**阅读笔记**

**张畅-201800130005**

阅读章节：

《Software Architecture in Practice》--CHARPATER15：architecture in agile Projects

阅读内容目录：

15.1 How Much Architecture?

15.2 Agility and Architecture Methods

15.3 A Brief Example of Agile Architecting

15.4 Guidelines for the Agile Architect

15.5 Summary

15.6 For Further Reading

15.7 Discussion Questions

阅读此段原因：

由于在团队中主要负责敏捷开发的策略制定、时间规划和团队规划管理工作，需要输出敏捷开发相关的文档和安排图标，因此学习敏捷开发中的软件架构能够比较好的提升自己负责的工作质量，同时加深对敏捷开发的理解。

15.1 How Much Architecture?

重点一：敏捷开发的宣言

1．我们的首要任务是通过尽早并持续交付有价值的软件来满足客户的需求。

2．即使在开发后期，也欢迎不断变化的需求。敏捷流程利用变更来获得客户的 竞争优势。

3．频繁交付工作软件，从几周到几个月不等，而更倾向于较短的时间范围。

4．在整个项目中，业务人员和开发人员必须每天一起工作。

5．围绕有上进心的个人建立项目。给他们提供所需的环境和支持，并信任 他们来完成工作。

6．向开发团队内部传达信息的最有效方法是面对面的交谈。

7．工作软件是进度的主要衡量标准。

8．敏捷过程促进可持续发展。赞助者，开发者和用户应该能够无限期地保持恒定的步伐。9。持续关注技术卓越性和良好的设计可增强敏捷性。

10．简洁性(最大化未完成工作量的艺术)是必不可少的。

11．最好的体系结构，需求和设计来自自组织团队。

12．团队会定期思考如何提高效率，然后相应地调整和调整其行为。

讨论：敏捷开发还是BDUF(大前端设计)的设计流程

在不同的工程项目中，应该采用截然不同的规划、策划、设计、实践策略。

* 系统性极强、对质量和系统的统一性、决策的安全性具有较高要求的工程项目，更加适合使用BDUF(大前端设计)的设计流程
* 质量要求平常、需要控制成本、需要快速启动的工程项目，更加适合于使用使用敏捷开发的流程

作者使用一个造摩天大楼和家里小院的例子形象地描述出了两者在选择上的差距点。

作者通过项目工程情况和返工率，得到了如下的结论：

成功的项目显然需要两种方法的成功融合。过多的前期计划和承诺会扼杀创造力和适应不断变化的需求的能力，敏捷度过高可能会造成混乱。没有人愿意在拥有不经过严格的计划和彻底的分析的飞行控制软件的飞机上飞行。同样，没人会花18个月的时间 为他们的最新手机型号，视频游戏或口红计划电子商务网站。

我们所有人都想要的就是最佳位置‐乔治・费尔班克斯(G eorge Fairbanks)所说的“足够的建筑”。这不仅是完成正确数量的架构工作的问题，而且还在于正确的时间完成它的问题。敏捷项目倾向于希望根据需要实时地发展体系结构，而大型软件项目传统上则倾向于进行大量的前期分析和规划。

15.2 Agility and Architecture Methods

作者提出：用于体系结构文档的方法称为Views and Beyond

Views and Beyond与Agile在以下方面着重达成共识:如果不需要信息，请不要花费资源来对其进行文档化。所有文档都应有预定用途和目标读者，并应以能够为两者提供服务的方式进行制作。

技术文档的基本原则之一是“为读者写信”。这意味着了解谁会阅读文档以及他们将如何使用它。**如果没有听众，则无需制作文档。**该原理在敏捷方法 中是如此重要，以至于它被赋予了自己的名字:YAGNI。YAGNI的意思是“您将不需要它”，它指的是只有在确实有需要时才实施或记录某些东西的想法。不要花时间尝试预期所有可能的需求。

15.3 A Brief Example of Agile Architecting

作者以一个例子：WEB会议系统对上述的原理进行了阐释

Web会议系统是复杂而苛刻的系统。它们必须提供实时响应能力，竞争功能 ，易于安装和使用，轻巧的占地面积等。例如:

■ 他们必须在各种硬件和软件平台上工作，其细节不受架构师的控制。

■ 它们必须可靠并且提供低延迟的响应时间，尤其是对于实时功能，例如IP 语音(VoIP)和屏幕共享。

■ 它们必须提供高安全性，但是必须在未知的网络拓扑以及一组未知的防火墙和防火墙策略中提供高安全性。

■ 必须轻松修改它们，并轻松将它们集成到各种环境和应用程序中。

■ 它们必须具有很高的可用性，并且必须由具有广泛IT技能的用户轻松安装和学习。

存在的问题：

1. 功能的矛盾和兼容性：

许多目标相互取舍。通常，安全性(以加密形式)是以实时性能(延迟)为代价的。可修改性是以缩短上市时间为代价的。可用性和性能通常以可修改性和成本为代价。

即使有可能收集，分析并确定所有相关数据，功能需求和质量属性需求的优先级，但在竞争激烈的市场(例如网络会议)中普遍存在的严格的上市时间限制将使我们无法做到这一点。试图支持所有可能的用途非常棘手，并且用户本身也不足以预见该系统的所有可能用途。因此，仅以传统的需求启发方式询问用户他们想要什么，是不可能的。

1. 对客户需求快速响应与产品的长期生存盈利

经典的“敏捷与承诺”问题：一方面，架构师希望快速提供新功能，并快速响应客户需求。另一方面，系统和公司的长期生存意味着必须针对扩展性，可修改性和可移植性进行设计。这可以通过基于少量常规应用的模式和策略为体系结构提供简单的概念模型来最好地实现。尚不清楚我们如何将自己的方式“演化”为这种架构。

一种比较好的架构方案，同时也是针对这种矛盾的解决方案：

使用两种方式进行问题的思考：

■ 自上而下‐设计和分析建筑结构，以满足苛刻的质量属性要求和权衡

■ 自下而上—分析各种特定于实现的约束和特定于环境的约束，并为 它们制定解决方案

对于给出的问题，作者提供了一种思考流程和规划思路：

1. 首先，WebArrow团队快速创建并粗略地分析了初始软件和系统体系结构概念，然后从可以向客户展示的最关键功能开始，逐步实现并充实了它。
2. 每当出现新的要求或对问题域有了更好的了解时，他们都会调整体系结构 并重构设计和代码。
3. 随着产品的发展，连续实验，经验评估和架构分析可帮助确定架构决策。

这里可以获得的经验是：

使体系结构流程敏捷并不需要彻底地重新发明敏捷实践或体 系结构方法。WebArrow团队对实验的重视了关键因素。这是我们实现敏捷的架构概念、实施和评估形式的方式。

15.4 Guidelines for the Agile Architect

Barry Boehm及其同事开发了增量承诺模型，这是一个混合过程模型框架，旨在 在敏捷性和承诺之间找到平衡。该模型基于以下六个原理:

1. 对成功至关重要的利益相关者的承诺和责任
2. 利益相关者基于成功的谈判和权衡而“满意”(达到可接受的阈值)
3. 系统定义和利益相关者承诺的增量和演进增长
4. 迭代系统的开发与定义
5. 交错的系统定义和开发，可以尽早部署核心功能，不断进行更改以适应变化，以及及时增长复杂系统，而不必等待定义每个需求和子系统的情况
6. 风险管理‐风险驱动的锚点里程碑，这是同步和稳定所有这些并发活动的关键

Booch还指出：有效的敏捷过程将使体系结构随着系统的发展和成熟而逐步增长。成功的关键是要具有可分解性，关注点分离以及各部分几乎独立。

作者针对敏捷开发的系统架构给出的建议：

1. 如果您要构建具有相对稳定且易于理解的要求的大型复杂系统，则最好预先进行大量架构工作(“大”的一些示例值请参见图15.1)。
2. 在需求含糊不清或不稳定的大型项目中，即使它只是“ PowerPoint架构” ，即使它遗漏了许多细节，即使您只花了几天时间，也要快速设计一个完整 的候选架构。根据情况，在执行峰值和实验时以及在功能和质量属性要求出现并巩固的情况下，更改和详细说明此体系结构 。这种早期的体系结构将有助于指导开发，帮助早期的问题理解和分析，帮助确定需求，帮助团队进行协调，
3. 在需求不确定的小型项目中，至少要设法就要采用的中心模式达成共识，但不要在前期的建设，文档编制或分析上花费过多的时间。