DevOps =

学习笔记

【LEEB】@6/18/2013

目录


DevOps
一个故事
DevOps概述
DevOps解决那些问题？
如何实施DevOps?
DevOps与ITIL
总结

一个故事：部署软件产品

【开发部门】

开发部门要开发一款新产品。这款产品要使用最新最炫的技术，来保证客户的所有花俏的需求，从而给公司带来百万美元的利润。这款产品被要求使用最新的技术和运行平台，还得马上交付。于是开发部门没日没夜的加班、赶代码(cuts code like crazy），终于如期完成了任务。然后他们把自己的“杰作”一股脑的甩给了运维部门，后者还没能完全接手，前者已经迫不及待的开始了庆功会。

【运维部门】

接到产品后，运维部门每个人的心中都充满了恐惧。下面就是运维部门的恐惧之源：

① 这款优秀的产品在目前的底层平台上无法运行，因为这个平台{太古老了，空间不足，不支持某某版本}

② 这款产品的体系结构跟我们的{存储，网络，部署，安全}模型不匹配。

③ 这款产品的{ 报告，安全，监视，备份，服务提供} 我们搞不懂 ，所以没法把它做成实际可用的产品。

尽管伴随着不绝于耳的抱怨和咒骂，运维部门最终还是把这款产品安装好了。不幸的是，由于做了很多蹩脚的修改和不合理的强迫式运行，这款产品的性能最后被归结为：终极失败（Epic Fail）。

于是非常沮丧的运维部门开始记录各种问题，源源不断的给开发部门提Issue。而开发部门的回应基本上都是：

① 这不是我们的错 —我们的代码非常完美—而是（运维部门的）部署做的太差劲了。

② 运维部门比较笨，他们不懂新技术—为什么他们没法实现最新的技术呢？为什么他们这么落伍呢？

③ 在我的机器上运行的没问题啊……

两个部门之间的交流很快变成了一场暴风骤雨。客户（以及股东、投资方和管理层）则成了蒙受损失的失败方。最终公司损失了无数的金钱，大家也都失业了。终极的失败。

DevOps 就是想方设法的避免这种“终极失败”，同时让大家用更聪明更有效的方式去工作。**它是一种框架，包含了很多优秀想法和原则，它鼓励开发部门和运维部门通力合作**。在DevOps环境中，开发人员和系统管理员会构建一些关系、流程和工具，从而更好的与客户互动，最终提供更好的服务。

一个短片

短片

影片是2010年DevOpsDay的开场短，影片中，查理·卓别林的角色是运营人员，但一连串的事件让他觉悟到DevOps的能力，影片中影射的问题是：

1、开发人员经常不考虑自己写的代码会对运营造成什么影响。他们在交付代码之前，并不邀请运营人员参与架构决策或代码评审。

2、开发人员对配置或环境进行修改之后，经常没有及时与运营人员沟通，导致新的代码不能运行。

3、开发人员可能对运行时环境缺乏了解，从而难以正确地对代码进行调整。

4、运营人员可能对应用程序内部缺乏了解，从而难以正确地选择运行时环境和发布流程。

5、运营人员尝试避免变更，新功能流入生产环境的速度因此被延缓，从而延缓了开发人员将特性交付给用户使用的速度。

目录

DevOps

什么是DevOps

集合性概念

能够理顺开发和运维之间相互配合关系的任何事物。

DevOps是一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运营和质量保障（QA）部门之间的沟通、协作与整合。

它的出现是由于软件行业日益清晰地认识到：

为了按时交付软件产品和服务，开发和运营工作必须紧密合作。

在思想上，DevOps是敏捷理念向运维领域的延伸；

在流程上，它是“需求”到“上线”全线贯通的关键；

在工具及技术准备上，虚拟技术、云计算以及各类工具的日趋成熟，为这场变革奠定坚实的基础。

什么是DevOps

DevOps 也不仅仅是一种软件的部署方法。它通过一种全新的方式，来思考如何让软件的作者（开发部门）和运营者（运营部门）进行合作与协同。使用了DevOps模型之后，会使两个部门更好的交互，使两者的关系得到改善，从而让很多领域从中受益，例如：自动化、监视、能力规划和性能、备份与恢复、安全、网络以及服务提供（provisioning）等等。

“对于DevOps是什么？” 这个问题，DevOps社区中的每个人的回答都不尽相同。因为我们的工作经验不同，关注的问题也不同。就我个人而言，DevOps分成四大部分：

简单

KISS（Keep it Simple and Stupid，简单就是美）原则是最重要的。所以本段文字也很简单。我们要尽量提供简单、可重用的解决方案。“简单”节约了书写文档、培训和提供支持的时间。“简单”增加了沟通的速度、避免混淆、减少了开发和运维出错时的风险。“简单”让人更快的发布产品。

部门之间关系

早参与，多参与。对于开发人员，要让运维人员常驻到开发部门，全程参与开发流程。邀请运维人员参与你的Scrum或者开发会议，与他们分享项目计划、分享新技术的点子和心得。搜集功能性需求（指开发人员用到的需求）的同时也要搜集运维方面的需求。把对于“发布、备份、监控、安全、配置管理和系统功能”的测试作为一项独立的项目流程。软件产品在开发时解决的问题越多，那么在使用时暴露给用户的问题就越少。给运维人员做培训，让他们弄清楚项目的体系结构和核心代码。如果运维人员在反馈bug时提供的信息越多，那么你花在排查问题（trouble-shooting) 的时间就越少，这个bug也就会更快的被解决掉。

对于运维人员，在遇到问题时需要把开发人员加进来，大家一起解决问题。邀请开发人员参与你们的会议，分享项目进度(roadmaps)，并且共同修订工作计划。运维人员一定要了解开发部门下一步的工作方向，从而确保产品运行的底层平台能够良好的支持最新技术。开发人员也会带来相关的技术、知识和工作，帮助你们改善产品的运行环境，使其更加易于维护、简洁有效。

•

部门之间关系

一些开发领域的概念，例如：“要根据API而非封闭的interface来构建工具”，分布式版本控制，驱动测试开发，以及诸如敏捷开发、看板管理(Kanban) 和Scrum等方法论。如果把这些概念应用在运维领域，同样会产生革命性的变革。

不要惧怕新点子和新技术。我们可以随时随地的向他人学习，哪怕是一句“我们再也不要那样做了！” 也会让我们从中获益。尽管处于不同的部门，但是我们要共同学习、共同成长，这样才能协同工作的更好！

按照从高到低的顺序，有效的沟通方式应该是：

1. 面对面交流
2. 视频会议
3. 电话
4. 即时通讯软件
5. Email

工作中的流程

不要低估流程和自动化的作用。很多公司都有自己的流程管理(process engineering)—— 从原始的笔录到 ISO9001。但它们都存在一个关键的缺陷：过于理想化，它要求每个步骤都必须成功执行。例如：为了搭建一台新主机，会有下列一套简单的流程：步骤一：装机（把各个硬件组装到一起）。步骤二：接线、通电。步骤三：安装操作系统。接下来还有步骤四、五、六。如果一切顺利的话，第N步结束之后就会有一个功能完整、运行正常的新主机。但万一有个流程没跑通怎么办？比如说在某个步骤断了，走不下去了，或者在这一步得到了异常的输出，有没有另外的步骤来处理这个异常？

所以，流程绝对不会从头到尾一帆风顺，所以我们要把每一步流程都认真对待，找出所有潜在的问题和障碍。跟软件产品一样，在流程的管理中也要有异常处理。我们不必做到精确预见每一个问题，但一定要保证：即使流程出错，它还能往下走。

工作中的流程

把不同领域的所有流程串到一起。这些领域包括：部署、监控、能力计划(capacity planning) 等等。从逻辑上讲，“部署”是软件开发周期的最后一环，所以它应该属于“开发流程”，而非“运维流程”。另一个例子是度量和监控。在开发领域，如果不理解底线标准和估算，就什么评估都做不了。把开发部门和运维部门的流程衔接在一起，也会让两个部门更好的配合、相互理解、承担共同的责任。最后还有个优点：文档只需要一份而不是两份（开发一份、运维一份），从而节省了资金。

自动化，自动化，还是自动化。构建或使用简单、可扩展的工具（确保提供API, 机器可读的输入、输出 -- 参考 James White的文章：Infrastructure Manifesto ）。使用Puppet一类的工具做配置管理。要扩展这些自动化工具，使其能够支持多个领域（开发领域和运维领域），并且在产品的不同环境（开发环境、测试环境、发布环境和生产环境）中使用相同的工具(也叫end-to-end)。这样不但会在产品支持和管理方面带来经济效益，而且也可以在编写新代码的同时，进行产品的发布和管理。

最后，在构建流程和自动化时，要把KISS原则牢记于心。越复杂就越易错。只有简单的流程和工具才易于实现、易于管理和易于维护。

持续改进

不要停止创新和学习。当今技术发展的很快，客户的需求也往往如此。把“持续改进和持续集成” 加入到你的工具和流程中去，这也是运维人员向（优秀的）开发人员学习的好途径，可以学到诸如测试驱动开发等最佳实践。例如：可以向你的部署流程中加入单元测试。做监控时也应该增加些行为测试，提高交付质量。尝试用开发领域中的工具（例如Hudson ）在运维领域中做些工作（例如浏览数据(explore)、测量性能(measure)等等）。

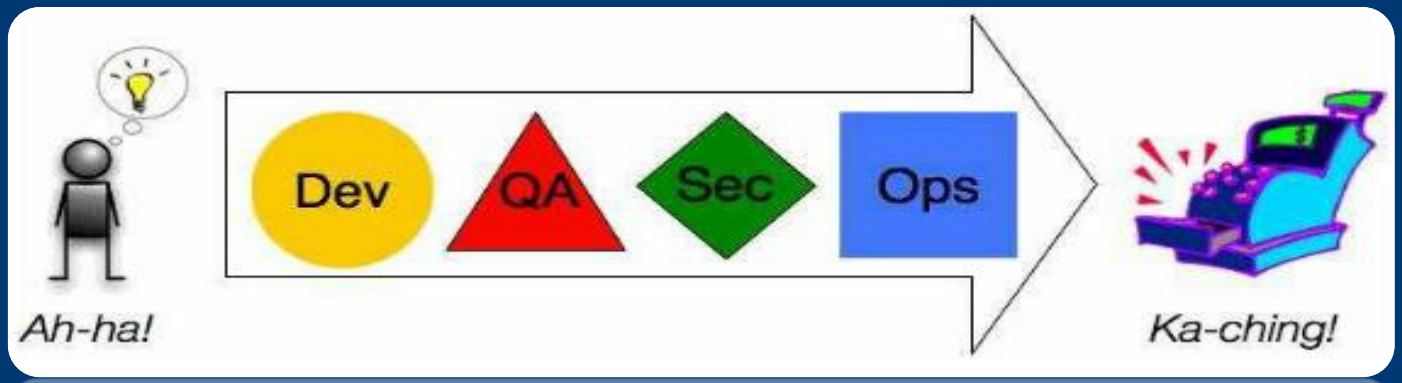
要不断的总结教训。要积极主动的、在不同领域寻找错误的根源。一旦收到错误报告，就果断把开发小组和运维小组找来，一起解决这个问题。有时候开发人员很简单的几次代码重构，就可以很好的避免底层运行环境的改变，减少运维人员的负担。总之，遇到问题时，开发部门和运维部门要密切配合、共同解决，而不是互相推诿、踢皮球。

业务流程 vs DevOps

DevOps不是个技术问题，而是个业务问题。

在任何公司里，最根本的业务流程都是这样：使一个最初的想法经过流程最终赚到钱。

需要各种各样的活动组成这个业务流程，这其中一些活动是技术驱动的，其他一些则是人驱动的。这里正是IT所有不同功能所发挥作用的地方。开发者、QA、架构、发布工程、安全、运维，它们都在这一流程中发挥自己的作用。



业务流程 vs DevOps

DevOps不是个技术问题，而是个业务问题。

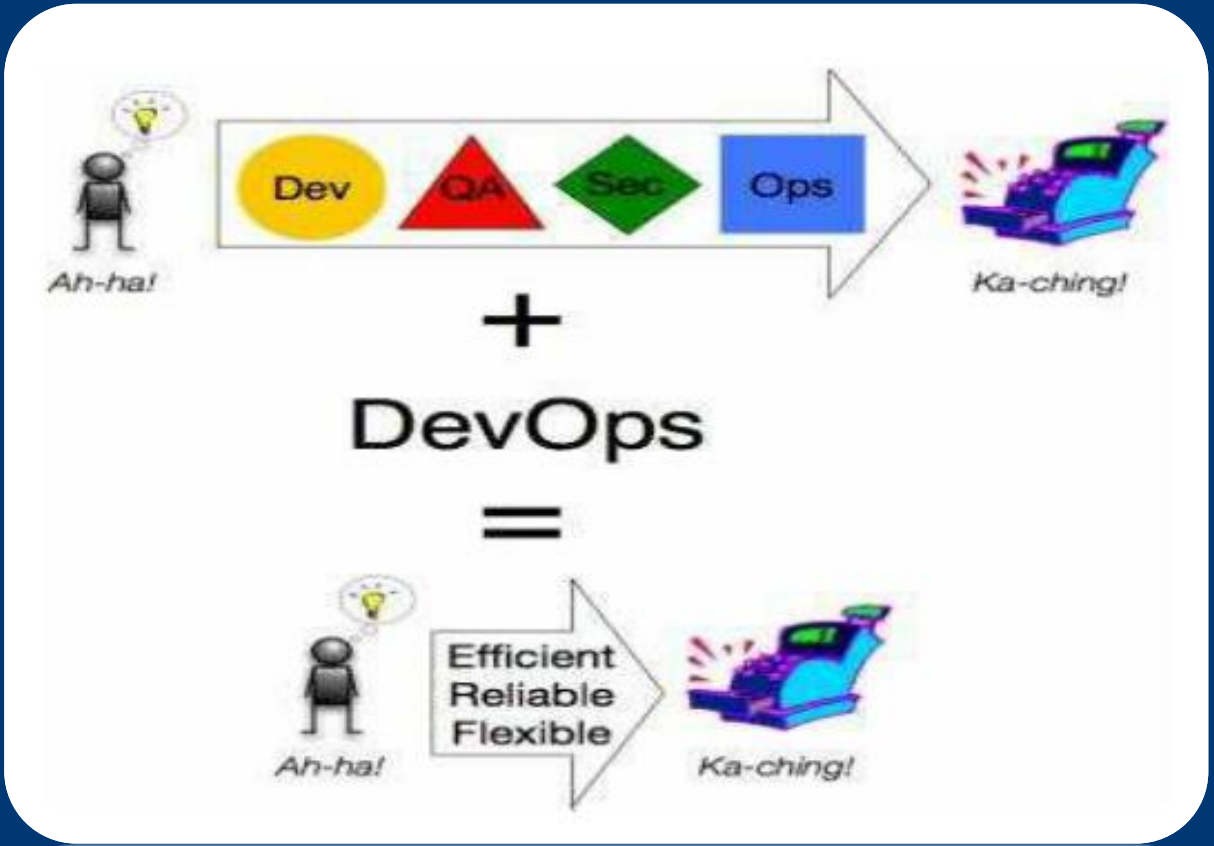
但是如果抛开这一业务流程的上下文，看看我们还剩下什么？是的，我们有一伙人，还有一些部门，它们各自做着它们自己的分内事。但是我们失去了真正做事的动力，到处是效率低下的工作、浪费、冲突和部门间的孤立。从表面上看，每个人都在仅仅为自己工作。

如果没有业务流程这一上下文，还会发生什么事呢？我们的工作将失去意义并最终消失。实现业务目标是我们得到薪水和花时间做事的原因。如果没有业务目标或者我们所做的事情根本对实现业务目标都没有助益又会怎样呢？糟糕，我们所做的一切都变成了一种爱好。想想吧，有谁会傻到给爱好付薪水呢。

业务流程 vs DevOps

DevOps不是个技术问题，而是个业务问题。

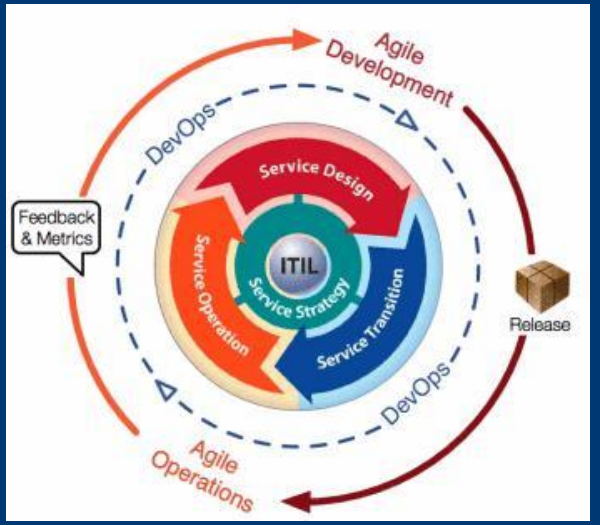
DevOps的立足点正在于对市场压力做出尽可能快速、高效和可靠的反应，从而实现业务目标。*抛开业务，谈论DevOps问题毫无意义，更别提花时间解决这些问题了*。



敏捷 vs DevOps

如果DevOps和敏捷所要达到的目标听起来很相似，那是因为他们的目标就是一致的。敏捷与DevOps的实质是一种思想，一种高效组织企业运作、产生价值的思想。但是敏捷和DevOps是两个截然不同的事物。

我喜欢将敏捷和DevOps描述为两个相关联的思想，它们都有一个共同的祖先，这个祖先就是精益，但是它们关注了不同的层面。敏捷深度关注于改善一个主要的IT功能（交付软件），同时，DevOps关注于对跨IT功能的流程和交互的改善（它拉伸了整个开发生命周期的长度，使其包括了运维）。



DevOps是ITIL的一种实现和延伸方式，为开发、发布、运营、改进做流程指导。

敏捷 vs DevOps

敏捷的价值体现在：及时响应需求变化，高效地组织进行软件开发，快速交付（可以工作的软件）。

DevOps（DeveloperOperations）的价值体现在：与开发统一节奏，将开发交付的软件或版本及时部署获得回报。

二者的结合非常适用于当今的互联网企业：强调端到端的一气呵成，消除部门墙，降低内耗，降低成本，快速回报。

敏捷侧重于市场与开发间的配合，DevOps则注重开发与运维间的无缝衔接。

客户需求—需求分解与设计—小粒度迭代开发—质量控制—持续小粒度部署获取回报

敏捷 ——————————————————————————————————DevOps

敏捷 vs DevOps

共通的东西

- 价值导向；
- 按实际业务组建团队，多种角色在同一团队中紧密交流，共通推动产品从无到有的过程。
- 消除不同阶段间的阻隔，降低成本。

工具 vs DevOps

技术几乎能使所有业务流程更加高效、可扩展和可靠，但是工具始终只是工具。

新工具的使用可能使得原有的坏习惯和支离破碎的流程更加恶化，如果想在改善业务流程上取得理想的效果，那么必须明确为什么要使用该工具以及如何最有效的利用该工具。

明确我们的DevOps问题究竟是什么，以及如何改善流程以减少DevOps问题时，对工具的讨论往往变得非常简单（如果还值得讨论的话）。

让所有人都知道为什么需要这些工具以及期望中的业务流程改进是什么，这才是**重点**！

为什么叫它“DevOps”呢？

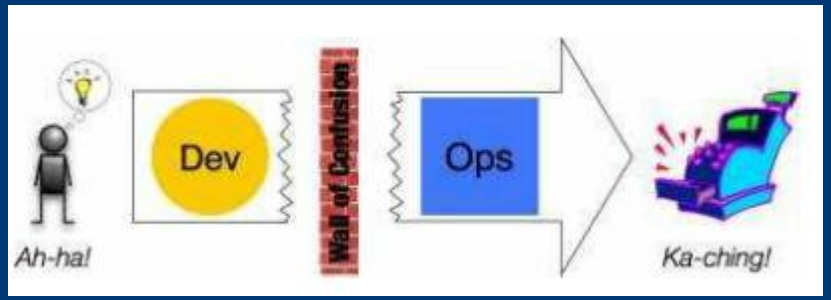
早期DevOps的一个不足之处是：没有直接地明确DevOps问题的真实范围，即它的问题域到底有多大。

解决问题：如何面对市场压力做出尽可能迅速的反应从而实现业务目标。

开发者文化与运维文化之间存在的冲突和脱节。

在开发和运维之间存在着一面混乱之墙，在这种情况下，大部分早期DevOps的注意力都放在在改善部署活动上。因为部署活动构成了整个IT组织的大部分工作，所以从部署开始改善，这是一个合理而自然的选择。

Patrick将他的第一次活动叫了DevOps。



DevOps的特征

DevOps是处于软件产品开发和运营之间的，它关注于如何开发对基础设施友好的软件，而这些基础设施正是商业软件赖以生存的环境。有时，DevOps也指代开发基础设施软件以及软件部署。DevOps有如下的特征：

① **编写复杂应用的能力，而不仅仅是简单的脚本；**

② **关注稳定性和无故障时间；**

③ **额外关注状态间的迁移；**

④ **关于营业收入的不同视角；**

⑤ **我们是自己软件的用户；**

⑥ **架构师、开发者、测试人员、产品经理、项目经理五位一体；**

⑦ **对意外更为关注；**

⑧ **工作于产品环境的QA；**

⑨ **先手动，再自动；**

⑩ **分布或者超级分布式环境。**

目录


DevOps
一个故事
DevOps概述
DevOps解决那些问题？
如何实施DevOps?
DevOps与ITIL
总结

Why DevOps?

DevOps是一个旨在为这个充斥着应用程序/服务开发、测试、部署、弹性、和监测的纷繁世界带来次序的新生IT学科，它同时还能够确保软件的质量、安全性、可用性、可靠性和性能。

这个定义似乎并没什么特别新鲜的内容，您可能会说：“我们的日常工作中不是一直需要各部门间的紧密协作吗？”然而，就象很多敏捷实践（比如迭代开发、每日构建）一样，它们在“敏捷”一词出现之前就已被应用到工作中，但敏捷让这些实践有机地结合在一起，发挥出前所未有的作用。敏捷强调打破需求、开发和测试之间的隔阂，形成自组织的跨功能交付团队，而DevOps则又向前迈进一步，强调以业务目标为导向，推倒交付团队与运维团队之间那道墙，将敏捷开发理念延伸至运维领域，使我们在IT方面的投入可以快速地转化为业务价值，从而打通“需求提出”到“上线运行”之间的所有环节。正如Puppet Lib的运维总监Kartar所说：

Why DevOps?

它是一种旨在促进开发和运维两个团队相互合作、学习的思想、原则和框架。在一个DevOps环境中，开发人员和系统管理员建立关系、流程和工具，让他们可以更好的交互，并最终为客户提供更好的服务。

解决问题1：“混乱之墙”

“混乱之墙”

相互冲突的动机、流程和工具导致了这面“墙”的存在

- 解决开发行为与运维行为的脱节现象
- 消灭冲突和低效

①角色认知

②所用工具

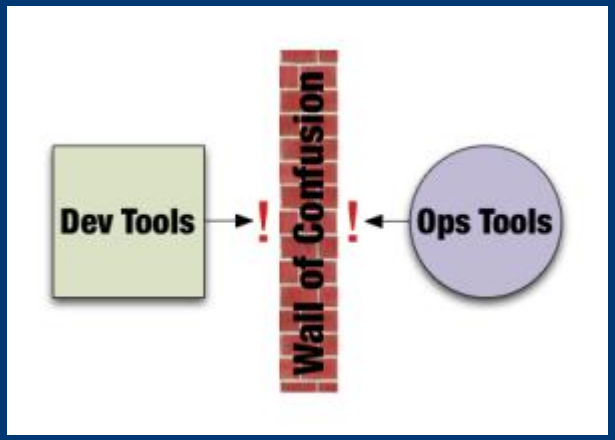
角色认知差异

以开发为中心的人员	以运维为中心的人员
通常认为变化会带来回报。企业依靠他们来应对不断变化的需求。因此他们被鼓励尽可能进行变革。	往往视变化为敌人。企业依靠他们维持正常业务运维和实施让企业赚钱的服务。由于变化会影响稳定性和可靠性，运维业务有理由对它说不。我们已经多次听到过如下统计数字：在所有宕机事件中有80%情况是源于自杀式的改变。

开发人员和运维人员认识世界的方法，以及各自所处的角色，存在根本性的差别。他们都认为自己的做法是正确的。的确，孤立的来看他们都是正确的。

更糟糕的是，开发和运维团队通常处于公司组织架构的不同部分，通常具有不同管理者的和竞争关系，而且通常工作在不同的地点。

所用工具差异



让混乱之墙更坚固的还包括开发和运维工具之间的错位。看一下开发者要求和日常使用的常见工具，再看一下系统管理员，你会发现两者存在很大不同，开发人员没有兴趣使用运维人员的工具，反之亦然；而且两部分工具之间也不存在重要的集成。即使在某些工具类型上有一些重叠之处，使用方式也完全不同。

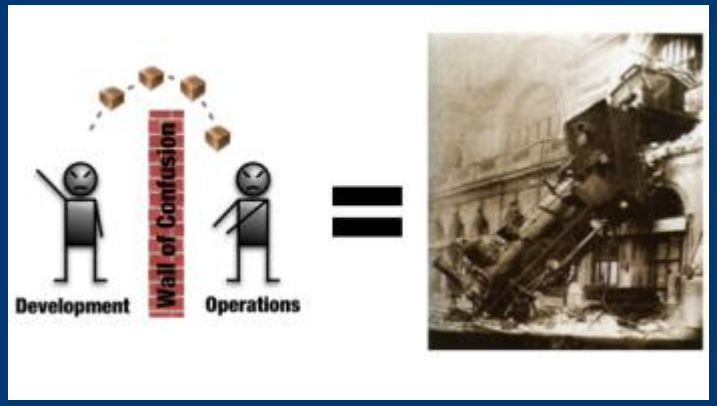
当应用程序变动需要从开发团队推向运维团队时，混乱之墙的存在则将变得更加明显。有人将其称为一个“版本发布（Release）”，有人则称其为一次“部署（deployment）”，但有一件事情是公认的，问题可能会随之而来。左图虽然是一个抽象化场景，但是如果你经历过这一过程，一定会感觉到它的真实性。

所用工具差异

开发人员把一个软件版本“扔”给墙对面的运维人员。后者拿到该版本产品后开始准备将其部署。运维人员手动修改由开发者提供的部署脚本或创建自己的脚本。他们还需要修改配置文件来适应与开发环境大不相同的真实生产环境。最完美的情况是，他们重复在此前环境中已完成的工作；而糟糕的情况是，他们将引入或发现新的漏洞。运维人员然后开始进行他们自认为正确的部署过程。由于开发和运维之间的脚本、配置、过程和环境存在差别，这一部署过程实际上也是首次被执行。当然，期间如果发生一个问题，开发人员会被要求来帮助进行排障。运维人员会说开发团队给的产品存在问题。而开发人员则会回应称该产品在他们的环境下运行良好，因此一定是运维人员在部署的过程中做错了什么。由于配置、文件存储位置和过程的不同，开发人员诊断问题也并非一件易事。

没有一个可靠的方式来把环境回滚到此前已知的正常状态。本来应该一帆风顺的部署过程最后变成一场救火行动，经过反复测试之后才让生产环境恢复到正常状态。

部署阶段已经非常明显的需要DevOps理念来解决问题，但需要DevOps的绝不仅仅是这一阶段。正如约翰·阿尔斯帕瓦（John Allspaw）所指出的那样，开发和运维之间的协作需求在部署之前就已存在，同时也会在部署之后的长时间之内继续存在。



解决问题2：企业的需求

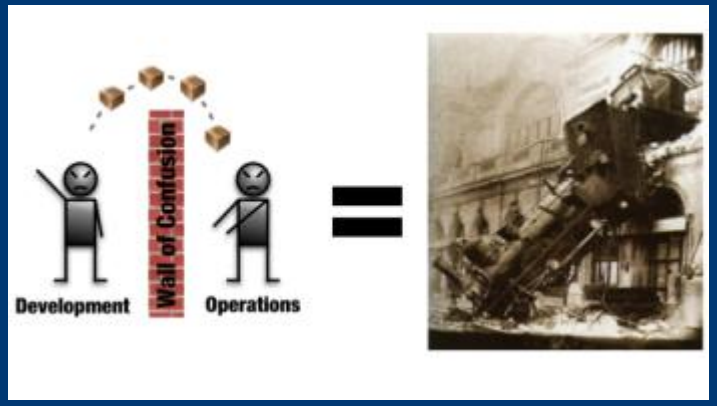
1. **快速响应**：能快速接纳需求、快速完成开发、快速上线投入使用；缩短发布和运维工作的周期；实现端到端的快速响应，快速响应业务需求、快速实现业务价值。
2. **质量**：开发人员对生产环境缺乏了解，在代码中引入了只有在生产环境才会暴露的缺陷；开发人员对非功能性需求缺乏关注，并且没有相应验证环境，导致非功能性缺陷；生产环境和测试环境缺乏有效管理，因为环境差异引入缺陷；部署和维护工作缺乏自动化，在发布过程中手工操作引入缺陷；缺乏针对生产环境的回归测试，导致缺陷不能及时被发现。
3. **敏捷拉通的尝试**：需要从运营维护工作本身的特点出发，引入符合客观情况的流程、技术和工具，才能有效改善运营维护工作的质量和效率。

所用工具差异

开发人员把一个软件版本“扔”给墙对面的运维人员。后者拿到该版本产品后开始准备将其部署。运维人员手动修改由开发者提供的部署脚本或创建自己的脚本。他们还需要修改配置文件来适应与开发环境大不相同的真实生产环境。最完美的情况是，他们重复在此前环境中已完成的工作；而糟糕的情况是，他们将引入或发现新的漏洞。运维人员然后开始进行他们自认为正确的部署过程。由于开发和运维之间的脚本、配置、过程和环境存在差别，这一部署过程实际上也是首次被执行。当然，期间如果发生一个问题，开发人员会被要求来帮助进行排障。运维人员会说开发团队给的产品存在问题。而开发人员则会回应称该产品在他们的环境下运行良好，因此一定是运维人员在部署的过程中做错了什么。由于配置、文件存储位置和过程的不同，开发人员诊断问题也并非一件易事。

没有一个可靠的方式来把环境回滚到此前已知的正常状态。本来应该一帆风顺的部署过程最后变成一场救火行动，经过反复测试之后才让生产环境恢复到正常状态。

部署阶段已经非常明显的需要DevOps理念来解决问题，但需要DevOps的绝不仅仅是这一阶段。正如约翰·阿尔斯帕瓦（John Allspaw）所指出的那样，开发和运维之间的协作需求在部署之前就已存在，同时也会在部署之后的长时间之内继续存在。

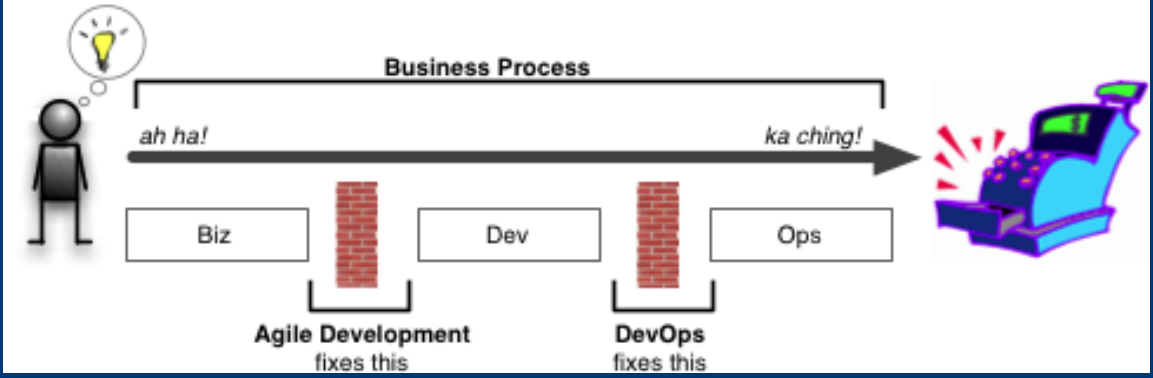


DevOps所带来的好处

- DevOps是一个非常强大的概念，因为它可以在众多不同层面上产生共鸣。
- 从开发或运维的一线人员的观点来看，DevOps可以让他们从众多烦恼中解脱出来。它虽然不是具有魔力的万灵药，但是如果你能够让DevOps奏效，则会节省大量时间，而且不至于打击自己的士气。显而易见，投入精力将DevOps落到实处，我们应该会更加高效、更加敏捷和减少挫败感。有些人可能会反驳称DevOps是一个遥不可及的目标，但这并非说我们不应该去尝试实现它。
- 对于企业来说，DevOps直接有助于实现两个强大战略性企业品质，“**业务敏捷性**”和“**IT融合**”。它们可能不是IT人士所担忧的事情，但是却应该获得掌握财政大权的管理者的注意。

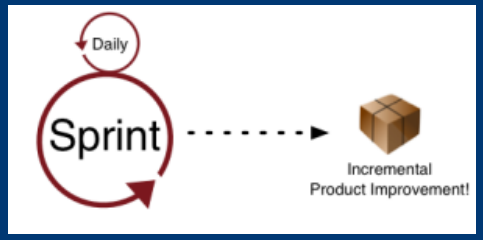
DevOps好处-IT融合

- IT融合的一个简单定义是，“企业渴望达到的一个状态，能够高效的使用信息技术来达到企业目标——通常是提高公司业绩或市场竞争力。”
- 通过从共同企业目标角度出发来校准开发和运维的职责和流程，DevOps有助于实现IT融合。开发和运维人员需要明白，它们仅仅是一个统一业务流程中的一部分。DevOps思想确保个体决策和行为应力求支持和改进这个统一的业务流程，无论你是来自哪一个组织架构。



DevOps好处-业务敏捷性

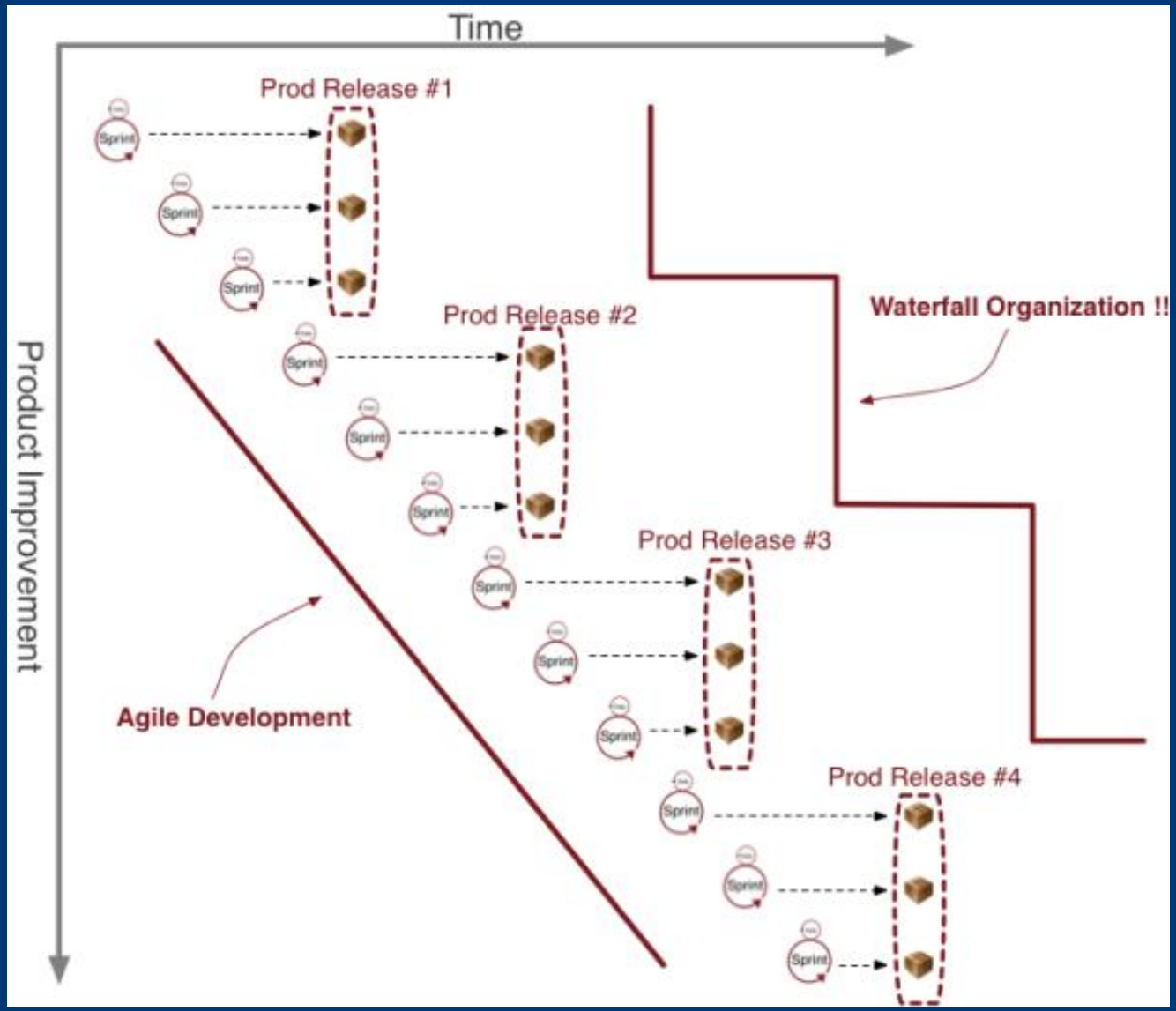
- 业务敏捷性的一个简单定义是，“一个机构以高效、经济的方式迅速适应市场和环境变化的能力。”
- 当然对于开发人员来说，“敏捷”有专门的含义，但目标是非常类似的。敏捷开发方法旨在保持软件开发工作与客户/公司的目标同步，尽管需求不断变化，也可以产生高品质软件。对于多数机构来说，迭代项目管理方法Scrum是敏捷的代名词。
- 业务敏捷性承诺，**在企业权益集团作出决策和开发者进行响应之间能够紧密互动和快速反馈**。看一下一家运转良好的敏捷开发团体的产品，你会看到一个与业务需求保持一致的稳定持续改进。



DevOps好处-业务敏捷性

- 但是，当你从企业角度回顾一下整个开发-运维周期，敏捷方法的相关优势通常会变得非常模糊。混乱之墙导致了应用程序生命周期的分裂。开发和运维分别按照不同的节奏进行。实际上，产品部署之间的长期间隔使得一个团体的敏捷工作变成了它一直试图避免的瀑布生命周期。当存在混乱之墙时，无论开发团体有多么敏捷，改变企业缓慢和迟钝的特点是极其困难的。
- DevOps使得敏捷开发的优势可以体现在机构层面上。通过考虑到快速、反应灵敏但稳定的业务运维，使其能够与开发过程的创新保持同步，DevOps可以做到这一点。

DevOps好处-业务敏捷性



DevOps好处-业务敏捷性

- 但是，当你从企业角度回顾一下整个开发-运维周期，敏捷方法的相关优势通常会变得非常模糊。混乱之墙导致了应用程序生命周期的分裂。开发和运维分别按照不同的节奏进行。实际上，产品部署之间的长期间隔使得一个团体的敏捷工作变成了它一直试图避免的瀑布生命周期。当存在混乱之墙时，无论开发团体有多么敏捷，改变企业缓慢和迟钝的特点是极其困难的。
- DevOps使得敏捷开发的优势可以体现在机构层面上。通过考虑到快速、反应灵敏但稳定的业务运维，使其能够与开发过程的创新保持同步，DevOps可以做到这一点。

能解决的具体问题

- DevOps就是试图避免重大失误，并更聪明且高效地工作。它是一种旨在促进开发和运维两个团队相互合作、学习的思想、原则和框架。在一个DevOps环境中，开发人员和系统管理员建立关系、流程和工具，让他们可以更好的交互，并最终为客户提供更好的服务。
- DevOps对下述问题的解决有很大帮助：
- 对变更的恐惧。
- 部署充满了风险。
- “它在我的机器上没有问题！”
- 部门壁垒。

敏捷开发现已渐成主流。而随着虚拟技术和云计算的成熟与发展，对原有的运维模式与要求也发生了变化。

收益-案例分析

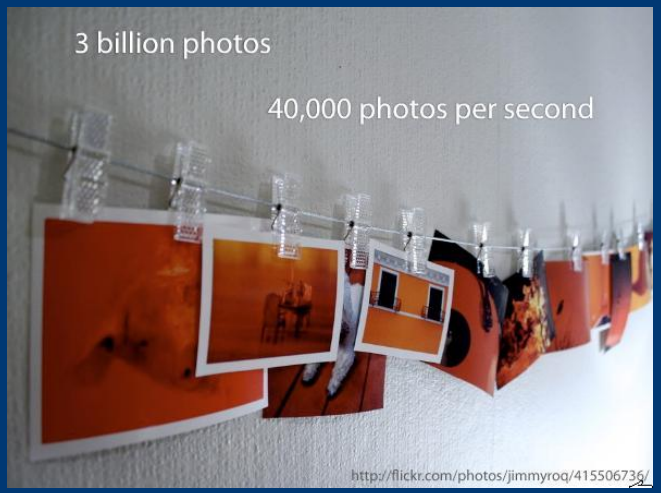
通过建设DevOps能力，软件组织能够明显软件产品发布和运营过程中的质量与效率。具体而言，可感知的收益包括：

① 缩短交付周期，新需求能更快投入使用并创造业务价值。
② 增加软件发布的可靠性，减少上线后的质量事故。
③ 减少发布和运营中的浪费，提高运营团队的工作效率。
④ 可视化度量软件交付过程，以便快速识别问题、持续改善。
⑤ 在开发与运营团队之间建立更加高效的协作关系。

收益-案例（Flickr）

Flickr是全球最大的图片共享网站。根据2007年的统计数据[6]，Flickr拥有超过850万注册用户，存放了超过30亿张照片，每秒钟响应4万个照片访问请求。

通过自动化基础设施、共享版本控制、自动化构建和部署、共享度量体系、强化沟通机制等手段，Flickr在保证网站稳定性和性能的同时，达到了每天能部署10次以上的需求响应水平，同时在开发团队与运营团队之间建立起互相尊重、彼此信任的协作关系。



收益-案例（某在线社交网站）

该网站从2000年开始运营，目前拥有超过3000万注册用户。随着业务发展，该网站的运营团队感受到来自业务负责人和最终用户的压力。根据ThoughtWorks所做的价值流分析，该网站从接纳一个需求到最终将其上线投入使用需要15～40天，其中超过50%时间是被浪费的。

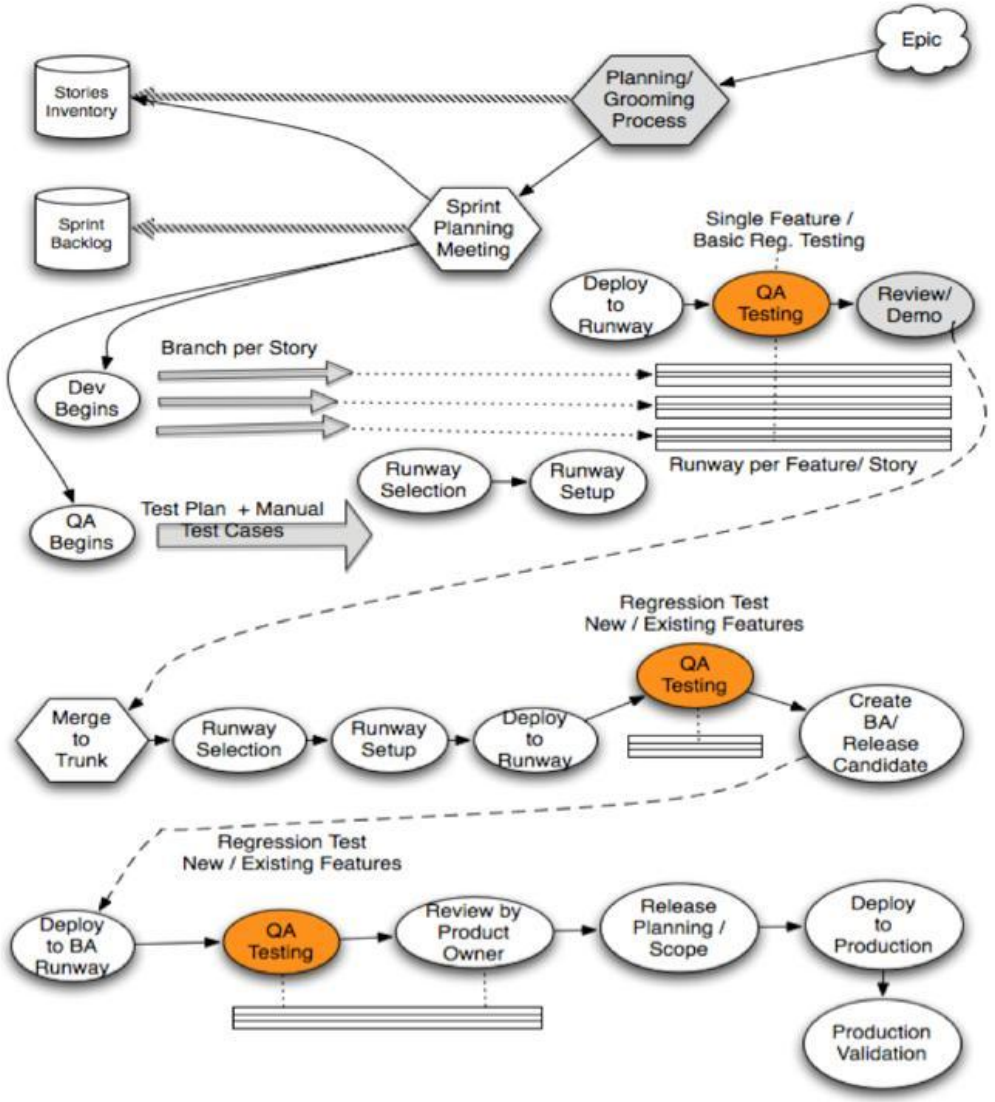
ThoughtWorks帮助该网站进行了DevOps能力建设，尤其加强了基础设施自动化、环境自动化、测试自动化和部署自动化能力，并改进了开发和运营团队的工作流程，使得典型需求的交付时间缩短50%以上，有效工作时间比达到90%以上，从而使该网站能够实现全面的业务敏捷。

收益-案例（某在线社交网站）

L/T: 15d – 40d

P/T: 8d – 16d

W/T: 0d – 21d



挑战

DevOps能力建设是一项系统工程，很多方面的因素可能对其造成影响。以下列举几项最常见的风险：

跨部门协作。很多大型软件组织都将开发与运营划分为不同的部门，而DevOps需要开发人员与运营人员无缝融合、紧密协作，这必然涉及部门之间的协调。如果处理不当，部门墙有可能严重损害软件组织交付业务价值的能力。

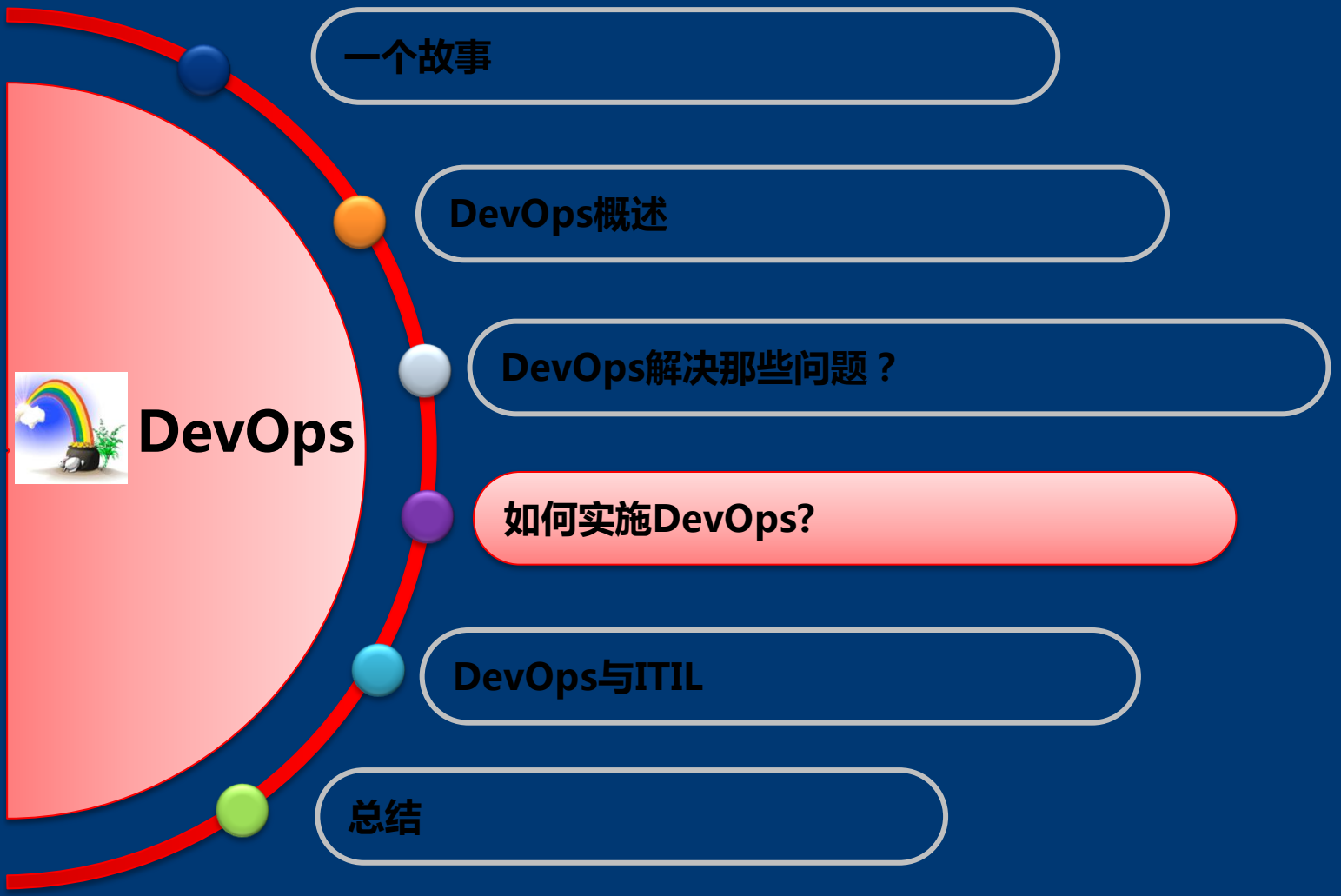
高层领导投入。相比传统的瀑布式发布，DevOps是工作方式的变革，涉及到技术、流程乃至团队文化的改变。如果缺乏高层领导的关注，或者如果高层领导只把DevOps看作小范围、技术性的改善，DevOps建设将很难收到预期的效果。

团队稳定性。传统意义上的“运维”是技术含量较低的岗位，人员流动率也相对较高。DevOps要求开发团队和运营团队（尤其是运营团队）掌握更全面的技能，尤其是项目自动化技能。如果不能保证团队相对稳定，学习投资就会被浪费。

挑战

软件的发布过程是一个整体系统，需要对其进行端到端的流程优化。ThoughtWorks采用精益价值流改善（Lean Value Stream Improvement）作为DevOps建设的框架，并在其中嵌入针对软件构建、发布、运营的知识和实践，以迭代方式管理改善活动，全程以可视化形式直观展现工作进展状态，从而最大程度地保障改善得以成功实施。

目录


DevOps
一个故事
DevOps概述
DevOps解决那些问题？
如何实施DevOps?
DevOps与ITIL
总结

如何将DevOps落到实处？

和多数新出现的话题一样，**发现问题的共性特点要比找到解决方案容易的多**。

要想实现DevOps相关解决方案，以下三方面需要关注：

1、评价和鼓励改变文化

改变文化和激励系统从来不是一件易事。但是，如果你不改变企业文化，兑现DevOps的承诺将非常困难。考察一个企业的主导文化时，你需要紧密关注如何评价和判断企业业绩。评价的内容将影响和刺激行为的发生。开发-运维生命周期中的所有当事方需要明白，在更大的企业流程中自己只是其中一部分。个体和团队的成功都要放在整个开发-运维生命周期内来进行评价。对于许多机构来说，这是一个转变，不再是孤立的来进行业绩评价，每一个团队不再是基于自己的团队来评价和判断业绩好坏。

如何将DevOps落到实处？

2、统一标准化的流程

要从软件交付的全局出发，加强各角色之前的合作，整个开发-运维生命周期必须被看作一个端对端过流程。流程的不同阶段可以采取不同的方法，只要这些流程可以被组合到一起创建一个统一的流程。与评价和激励的问题相似的是，实现这个统一的流程时每个组织可能会有略微不同的需求。

3、统一的工具

这是大多数DevOps讨论一直在关注的领域。这一点不令人吃惊，因为当技术专家在考虑解决一个问题时，第一反应往往就是直接跳转到工具讨论上。如果你关注Puppet、Chef、 Salt 或ControlTier等工具社区，那么你可能已经意识到人们对在开发和运维工具之间建立桥梁的重大关注。“基础设施即代码（Infrastructure as code）”、“模型驱动自动化（model driven automation）”和“持续性部署（continuous deployment）”都是可以划归DevOps旗下的概念。

统一的工具

一个版本控制软件库

它可以确保所有系统产品在整个版本发布生命周期中被很好的定义，且能够实现一致性共享，同时保持最新信息。开发和QA机构能够从中取得相同平台版本，生产机构部署已经被QA机构验证过的相同版本。

深层模型系统

它的版本系统清晰的描述了软件系统相关的所有组件、策略和依赖性，从而可以简单的根据需要复制一个系统或在无冲突的情况下引入变化。

人工任务的自动化

在依赖关系发现、系统构造、配置、更新和回滚等过程中，减少人工干涉。自动操作变为高速、无冲突和大规模系统管理的命令和控制基础。在从开发到运维的生命周期中存在许多不同的工具。工具选择和执行决策需要根据它们对端到端生命周期的影响来决定。

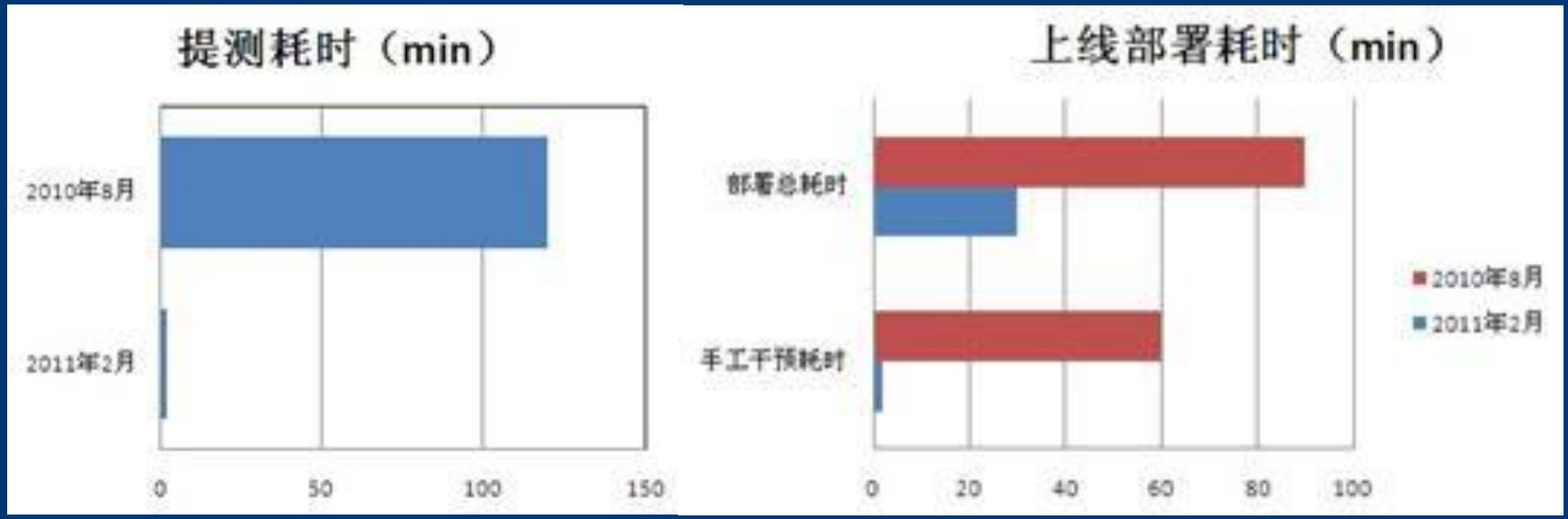
统一的工具

目前从开发到运维的生命周期中存在许多不同的工具，它覆盖了监控、准备、配置管理以及控制等方面，逐渐形成该领域中流程及工具标准化的生力军, 工具选择和执行决策需要根据它们对端到端生命周期的影响来决定。

作为DevOps运动的实践者，ThoughWorks的IT主管Ajey Gore介绍说：在ThoughtWorks内部，我们已经在全球22个办公室的500多台服务器和虚拟机上安装了Puppet。在公司内部，我们现在：

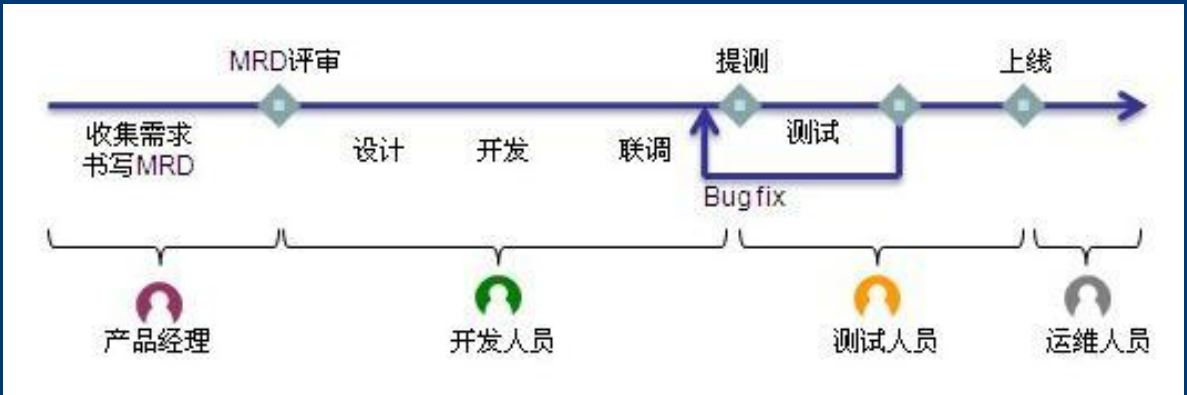
- 建立了Puppet的版本控制库。
- 每个Puppet服务器每30分钟会检查一次版本控制库。如果发现有变更，就会进行本地更新，并重新启动服务器。
- Puppet服务器也检查puppet脚本，反过来重新启动版本控制库。
- 如果有Puppet服务器出了问题，Nagios就会告诉我们那台服务器上的Puppet没有运行，我们就可以修复它了。

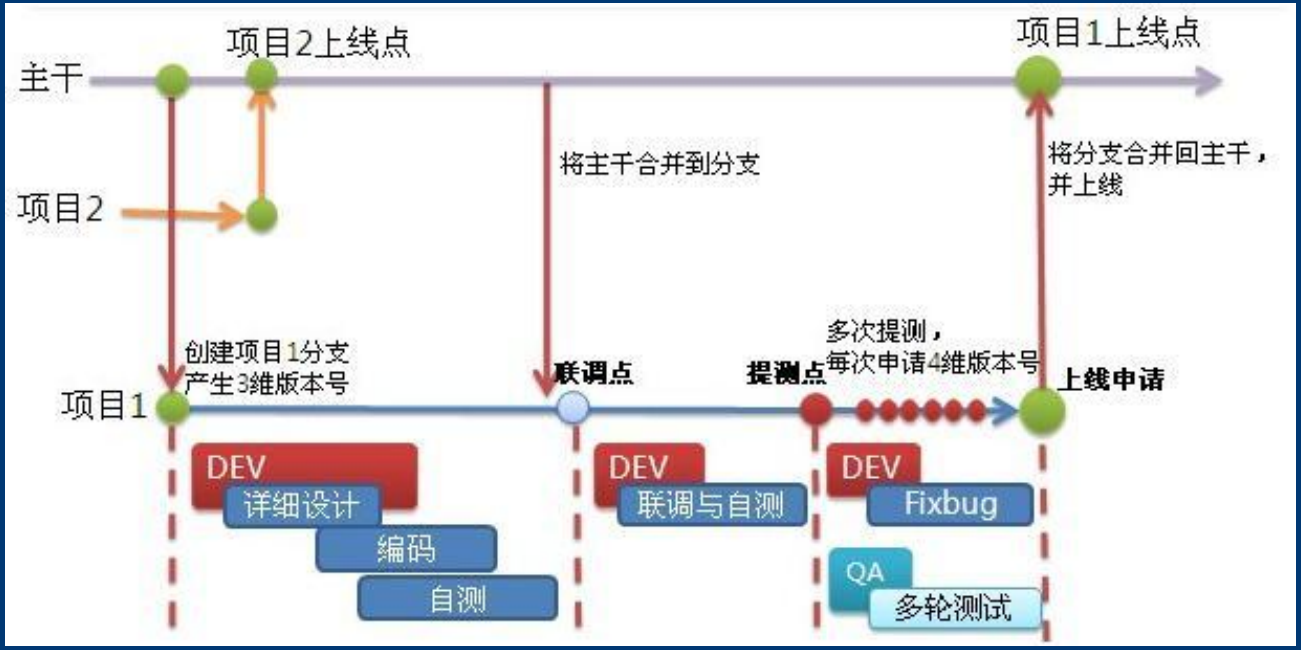
让数据说话



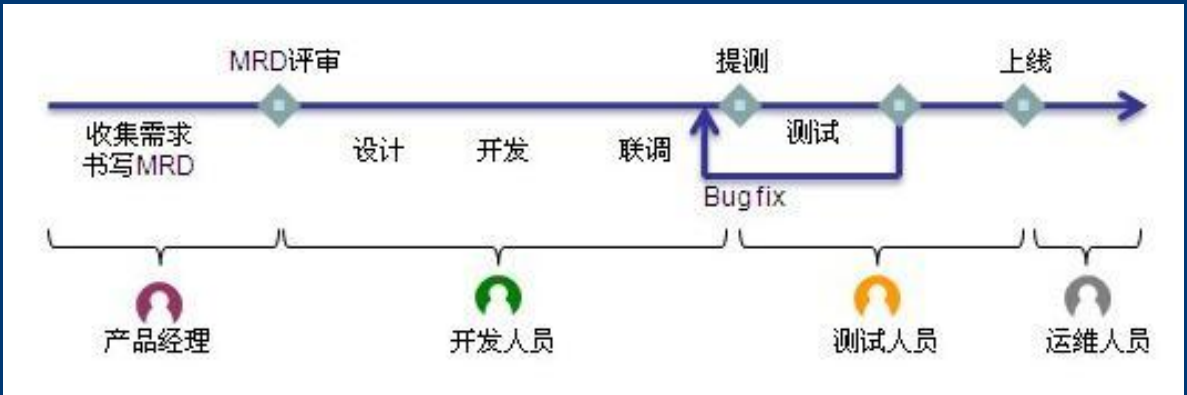
百度某产品线在半年内的变化

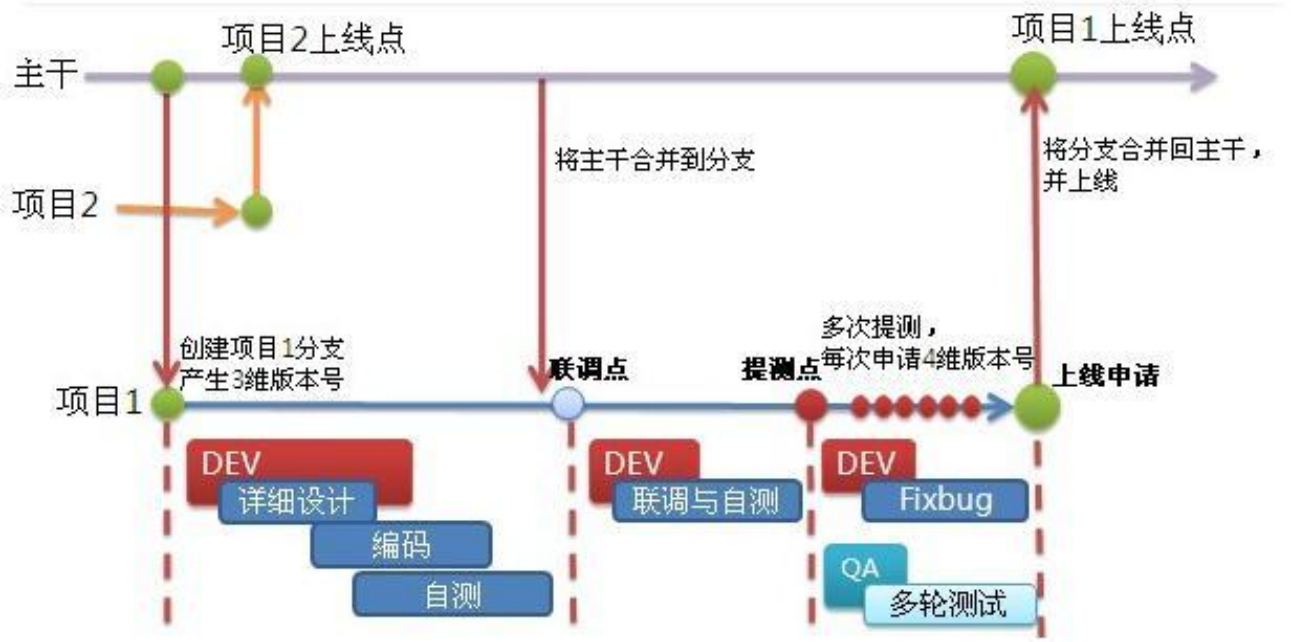
流程建模


MRD评审
提测
上线
收集需求
书写MRD
设计
开发
联调
测试
Bug fix
产品经理
开发人员
测试人员
运维人员


项目2上线点
项目1上线点
主干
项目2
将主干合并到分支
将分支合并回主干，
并上线
创建项目1分支
产生3维版本号
联调点
多次提测，
每次申请4维版本号
提测点
上线申请
项目1
DEV
详细设计
编码
自测
DEV
联调与自测
DEV
Fixbug
QA
多轮测试

流程建模


MRD评审
提测
上线
收集需求
书写MRD
设计
开发
联调
测试
Bug fix
产品经理
开发人员
测试人员
运维人员

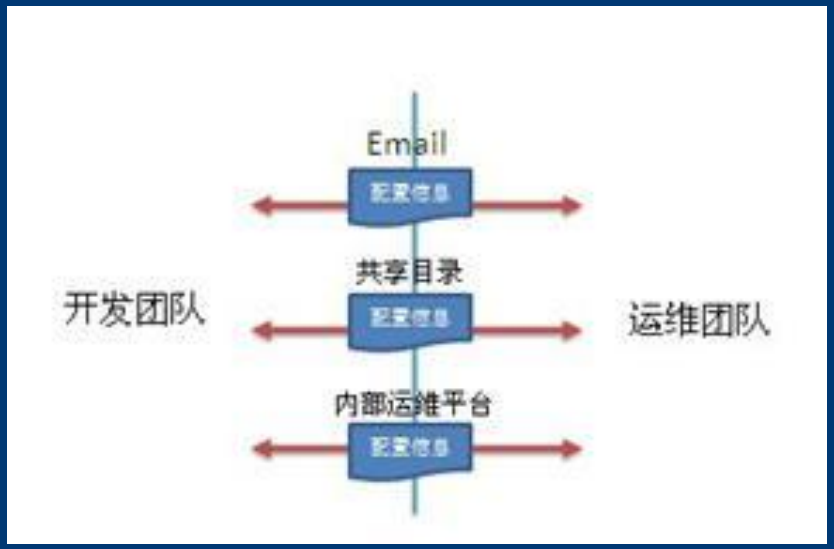

项目2上线点
项目1上线点
主干
项目2
将主干合并到分支
将分支合并回主干，
并上线
创建项目1分支
产生3维版本号
项目1
联调点
提测点
多次提测，
每次申请4维版本号
上线申请
DEV
详细设计
编码
自测
DEV
联调与自测
DEV
Fixbug
QA
多轮测试

发现浪费

从精益思想出发，为了尽早交付价值，必须首先找出整个流程中的浪费，并将其消除，从而提高流程效率，让“一个想法从提出到实现”可在最短时间里完成。

那么，浪费到底表现在哪里呢？

- 一些不必要的多分支开发，合并后发生问题的风险高。
- 推迟问题被发现的时间。
- 基于流程平台的沟通。
- 常规的例行工作很难自动化。



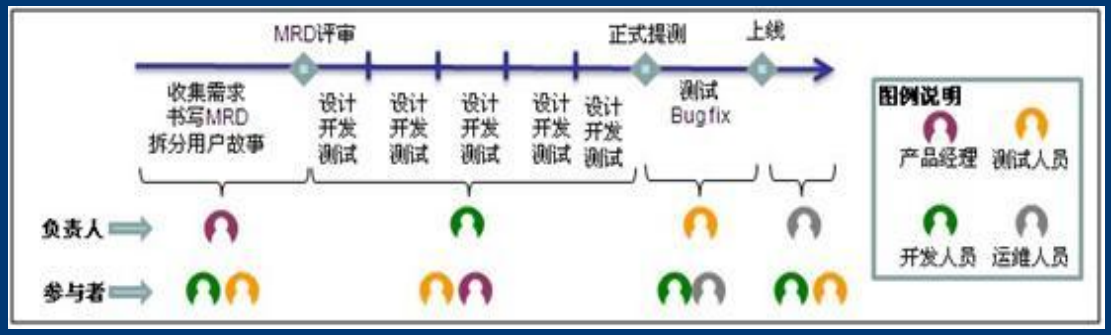
应对措施

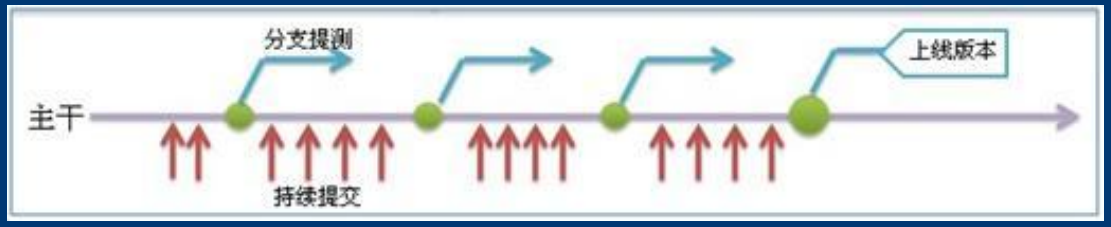
无人工干预方式的脚本自动化

- 自动化提测
- 统一配置信息源
- 常规流程脚本化

尽早发现问题，解决问题

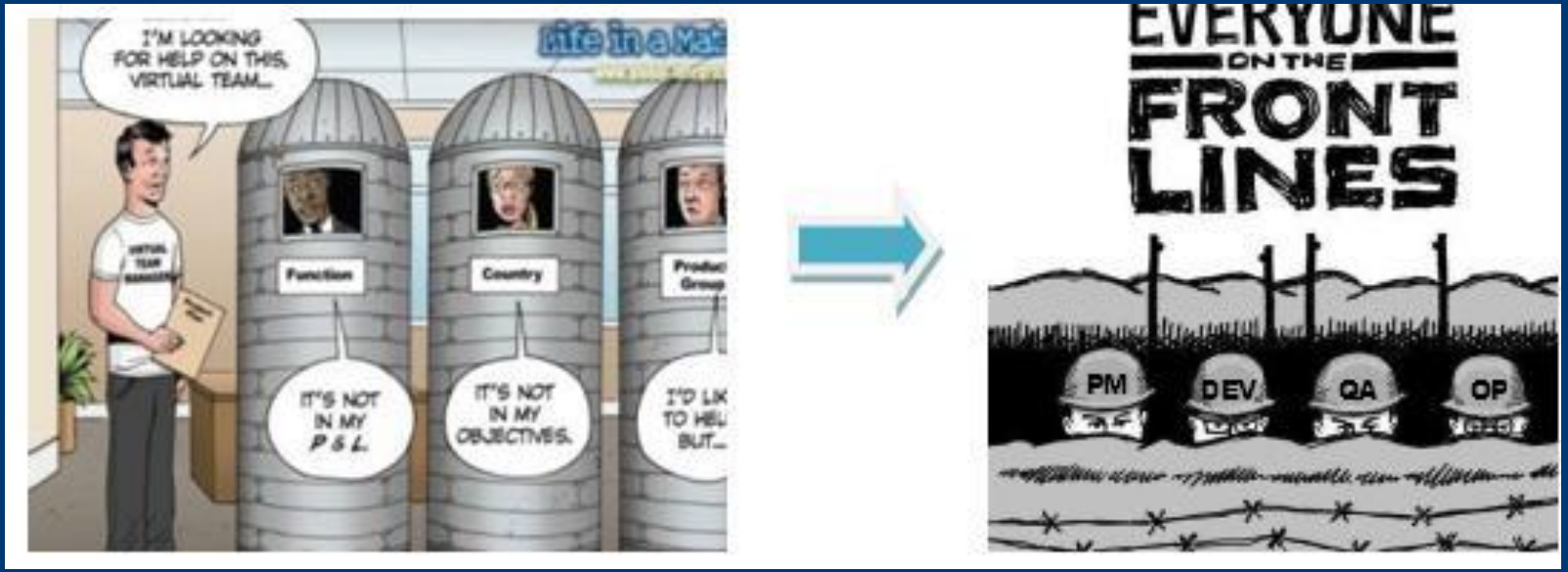
- "需求细分，及时开发，及时验证 “
- "主干开发，分支提测 “
- "持续集成"



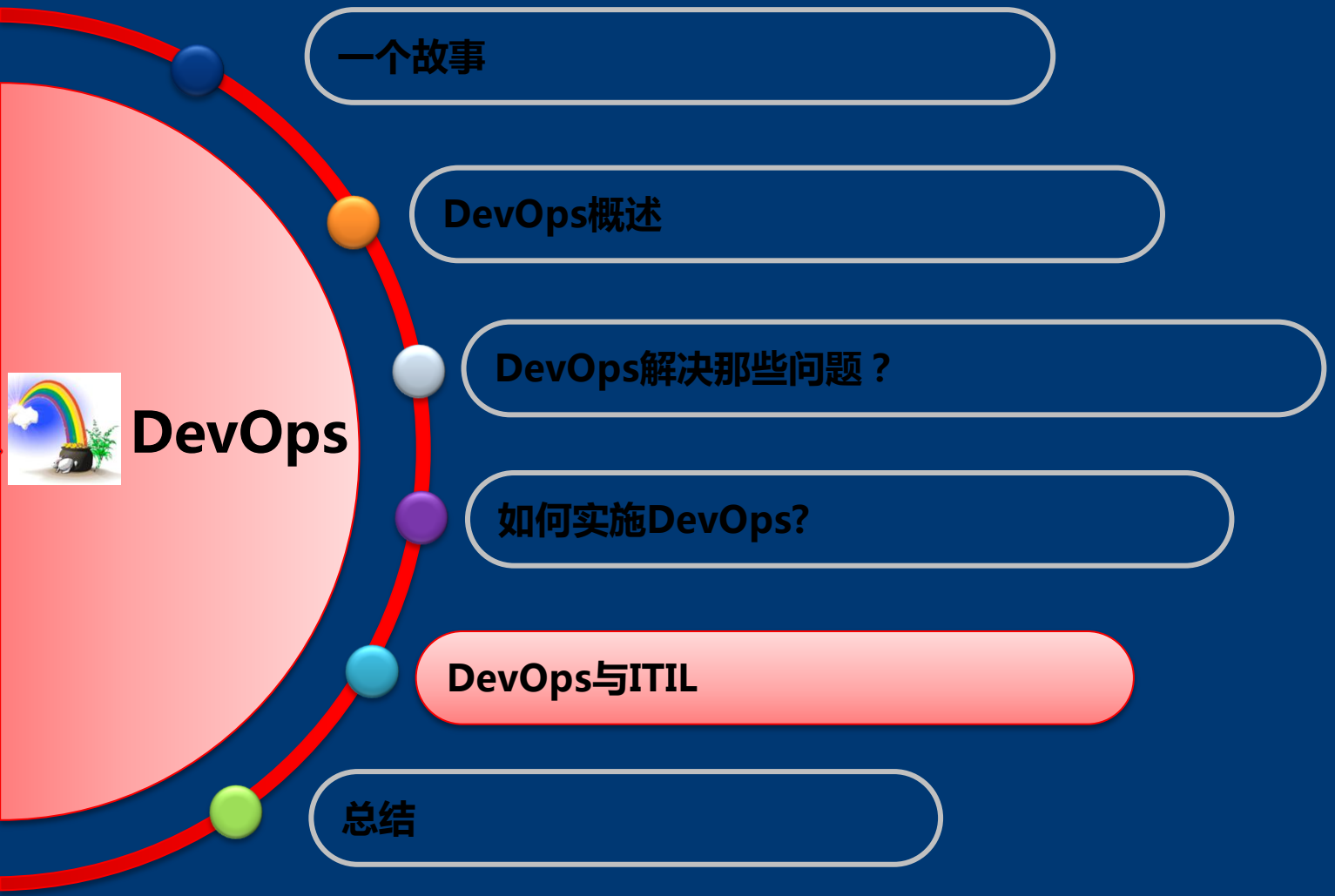


小结

从"碉堡防御"走向了"战线统一"



目录


DevOps
一个故事
DevOps概述
DevOps解决那些问题？
如何实施DevOps?
DevOps与ITIL
总结

明智地整合DevOps和ITIL

Thoughtworks敏捷发布管理工具GO的产品经理Jez Humber在其博文《持续交付与ITIL：变更管理》中指出，ITIL完全可以通过“持续交付”来实现一个轻量级的ITIL。

Tripwire公司创始人Gene Kim写的《Visable Ops》中也阐述了如何用现实且合理的方式实现ITIL。

ScriptRock的联合创始人Alan Sharp-Paul最近强调了企业需要明智地整合DevOps和ITIL（IT基础架构库）。

ITIL是一系列规划、文档、流程和合同的最佳实践，它似乎处于敏捷DevOps范畴的对立面。Alan认为，采用ITIL自有其价值所在，并且是企业界中的“必要之恶”。尽管如此，在他看来这也是荒谬的：

……正在尝试强行推动DevOps方法论进入大型企业，这与五到十年前强行推动ITIL流程的做法相同。在引入ITIL以前，大型企业中的环境就像是狂野的西部；然而在ITIL支配的秩序和结构中运用DevOps原则却是完全不同而且更危险的主张。

明智地整合DevOps和ITIL

独立IT咨询师Patrick Debois是DevOps术语的创造者，并与人合著了即将出版的《DevOps Cookbook》。他赞同DevOps和ITIL的整合是个大话题的说法，并继续道：

ITIL的流程拥有自己的位置。ITIL最初也像DevOps一样，随着实践者的工作而成长。（……）它的问题不在于理念，而是实践。它可以是实用的。这一点体现在ITIL专注于变更的控制。但许多人认为这是避免变更。

Patrick的观点得到了IBM的David Norfolk的支持。David Norfolk主张：ITIL拥有宝贵的逻辑模型，它们应该与ITIL过时的最佳实践分离。这将避免当DevOps专注于业务服务交付的时候，去“重新发明轮子”。Patrick反驳了“DevOps不应该改变ITIL”的观点：

我认为ITIL需要彻底改造自己。它将如何处理频繁的变更，并进行信息的分享？许多人正在安心地接受DevOps，我希望ITIL也能做到这一点。毕竟在ITIL v3中有许多与持续改进方面的交互。

明智地整合DevOps和ITIL

CA的副总裁和治理布道者Robert Stroud同意Alan的观点。他并不认为ITIL会完全“消失”，只不过使用ITIL流程的方法发生了改变。他认为“自动化和自动化控制的使用”是至关重要的。Patrick补充道，通过自动化提高再现性、自动批准变更以及将基础设施视作代码，将释放宝贵的资源：“变更咨询委员会将重新专注于重要的事情，而不是手工劳动和特定的步骤。”

这个话题的讨论延续了DevOps在企业中实践的探索，后者在近期逐渐获得关注。例如，InfoQ组织了一个专家小组，围绕着企业是否已经准备好迎接DevOps的话题进行讨论。诸如BMC等传统的ITIL拥护者则在寻找ITIL和DevOps间的协同效应，尝试借此统一概念和工具层面的方法。此外，在每次DevOps Days会议，包括即将在美国山景城举行的2013年DevOps Days，与会者都会询问：“DevOps如何与ITIL和Cobit共存？”

目录


DevOps
一个故事
DevOps概述
DevOps解决那些问题？
如何实施DevOps?
DevOps与ITIL
总结

总结

很多企业才刚刚开始敏捷之旅，还没人谈及"DevOps"运动，虽然还没有什么强大的工具支撑，但基于"提高效率"的朴素思想进行的过程改进也带来了"DevOps"效果。可见，DevOps听上去很神秘，但其实并不难。只要本着精益思想，聚焦于快速交付价值，不断发现并消除浪费，你也一定会有很大收获。

随着敏捷软体开发、迭代式开发、SaaS以及云端运算的普及化，以及未来即将需要扩大规模营运的公司来说，客户的需求越来越大且多变，也意味着开发及营运部门的沟通变得更频繁及复杂，因此需要一些DevOps专家来加强发布协调，并且部署协调以弥合开发与运营的鸿沟，迎合越来越快速及多变的软件开发流程。

本文来源

本文来源于搜索引擎收集整理，并非原创内容，以下为一部分原文的来源：

1、《DevOps，不是一个传说》：

http://www.infoq.com/cn/articles/devops-not-legend/

2、《明智地整合DevOps和ITIL》

http://www.infoq.com/cn/news/2013/05/integrate-devops-itil

3、《我眼中的DevOps》

http://www.infoq.com/cn/articles/devops-in-my-view

4、《DevOps不是个技术问题，而是个业务问题》

http://www.infoq.com/cn/articles/devops-business-question

5、《IT就像沙拉酱，有时你需要把它拌匀》

6、《DevOps，让持续交付成为可能》

7、《DevOps入门实战手册》