活动可以在代码中嵌入架构知识

Templates can be used for critical sections that reoccur

可以用模板的关键部分，再次发生

Architecture erosion can be prevented through use of tools and management processes

通过使用工具和管理流程，可以防止架构侵蚀

Testing

测试

Unit and integration tests depend on architectural knowledge and a test harness.

单元和集成测试依赖于建筑知识和测试工具。

Network effects are difficult to discover when deploying a system to 1000s of servers.

网络的影响是很难发现在部署系统的服务器上。

The architect should be involved in a wide variety of test activities.

架构师应该参与各种各样的测试活动。

Chapter20

Architecture reconstruction and architecture conformance are crucial tools in the architect’s toolbox to ensure that a system is built the way it was designed, and that it evolves in a way that is consistent with its creators’ intentions.

架构的重建和架构的一致性是关键的工具，在架构师的工具箱，以确保一个系统的设计，它的发展方式，是符合其创作者的意图。

The results of architectural reconstruction can be used in several ways:

架构改造的结果可以用在几个方面：

If no documentation exists or if it is seriously out of date, the recovered architectural representation can be used as a basis for documenting.

如果没有文档存在，或者如果它是认真的日期，回收的架构表示可以用作记录的基础。

It can be used to recover the as-built architecture, or to check conformance against an “as-designed” architecture.

它可以用来恢复所建的架构，或检查是否符合“设计”架构。

The reconstruction can be used as the basis for analyzing the architecture or as a starting point for reengineering the system.

该重构可以作为分析体系结构的基础，也可以作为再工程系统的起点。

The representation can be used to identify elements for reuse or to establish an architecture-based software product line.

表示可用于识别重用的元素，或建立基于体系结构的软件产品线。

The software architecture reconstruction process comprises the following phases:

软件体系结构重构过程包括以下几个阶段：

Raw view extraction

原始视图提取

Database construction

数据库建设

View fusion

视图融合

Architecture analysis

建筑分析

Chapter21

If a system is important enough for you to explicitly design its architecture, then that architecture should be evaluated.

如果一个系统是很重要的，你可以显式地设计它的架构，那么该架构应该评估。

The number of evaluations and the extent of each evaluation may vary from project to project.

评估的数量和每一个评价的程度可能会有所不同从项目到项目。

A designer should perform an evaluation during the process of making an important decision.

一个设计师应该在作出重要决定的过程中进行评估。

Lightweight evaluations can be performed several times during a project as a peer review exercise.

轻量级的评价可以在一个项目中多次进行，作为同行评审练习。

The ATAM is a comprehensive method for evaluating software architectures.

ATAM是评价软件体系结构的综合方法。

Lightweight Architecture Evaluation, based on the ATAM, provides an inexpensive, low-ceremony architecture evaluation that can be carried out in an afternoon.

轻量级架构的评价，基于ATAM，提供了一种廉价、低仪式架构评估，可以在一个下午进行。

Chapter26

The cloud provides a new platform for applications with some different characteristics.

云为应用程序提供了一个不同的特性的新平台。

Architect needs to know how a cloud cluster works and pay special attention to

建筑师需要知道如何一个云集群的作品，并特别注意

Security

安全

Performance

性能

Availability

可用性