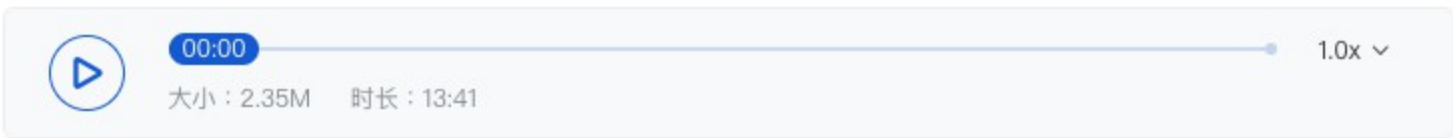




你的架构决策记录是否失去了它的目的？

作者 | Pierre Pureur, Kurt Bittner | 译者：张凯峰 | 策划：丁晓昀

2023-11-23 · 北京 · 本文字数：4910 字 · 阅读完需：约 16 分钟



架构决策记录（ADRs）是开发团队对系统所做的架构决策的重要沟通工具。如果缺乏对什么是架构的明确定义，同时也没有其他地方来记录重要决策，ADR 可能会远离其初衷，失去焦点和效果。

ADRs 旨在揭示架构决策，以此来提高透明度和责任性。但当它变得臃肿，并包含团队所做的每一个决策时，它就成为了对立面，因为架构决策淹没在其他被扔进 ADR 的内容中，难以被轻易发现。

为什么需要 ADR？

在[🔗之前的文章](#)中，我们观察到在动态软件开发方法中，解决方案会随着时间的推移而演变（例如敏捷开发），软件架构是由一系列关于系统如何处理质量属性需求的决策来定义的。这与以软件架构文档为主要定义的前期架构方法形成了对比。

ADR 使得架构决策变得透明，帮助开发团队澄清正在做什么以及为什么这样做，并为将来支持和增强系统的人保留这种推理过程。

当软件架构通过一系列决策的变化而演进时，这些决策是基于假设和测试这些假设的实验而产生的，开发团队需要一种跟踪他们所做的架构决策的方法。

随着时间的推移，其中一些决策可能会发生变化，开发团队需要一种方便查看这些决策的方式。即使它们没有变化，未来的团队也需要了解所考虑的选择和权衡，以便能够做出更好的系统演进决策。

ADR 的目的是什么？

对于这个问题，没有一个简单的答案。[🔗撰写这个主题](#)的作者们认为 ADR 应该记录*重要的*决策，并且有些人甚至进一步说应该是*架构上的重要决策*。这听起来是合理的，但是很难就*重要性*达成一致，而且更难决定什么是*架构上的*。

许多团队由于没有地方记录任何重要决策，将他们认为重要的任何决策都放入 ADR 中，这就淡化了架构方面，使 ADR 变成了一个“任意决策记录”。这样做会使 ADR 负载过重，有很多不应该存在的决策，而这些决策的存在只会让真正重要的架构决策更难以看到。所以，要决定将什么放入 ADR 中，我们必须决定*什么是架构的*？这并不像看起来那么容易。

"架构的"是什么意思？

最近，我们陷入了一个《公主新娘》的剧情里，伊尼戈·蒙托亚（由曼迪·派廷金饰演）对维兹尼（由华莱士·肖恩饰演）说：“你总是用那个词……我觉得你并不明白它的意思。”

我们在软件领域经常提到“架构”这个词，好像我们完全知道它的意义。但是当我们更仔细地审视这个概念时，我们很难准确地界定它的含义。在本文中，我们来尝试更明确地定义我们认为的架构和非架构，并为做出决策提供更明确的标准。

“架构”这个术语在软件开发中的应用实际上是问题的一部分；它是错误的隐喻。在物理世界中，架构主要关注可用性和美学。我们所谓的“软件架构”更类似于结构工程，主要关注物理系统如何弹性地承受负载。

推荐阅读

架构智能——下一代人工智能

🔗 AI&大模型

24. 如何维护对方的自尊心和自信心？

2023-10-17

自我革新，如何让软件架构变得更好

🔗 架构

1. 微服务架构的演变

2023-09-27

07. 从自己解决到集体决策的转变

2023-10-17

从迪卡依的架构流程看如何管理大规模软件架构

🔗 架构

InfoQ 2024 年趋势报告：架构篇

🔗 架构，云计算

电子书



中国开发者画像洞察研究报告 2024

分析开发者的行为模式、工作价值、职业发展等内容，帮助整个行业生态更深入地理解开发者，为他们提供更精准...

立即下载

大厂实战PPT下载

换一换



AGI 与 AIGC 浪潮下的我们

杨攀 | 极客邦科技 副总裁、TGO 鲲鹏会总经理

立即下载

降本九成，提效十倍：统一资源池理念重塑CLS规模红利

林兆祥 | 腾讯云 专家工程师

立即下载

FinOps：从概念到落地

林昊（毕玄） | 贝联珠贵 创始人 & CEO

立即下载

同样，软件架构的艺术是预测软件系统的负载并为其设计。一个关键区别是结构工程是基于几千年经验的广泛知识体系，并通过科学推导出的物理定律和数学模型来加强。软件与此完全不同。它是编码的思想，并且除了某些类型的算法之外，解决问题的标准方法很少。

一个理解软件架构的良好起点是 Grady Booch 的观察，它对比了架构和设计：

"所有的架构都是设计，但并非所有设计都是架构。架构代表了塑造系统形式和功能的一系列重要设计决策，而重要性是通过变更成本来衡量的。" (Grady Booch [🔗 on Twitter](#))。

这个观察的重要部分是：

1. 架构决策具有高昂的撤销成本；
2. 架构决策定义了解决方案的基本特征或“形状”，我们将其解释为解决由系统的质量属性需求集合所定义的问题的基本方法 - 详见《[🔗 软件架构实践](#)》第 2 章进行更深入的讨论。

如果一个决策不涉及这两个方面的任何一个，我们认为它不应该包含在 ADR 中。让我们更详细地审视这个断言。

什么样的变更是昂贵的？

有些决策是昂贵的变更，但并不一定复杂，复杂指的是“在智力上具有挑战性”或者“如果做出错误决策可能会造成严重后果”。换句话说，重写代码并不复杂，真正复杂的是重新思考代码背后的概念。

举个例子，有些决策是昂贵的变更，但并不复杂，比如：

- **重新设计应用程序的用户界面。**即使使用了 UI 框架，改变视觉隐喻可能是耗时且昂贵的，但只要这些改变不影响系统处理的基本概念，就不会产生复杂性。
- **用具有相同功能的另一个主要组件或子系统替换原有的组件或子系统。**一个例子是从一个供应商的 SQL 数据库切换到另一个供应商的 SQL 数据库。这些变化可能需要一些工作，但转换工具会有所帮助，此外，避免使用专有功能也是有帮助的。只要新的组件/子系统支持与旧的组件/子系统相同的基本概念，这个变更不会改变系统的架构。
- **即使更换编程语言在架构上可能并不重要，只要这些语言支持相同的抽象和编程语言概念。**换句话说，语法变化并不重要，但是对基本概念或隐喻的改变是重要的。

使用适当的转换工具，这些决策实际上可能并不会很昂贵。以前重写用户界面是昂贵的，但现代 UI 设计工具和框架使这种工作相对廉价。决定使用哪个 UI 框架、SQL 数据库或编程语言是一个实现细节，而不是架构决策。这些都是重要的决策，但它们并未达到架构决策的层面。

甚至"架构上重要"的成本标准归结为"解决方案的形状"。

"解决方案的形状" 是什么意思？

对我们来说，“解决方案的形状”指的是系统用于解决问题的基本数据结构和算法。以上面关于 SQL 数据库的观察为例，选择特定的 SQL 数据库可能在架构上并不重要，但是从使用行和列来表示基本概念转变为使用树结构或非结构化数据是重要的。搜索、排序和更新这些不同类型表示的算法非常不同，具有不同的优势和劣势，因此你的选择将显著影响系统满足 QARs 的能力。

更一般地说，对我们来说，架构决策具有以下特点：

- 它们涉及系统使用的基本概念以及数据结构中表示的关键抽象（例如类、类型等），这些数据结构用于在整个系统甚至系统之间共享信息。
- 它们还涉及使用这些数据结构的方式，即访问和操作数据结构的基本算法。
- 对用于表示系统基本概念的数据结构的任何更改都会影响使用这些数据结构的算法，而对算法的任何更改都会改变它们所使用的数据结构。

架构首先为系统能够解决的问题类型设定了限制，有时甚至会对开发人员看待不同解决方案的能力造成一种类似锤子-钉子的盲目性，使他们无法看到其他可能的选择。改变架构决策意味着改变系统所处理的基本概念以及处理这些概念的方式。

除了表示关键概念的算法和数据结构之外，其他选择也在塑造架构中起着关键作用，例如：

- 对消息传递范例的更改 - 例如，从同步到异步
- 对响应时间承诺的更改 - 例如，从非实时到实时
- 对并发/一致性策略的更改 - 例如，乐观与悲观资源锁定
- 对事务控制算法的更改 - 例如，失败/重试策略
- 影响延迟的数据分布的更改
- 对缓存一致性策略的更改，特别是对联合数据的更改
- 对安全模型的更改，特别是在扩展到单个对象或元素时的安全访问粒度。

最终，所有这些选择都会转化为不容易更改的代码，因为这些选择的代码影响分散在整个软件中，而不是局部的。如果某个东西可以局部化和封装，通常不是属于架构范畴，因为可以在不对代码产生连锁影响的情况下进行更改。

架构和决策的持久性

有时候，决策的预期寿命会使团队认为该决策是架构性的。大多数决策都会变成长期决策，因为大多数系统的资金模型只考虑了开发的初始成本，而不考虑系统的长期演进。在这种情况下，每个决策都变成了长期决策。然而，这并不意味着这些决策就是架构性的；它们需要具有高成本和复杂性，才能在撤销/重做方面具有架构上的重要性。

举个例子，选择数据库管理系统的决策通常被认为是架构性的，因为许多系统将在其整个生命周期中使用它，但如果这个决策可以轻松地被撤销而无需修改整个系统的代码，那么它通常并不具备架构上的重要性。现代关系型数据库管理系统技术非常稳定，不同厂商的产品相对易于互换，因此只要与数据库的接口进行了隔离，就可以比较容易地将商业产品替换为开源产品，反之亦然。*架构上的决策*是局部化数据库依赖和抽象特定厂商接口，而不是选择数据库本身。

决策的持久性在 ADR 中起作用的地方是解决可持续性和弹性的问题。可持续性涉及系统能够应对未知概率和影响的未来事件集。弹性是指系统在发生这些事件时能够抵抗失败的能力。当人们说某个系统具有可持续性时，他们的意思是相信该系统能够处理他们能够想到的一切，甚至是无法预料的事情。

何时需要做架构决策？

在过去，一个团队会在系统开发的早期创建一个软件架构文档，该文档将指导系统在整个生命周期中的开发。

当一个团队使用敏捷方法来开发系统时，决策记录（ADRs）将集体取代软件架构文档，因为它们逐步记录架构以支持系统的逐步开发。我们在之前的文章中已经描述了最小可行架构（MVA）如何与最小可行产品（MVP）增量并行演化。实际上，这意味着团队将随着解决方案的演进而逐渐做出架构决策。与软件架构文档不同，ADRs 记录的决策是一次性地预先制定的。

在所有重要决策中使用 ADRs 有什么坏处？

简而言之，它会使事情变得混乱，使真正的架构问题更难以看清。这样做会使关于基本决策的讨论变得更加困难，因为问题不明确，尤其是如果决策的影响没有完全说明。

出于多种原因，团队仍然需要记录非架构性的决策，其中许多归结为需要记录决策及其原因，以便以后有人需要解释或证明。有时，为了将决策记录在 ADR 中，决策被分类为“架构性决策”，因为没有更好的地方可以记录它。

ADR 不应该用于以下用途：

- **促进重用**。有些人，尤其是管理者，将 ADR 视为强制实施重用的手段。他们希望看到常见的组件和子系统被重复使用，因为他们认为这样可以降低成本和简化开发。然而，只有当重复使用的组件和子系统适合并且能够带来更好的解决方案时，这种观点才是正确的。当它们不适用时，会使设计变得复杂，并且会使架构变得更糟。我们大家可能都有过这样的经历，为了使“公司标准”落实在当前问题上但并不是最佳解决方案而苦苦挣扎。结果通常是成本增加和弹性降低。促进重用是开发组织中知识共享的一个方面，但是有比在 ADR 中添加大量关于设计和代码可能重用的信息更好的方式，来促进这种知识共享。我们认为更好的方式是使 ADR 成为团队试图解决问题以及选择其方法的原因的清晰记录。
- **推卸责任（自我保护）**。有些团队认为，通过将决策放在 ADR 中，他们可以免除该决策带来的后果。越多的人看到并明确或默认地批准 ADR，对于一个糟糕决策的责任就被稀释了。他们认为，人越多就越安全。惩罚糟糕决策的人是有毒的管理文化的表现。开发团队根据当时可获得的信息做出最佳决策。当他们获得更多信息时，通常是通过构建和部署系统，其中一些决策将会改变。减少决策变更成本的关键是通过小步增量构建系统并频繁测试假设。批评过去的决策会让人感到沮丧且无效。如果团队不必担心因决策被指责而受到责备，他们就可以专注于通过实验构建更好的解决方案，而不是使用 ADR 来使自己免受责备。
- **记录非架构产品决策**。这经常发生是因为团队经常做出重要决策，但如果没有记录它们的地方，他们就会将它们放在 ADR 中。以这种方式滥用 ADR 会使架构更难理解：如果每个决策都是架构决策，那么没有一个决策是架构决策。换句话说，一个变成“任意决策记录”的 ADR 已经失去了它的目的。对此有一个简单的解决办法：只需保留与架构无关的重要决策的日志。架构决策通常只能被开发人员理解，而大多数其他重要决策的记录拥有更广泛的受众。将它们分开通常会让每个人都更加满意。

结论

尽管所有的架构决策都很重要，但并不是每个重要的决策都是架构决策。创建架构决策记录和其他重要决策的分离有助于提高组织间的沟通。ADR 中包含了通常不会引起广泛兴趣的技术讨论，将它们分开使系统的架构更易于理解。

如果 ADR 只关注架构，它们可以让人了解团队在权衡选择和取舍时的思考过程的演变。决策从来都不是错的，它们只是团队在某个时间点上思考的表现。是的，随着时间的推移，他们可能会选择不同的方法，但保留这种演变的记录是有用的。了解团队的思维如何演变可以提供对当前和未来权衡的见解。

在软件架构中，往往没有完美的解决方案，只有需要平衡的“不太完美”的替代方案。能够更清楚地看到这些选择有助于当前和未来的团队更好地理解他们可能需要做出的权衡。

原文链接：

<https://www.infoq.com/articles/architectural-decision-record-purpose/>

相关阅读：


[🔗 不要让框架影响你最初的架构设计](#)

[🔗 混合云的多活架构指南](#)

[🔗 深度解读：分布式系统韧性架构压舱石OpenChaos](#)

[🔗 架构师是怎样炼成的](#)

发布于：2023-11-23 08:00 | 阅读数：9988
文章版权归极客邦科技InfoQ所有，未经许可不得转载。

 业务架构

 架构

 轻点一下，留下你的鼓励

评论

快抢沙发！虚位以待

发布

• 暂无评论 •

更多内容推荐

06 | 领域拆分：如何合理地拆分系统？

在开发新需求的同时，我们要对系统定期做拆分整理，避免系统越跑越偏。

2022-11-04

不读代码也能做技术决策？试试可塑性开发吧

开发人员的大部分时间都花在阅读代码上。

 语言 & 开发, 最佳实践, DevOps & 平台工程, 软件工程, 性能优化, 编程语言, 框架

ThoughtWorks CTO：2025 年之前，我们会看到架构的演进，但不会看到革命

ThoughtWorks CTO Rebecca Parsons从演进式架构的定义开始，回顾了每项“能力”和属性，预测了在下一个阶...

 架构, ThoughtWorks, 可观测, 企业动态

25 | Android 系统解耦：殊途同归，Android 系统组件化之路

这节课，我们会学习整机组件化的架构设计，并探讨具体有哪些针对性的解耦策略。

2023-04-07

函数式编程的后期架构

与面向对象编程（OOP）相比，函数式编程能够支持后期架构并减少耦合。

 语言 & 开发, 文化 & 方法, 编程语言, 最佳实践, 性能优化, 领域驱动设计


架构师必须了解的 5 种最佳软件架构模式

在本文中，我们将看看什么是软件架构模式，并对其中一些模式进行详细介绍。请记住，可以在单个系统中使用许...

 架构, 最佳实践, 方法论, 编程语言, 框架, 在线混部, 银行, 医疗

软件架构决策指北：怀疑主义的软件架构设计

怀疑主义是一种架构超能力，可以帮助我们在错误的假设走得太远之前识破它们。

 架构, 方法论, 性能优化, 框架, 团队搭建, 保险, 汽车

30 | 限界上下文（下）：限界上下文之间如何集成？

在云原生的情况下，一般不会采用单体架构，微服务才是最佳实践。那么微服务应该怎么设计呢？

2023-02-21

将 MVP 和 MVA 应用于遗留应用程序

遗留应用的每个版本，都可以视为一个MVP。

 架构, 研发效能, 产品, Scrum, 方法论, 框架, 领域驱动设计, 团队搭建, 技术选型, 银行, 保险, 汽车, 企业动态...

最小可行架构注意事项：必须考虑分布式处理和数据的位置

在设计最小可行架构时，开发人员也必须考虑资源所处的位置，特别是当移动应用程序是分布式系统的一部分时。

 文化 & 方法, 架构, 数据处理, 性能优化, 框架, 微服务, 数据集成, 银行, 证券, 保险, 汽车, 企业动态, 行业深度

亚马逊 CTO 20 年架构经验之道：俭约架构师的七大黄金法则！

在管理云成本方面，是时候成为节俭的架构师了。

 框架, 服务革新

QCon SF 闭幕主题演讲：软件设计是一种人际关系活动

增量迭代开发是最具成本效益的软件构建方式。

 文化 & 方法, 研发效能, 最佳实践, 方法论, 架构

41 | 职业成长（下）：架构师成长的充分条件是什么？

如果你下定决心想要做一名优秀的架构师，就必须找到一个好环境。它是Pegasus，是你的天马。

2022-05-31

InfoQ 2023 年趋势报告：事件驱动架构、深度学习和人工智能、云原生架构和容器化技术

未来的软件架构需要更具可扩展性、弹性、安全性和高效性，以适应越来越复杂的业务需求。

 架构, AI&大模型, 云原生, 容器, 安全, 方法论, 编程语言, 框架, 微服务, 生成式 AI, 可观测, 企业动态

构建可持续架构的三大秘籍

本文介绍了一种包含三个构建块的架构决策框架。

 架构, 方法论, 编程语言, 框架, DevOps & 平台工程, 企业动态

不要让框架影响你最初的架构设计

在真正需要之前，不要对任何特定的框架、模式或策略过多投入。

 架构, 文化 & 方法, 方法论, 最佳实践, 性能优化, 编程语言, 框架, 亚马逊云科技, 银行, 保险, 企业动态, 行业深度

28 | 限界上下文（上）：怎样为更大的需求建模？

让我们暂时忘掉“限界上下文”这个本身，先回到我们的项目，一步一步地理解这个模式的概念和用法。

2023-02-16

写给架构师的技术债“偿还”指南

聊聊技术债务的定义，影响以及管理。

🔗 架构，文化 & 方法，研发效能，操作系统，框架

老架构师总结的 12 个软件架构陷阱 | 避坑指南

了解潜在的软件架构缺陷可以帮助团队避开很多陷阱。

🔗 业务架构

29 | 限界上下文（中）：限界上下文怎样影响架构设计？

今天我们继续完成“工时管理”上下文，帮你进一步深化这两个概念。然后，我们会根据限界上下文来完成架构设计。

2023-02-18

发现更多内容



促进软件开发及相关领域知识与创新的传播

关于我们
我要投稿
合作伙伴
加入我们
关注我们

联系我们

内容投稿：editors@geekbang.com
业务合作：hezuo@geekbang.com
反馈投诉：feedback@geekbang.com
加入我们：zhaopin@geekbang.com
联系电话：010-64738142
地址：北京市朝阳区望京北路9号2幢7层A701

InfoQ 近期会议

北京 · QCon 全球软件开发大会 2025.4.10-12
上海 · AICon 全球人工智能开发与应用大会 2025.5.23-24
北京 · AICon 全球人工智能开发与应用大会 2025.6.27-28

全球 InfoQ

🇨🇦 InfoQ En
🇯🇵 InfoQ Jp
🇫🇷 InfoQ Fr
🇧🇷 InfoQ Br