**Chapter 20. Architecture Reconstruction and Conformance**

**【无掌握点】**

了解：架构重构的背景和目的。

**Reconstruction背景:**

1. 系统已经存在，但你不知道它的架构。

从来没有记录。

有记录但文件已经丢失。

记录，但经过一系列更改后，文档不再与系统同步。

1. 如何维护这样一个系统?
2. 如何管理系统的演进，来维持架构所提供的QA?

**Reconstruction purpose：**

1. document一个documentation从未存在，或是已无可救药的架构。
2. 确保as-built architecture和as-designed architecture之间的一致性。
3. 在体系结构重构中，as-built architecture是从现有的系统构件逆向工程的。

**Reconstruction mapping:**

1. 当系统最初被开发时，它的架构元素被mapping到特定的实现元素:函数、类、文件、对象等等。
2. 当我们重构这些架构元素时，我们需要应用原映射的逆:

使用自动化和半自动化的提取工具

探究architect最初的设计意图。

通常我们使用两种技术的组合

**Tools for Reconstruction:**

架构重构是一项工具密集型活动:

工具提取关于系统的信息，通常通过搜索source code。

它们还可以分析其他工件，例如构建脚本或来自运行系统的跟踪。

我们需要帮助构建和聚合我们需要的体系结构抽象的工具。

架构重构工具不是万能的。

可能无法生成有用的体系结构表示。

并非架构的所有方面都易于自动提取。

架构重构是一个解释性、交互性和迭代性的过程，涉及到许多活动;它不是自动的。

它需要reverse-engineering expert的技能和注意力，最好是architect的。

**Reconstruction Workbenches:**

Reconstruction Workbenches是开放的，并提供一个集成框架，在这个框架中，添加到工具集的新工具不会不必要地影响现有的工具或数据。

**Guidelines：**

1. 在进行架构重构项目之前，要有一个目标和一组目标或问题。
2. 在开始详细的重构过程之前，获取系统的一些representation，无论多么粗糙。这种表示有几个目的，包括: 它标识需要从系统中提取什么信息。；它指导reconstructor决定在architecture中寻找什么以及生成什么视图。
3. Identifying layers是一个很好的起点。
4. 系统的现有文档在实现时可能不能准确地反映系统。
5. 工具可以支持reconstruction effort，缩短重构过程，但不能自动完成整个重构工作。：这项工作需要熟悉这个系统的人参与。

**Summary：**

1. 架构重构和架构一致性是架构师工具箱中的关键工具，它们可以确保系统按照设计的方式构建，并以与其创建者意图一致的方式演进。
2. architectural reconstruction的结果可以通过以下几种方式使用:

* 如果不存在文档，或者文档已经严重过时，那么可以将恢复的体系结构表示作为文档编制的基础。
* 它可以用于恢复已构建的体系结构，或者检查与“已设计”体系结构的一致性。
* reconstruction可以作为体系结构分析的基础，也可以作为系统reengineering的起点。
* 该表示可以用于标识reuse的元素，或者建立architecture-based的软件产品线。

1. 软件reconstruction过程包括以下几个阶段:

* Raw view extraction
* Database construction
* View fusion
* Architecture analysis

理解：架构重构的阶段。每个阶段的方法。

**Reconstruction Phases:**

1. Raw view extraction：

* 有关体系结构的原始信息可以从不同来源获得，主要是源代码、执行跟踪和构建脚本。
* 每一组原始信息都被称为视图。

1. Database construction.：

* 将提取的原始信息转换为标准格式
* 用它去填充reconstruction database。
* 我们将使用数据库生成权威的体系结构文档。

1. View fusion and manipulation.：

* 视图融合可以提高整体精度。
* 无法推断runtime behavior的静态视图。

1. Architecture analysis：

* 视图融合将产生一组关于架构的假设，需要测试这些假设是否正确。
* 如果没有，重复前面的步骤。

**extrcation.tiff**

**Reconstruction Phases method：**

1. Raw view extraction：

* 原始视图提取包括: 分析系统existing design和implementation artifacts，以构建其模型。
* 这些view将在稍后加以改进，以支持重建目标.
* 理想与现实的结合
* 从source artifacts (代码、头文件、构建文件等等)和other artifacts (例如，执行跟踪)中识别和捕获感兴趣的元素。
* Common Tools foe view extraction:

|  |  |
| --- | --- |
| **Tool** | **Description** |
| Parsers | Parsers analyze the code and generate internal representations from it |
| AST analyzers | Build AST tree for more further analysis |
| Lexical analyzers | Produce tokens |
| Profilers | Output information about the code as it is being executed |
| Code instrumentation tools | Intrusive tools to monitor systems |

1. Database construction.：

* 使用数据库存储提取的信息: 存储的信息量很大；数据的操作是繁琐的，如果手工操作很容易出错。
* 一些反向工程工具封装了数据库，因此该工具的用户不需要关心它的操作。

1. View fusion.：

* 提取的视图用来创建融合视图：融合视图来自一个或多个提取视图的信息组合在一起，每个视图可能包含专业信息。
* 创建一个融合视图就是创建一个架构的假设来帮助分析。：这些假设导致了新的聚合，这些聚合显示了元素的各种抽象或聚类。；通过解释这些融合视图，可以生成系统的hypothesized architectural views。；这些视图可以被解释、进一步细化或拒绝。；这个过程没有通用的完成标准。
* Example of fusion view：

显示了SonarJ工具的早期结果。SonarJ首先从一组源代码文件中提取事实，并允许您在系统中定义一组层和通过这些层的垂直切片。SonarJ将实例化用户指定的层和片的定义，并用提取的软件元素填充它们。

table.tiff

1. Architecture analysis——Finding Violations：

* 视图融合为我们提供了一组关于架构的假设。：这些假设以architectural elements和它们之间的关系的形式存在。
* 需要检验一下这些假设是否正确。
* 下图显示了向前一个图中最初创建的体系结构，添加关系和约束的结果。这些关系和约束是架构师添加的信息，以反映设计意图。

20_4.tiff

（数据层(第2行)可以访问，因此依赖于DSI层。数据层可能无法访问，并且对域、服务或控制器(第1、3和5行)没有依赖关系。）

* 在这个例子中，架构师指出了layers之间的关系。
* 使用这些关系和约束，SonarJ之类的工具能够自动检测并报告软件中分层的违规行为。
* Finding violation：

20-5.tiff

（SonarJ has found a violation: the **arc** between the **Service layer** and **JUnit**. ）