大型技术组织 DevOps 转型经验总结

> DevOps (a clipped compound of “development” and “operations”) is a software engineering practice that **aims at unifying software development (Dev) and software operation (Ops)**.
> DevOps 是一项软件工程实践，意在将软件的**开发和运维统一**起来。
> The main characteristic of the DevOps movement is to **strongly advocate automation and monitoring** at all steps of software construction, from integration, testing, releasing to deployment and infrastructure management.
> 其显著特征是**倡导在整个软件构建过程中实现自动化和监控**。
> DevOps aims at shorter development cycles, increased deployment frequency, more dependable releases, **in close alignment with business objectives**.
> DevOps 追求更短的开发周期、频繁增量地部署、更可靠的发布版本，**以达成业务目标**。

以上是 Wiki 关于 DevOps 的描述，但我们如何解读？

DevOps产生的基础条件

任何一项实践都有其诞生和适用的土壤。与敏捷的中心实践**持续集成**一样，DevOps 是云平台开发交付的最佳实践。换句话说，是因为云平台带来的两大变化：

1. 简化了IT基础设施运维工作，但是却增加了管理的服务器规模。
2. 各个环　一致性的增加（网络、操作系统版本、过去的生产环　License限制、准备相同环　的难度，现在都可以用 image/snapshot 解决了），使得开发人员可以使用少量脚本完成不同环　的部署，甚至控制部署资源的启动和回收。

一句话简述：就是太大了。千人规模以上的技术组织，在过去业务规模化发展时，没有像上面 Instagram 这个好例子那样，通过在几个层面的简化或治理来保障规模化之后的整体质量水平：

- 基础设施是否可扩展？
- 架构是否可扩展？
- 核心能力是否在组织中共享？
- 避免人工操作的误差？

当然，**欠缺清晰的业务价值主张**也是技术组织逐渐变得庞大的原因之一，比如 Instagram，它并没有在用户增长 7000 万之后成为一个庞然大物，依然是一个照片分享应用，这与其明确的价值主张有关，但这个话题不在本文讨论。

因此，在技术组织变大的过程中，通常面临的问题就是：

- 陈旧的技术，大量的债务
- 人员能力参差不齐，团队经常劳而无果
- 线上质量事故多，业务满意度低下

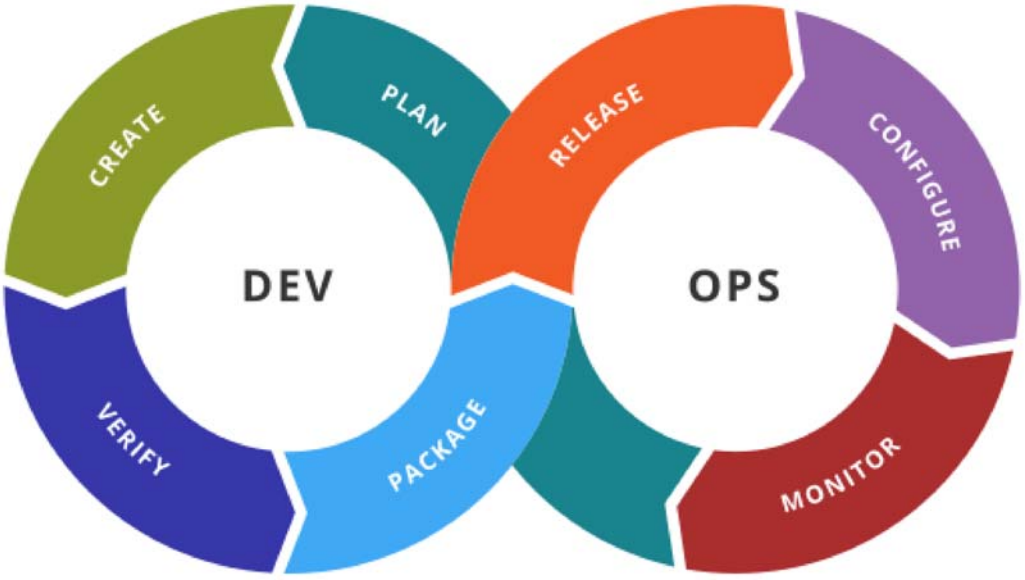
实践驱动不了组织

前文说了，DevOps 是一项软件工程实践，对当今而言，它是最为先进的软件工程实践之一，**意味着在低成本短周期持续发布可靠的业务版本的能力。**

组织不是一个项目，实践可以改变一个人对结果实现的过程认知，但组织是超越个体的联结，实践无法对组织产生任何改变。对组织而言，DevOps 的实施是一项综合性挑战。

我们来看看 Wiki 给出的 DevOps 覆盖范围描述：

1. **Code** — code development and review, source code management tools, code merging
2. **Build** — continuous integration tools, build status
3. **Test** — continuous testing tools that provide feedback on business risks
4. **Package** — artifact repository, application pre-deployment staging



它要求制定统一的构建部署标准和工具，达到一致的部署环境；对团队而言，它要求频繁集成代码的工作纪律，编写自动化测试的能力和尽早修复问题的决心。**以及，将一切步骤自动化。**

这意味着持续地挑战组织内现有的标准、流程、工具甚至安全策略（各个部署环境的打通）；也意味着持续地挑战团队的既有项目成果、工作方式甚至曾经引以为豪的经验履历。

在一场组织层面的变革中，如何更平滑地引入新的技术实践？

启动：利用六层变革阻力模型

约束理论（Theory of Constraints）为我们提供了利用变革阻力的六层模型。

Questions	Objectives	Layers of Resistance
What to change?	Situation assessment, description of "current reality," and identification of the core problem or conflict and assumptions that sustain it. Diagnosis, systemic root cause analysis.	1) Lack of agreement on the problem

第一层阻力：缺乏对问题根源的共识

许多组织在敏捷转型期间就在自动化测试上跌了大跟头。就在于对根本问题和根本原因缺乏分析和理解（**Systemic root cause analysis**）。为什么要做自动化测试？当前交付物质量，尤其是交付给测试人员的版本质量究竟如何（**Current reality**）？戴明的**红珠实验**表明，在上游环节交付物质量就不合格的情况下，**下游无论采取何种办法都无法提升最终产出物质量。**所以，在这个环节上实施自动化测试，是一个错误的变革起步点。

变成什么？

第二层阻力：缺乏对可行性方案的共识
第三层阻力：缺乏对方案解决问题信心的共识
第四层阻力：担心新解决方案又将产生新的问题

组织如何达成变革的目标，即是通过决策（Decision-making）形成的转型方案，它应该是根据组织的现状及根源问题，采取的一系列策略的组合。对每个组织而言，它应该是定制化、有针对性的。

让变革发生！

第五层阻力：缺乏排除阻塞实施方案的清晰路径

这是最常见也最棘手的阻力：好不容易在中上层达成了共识，通过了方案，去到团队一看，交付任务都堆积如山呢，如何配合你学习新知识新方法新工具？立即歇菜。通常这时候，需要一个身经百战的高手，迅速解决团队当前的问题将之“松绑”，就能够迅速建立起信心和声望，使团队进入新的轨道。

第六层阻力：缺乏跟进

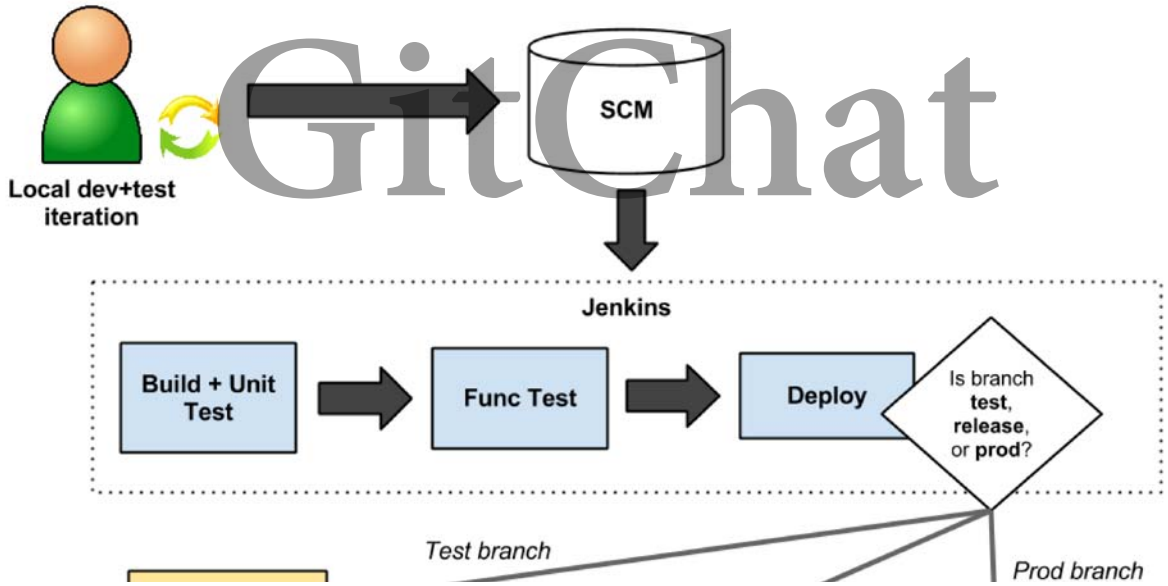
这也是最常见的终极阻力，也是客户常常报怨“教练”式顾问的原因：我们希望你们能够自带项目经理功能，不要什么事都让我们去做！

> **经验总结：**对落地软件工程实践的技术组织转型，规模化成功的关键在于，有平台级工具作为技术实践的支撑，以降低组织投入的复杂度。对 DevOps 转型而言，就是需要落地一整套 DevOps 工具平台及使用标准。

DevOps工具平台实施关键

一是基础的基础：版本管理。作为代码的来源，它支撑起了整个构建、部署、测试和发布的流程。版本基线管理已经是非常落后的办法了，相对于 `git-flow` ，我比较推崇的是 Netflix 的 `test release product` 三个恒定的分支分别支撑开发、发布、主干。`hotfix` 作为线上修复的临时分支。

- 开发人员平时工作在Test分支上，它的提交可以触发测试环境的自动部署。
- Release分支用于每周自动部署，只需要把代码提交到Release分支上，它会触发自动化测试及集成测试，◘后续过程会由某个特定成员或部署团队启动，所有的动作都是自动化的。部署完成之后，代码会自动Push到Prod分支上。
- 如果某个特性需要立即发布而不是通过每周固定发布流程；开发人员可以向Prod分支提交，这次提交会自动触发Release分支的合并并触发自动部署流程，如果审核人员通过，它将被立即部署。



能会变成另一个庞大繁杂的项目，这本身也违背了 DevOps 可以给组织带来**在低成本短周期持续发布可靠的业务版本的能力**的初衷。

最后的忠告

我曾经在超过 6000 人的IT组织里担任持续集成云的产品经理 + 架构师，拆分和重写了 Jenkins 最关键的任务调度、日志存储以水平扩展其构建性能。半年交付 beta 版本支撑 800 人开发，一年发布至稳定版本，支撑越过 4000 人每日数万次的构建发布。

随后作为咨询项目负责人帮助一家 2000 人规模的组织在两个月内建立起开源 DevOps 平台及团队，支撑试点团队的开发运维。

DevOps 实施的难点不在于技术，最大的阻力来自于组织过大之后，各项工具的管理、实施分散在各个部门，如果组织层面没有达成共识和调整集中决策，**即使后面市场上出现成熟的 DevOps 商业产品，也很难在组织内有效落地**。这点上，可以参考 20 年前的 ERP 变革对组织决策权的影响，自动化的趋势已不可阻挡。