有幸通过几次难得的学习机会，粗略的了解到DevOps知识体系，略有所悟。下面通过三部分内容：DevOps知识体系、DevOps理论发展、DevOps流水线构建，谈谈自己对系统研发过程中DevOps应用和当前研发中心所欠缺空缺的内容。

### DevOps知识体系

##### DevOps是什么？

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。当前软件业从瀑布流道敏捷开发过度到如今，软件工程逐渐转向零件化、标准化、规模化、流程化、系统化发展，形成一种组合体。

##### 目的干什么？

整合研发交付的业务能力（产品管理）、研发能力（创新开发）、技术能力（技术研发）、工程能力（质量体系建设，包含后续运维），从而使整个项目管理能够快速响应、持续迭代、质量控制、安全交付。

##### 能力建设需要什么？

从持续进行方向上理解可分为:客户（大脑）、业务开发（感官）、质量（监控）、运维（肌肉）；

从DevOps能力分类可分为：持续交付、系统架构、产品流程、质量管理和监控、企业文化。

持续交付能力主要涉及生产过程中的版本控制（目前我们刚刚实行，但缺乏统一版本管理查询以及版本数据存储）、是否运用自动化部署流程（存在自动化部署，但各个流程独立部署，并非正真意义上一码多端部署）、能否持续集成、主干开发的方法、实施自动化测试（目前初步存在雏形）、是否进行测试数据管理（目前我们没有）、是否有信息安全管理（目前我们没有）。

系统架构和产品流程这个术业有专攻，从理解上来说主要是底层规模化，零件轻量化，对于客户的需求需要甄别功能主次，培养和赋能团队进行试验。

在质量管理和控制过程中，实现应用轻量化审批，以目标为导向。业务决策能够得到从应用到基础架构的全堆栈支持，据有前瞻性监控系统地监控系统的状况，通过可视化工具监控团队工作质量和进行沟通。

还有非常重要的，也是被严重忽略的就是企业文化的培养和建设。发展并支持生机型企业文化，鼓励并支持学习，注重辅助团队间的协作，提供工作所需的资源和工具，支持落实领导力转型（个人理解从关注分工、部门协作转变到关注项目小团队的协作上，适应大巴车到小轿车的组合，投资在技术提升和实践上，营造良好工作氛围，减少部署痛点、降低筋疲力尽/加班加点、提高NPS员工净推荐值上），能力建设的关键也在于企业文化的贯穿，是否据有这些领导力和执行力是软件交付效能的重要指标。

##### DevOps技术融合怎么做？

作为商业指标，任何工具的选择都是需要为客户创造价值。软件的交付效能就是：持续交付的能力、精益开发、精益管理、职业文化、职业满意度、团队氛围。

关键的技术实践就是：自动化部署、持续集成、主干开发、松耦合架构、版本控制、持续测试、监控和可观察性、数据库、安全。

整个过程版本化要包含整个项目，而不仅仅是代码。注意安全性的考量，自动化实现：持续集成、持续交付、持续部署。一个项目维护一个源码仓库，每天都有需求（注意是完成需求）代码提交，自动build、测试、部署，每人都能看到进度。关键词：代码、构建、测试、部署、纪律（规矩）。

代码交付是一种沟通和协作，代码版本的管理是项目版本中的核心，总述为一个版本管理、相同的依赖、配置、环境（客户需求）、多人的协作。

在质量控制上，越是技术的越是要自动化起来，功能测试也可以逐渐自动化，任何价值中，总有一个流动的方向，一个制约点，而任何不针对词约束（代码函数）点做的优化都是假象，也就是时间的不必要浪费。形成固定高效的流水线形式：

冒烟测试

安灯拉绳（任何环节失败，停止对应流水线，当前我们并未做到）

保持一致（一次构建、一致部署、一致环境，当前依旧是各自部署，实践中已经遇到环境问题引起的不必要时间浪费）

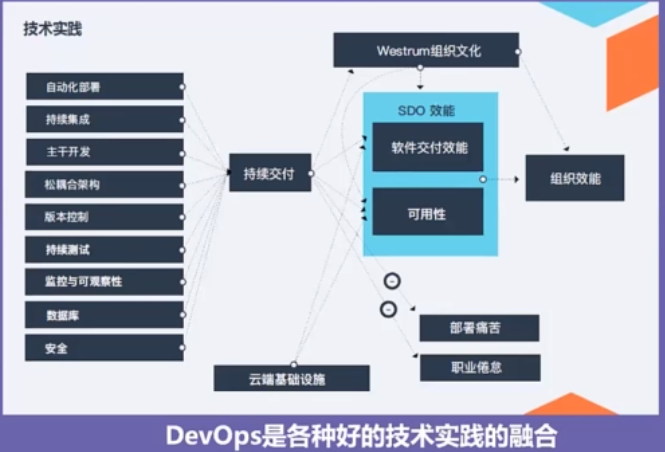
分成分级（明确质量出口标准，刚开始进行代码扫描以及接口规范、版本规范等）

快速反馈（自动化度量效率、质量，可视化流水线，当前正在准备和实施）

研发节奏（真实反映产品开发进度与质量，产品开发的心跳）

开发自测能力（推进质量标准，目前提交的提测版本质量反馈不佳，自测程度低）

在这个过程中，需要正确识别软件开发中的约束点，也是小团队基础：环境搭建（建立自服务环境）、代码部署（自动化部署）、测试准备执行（自动化测试）、紧耦合架构、组织和人（全功能、自组织、学习型）



##### 系统设计样式

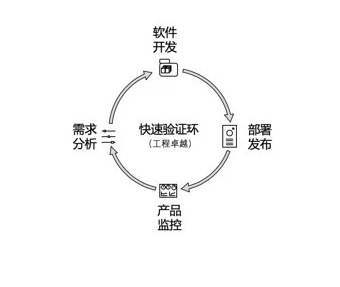
当前技术演进革新加速，要清楚研发目的，选用最方便的环境搭建，而不是自己再去维护机房，系统样式要抓住：一个中心，客户为中心，可理解为业务为中心或者价值为中心；三个要素，软件架构（DevOps、自动化系统）、服务架构、组织架构（自身组织架构匹配交付模式），架构决定了团队是否独立进行前进：变更系统的设计、测试系统、部署系统、而不依赖于外部人/系统进行沟通协作。

整个DevOps核心原则：消除浪费、增强学习、推迟决策、尽快交付、授权团队、嵌入完整性、着眼整体

## DevOps理论发展

##### DevOps持续交付1.0

持续交付发展1.0时，它的关注的就是从代码提交开始，一直到能够保证它能够正常在生产环境运行。它更强调的是快速把代码放到生产线上，让用户能用到。这也是目前DevOps聚焦推动的一件事，从代码提交到技术监管。



Etsy公司

2010年，该公司启动持续交付之旅，截止到2012年，每天可部署50多次。如果按每天工作8小时，相当于15分钟就会有一次部署

所有人向同一个代码仓库提交代码变更。每次提交代码变更后，立即进行自动化构建和测试（需要15分钟左右）。只要自动化测试通过，代码就会部署到生产环境上。即便是现在，在国内也是比较罕见的工作方式。

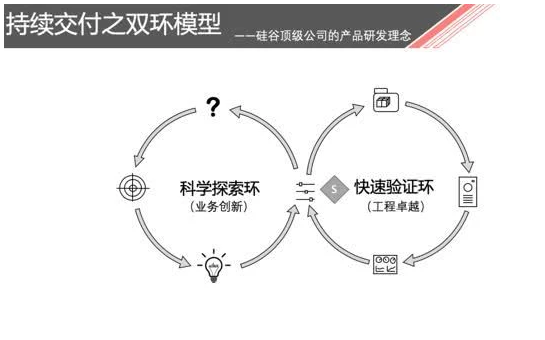
思考：为什么会有这样的工作方式呢？

总结：小功能实现（提交），快同步测试（扫描测试），轻容量发布（单个模块）

本次发展同样总结出形成DevOps的企业研发文化方向，主要包含：数据验证文化、快速反馈文化、工程质量文化。选择和使用工具，做到让员工自服务，要让机器等待，实现全流程的自动化过程。思考的方向是自上而下，要做到数据验证文化就要有快速反馈文化，要有快速反馈文化就需要有工程质量文化，但实际行动方向应该是自下而上的完成。而具备这样的文化需要打破部门壁垒，去除审批邮件，实现流程到岗负责制。

##### DevOps持续交付2.0

经多年的应用和实践，逐渐形成更加高效、敏捷的2.0模式。持续交付2.0的起点是业务价值探索，而非软件需求，强调业务问题的完整闭环。领导层更加需要考虑双环构造，即起点都是从想法（问题）出发，设定目标，分析洞察，找出尽可能多的解决方法，然后开发上线、数据评测，得到数据之后，决定是发布还是下架，然后再次快速迭代。

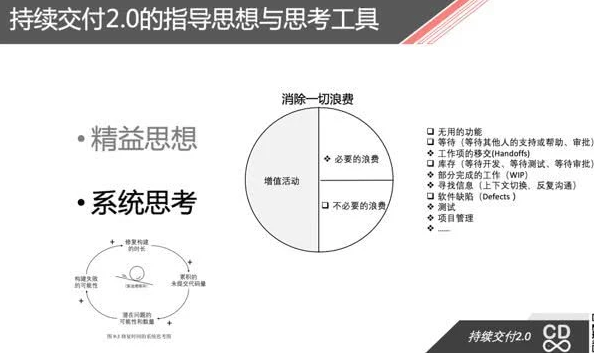


亚马逊的贝佐斯和Facebook的扎克伯格，他们所坚信的产品理念是“一万次实验法则”。这两句话真正体现硅谷互联网公司产品的研发理念。可理解为客户的反馈是产品定型的最终目标，如果能通过快速迭代发现适应更多客户的产品，所形成的价值转化，将是双环的效果体现。“解决一个业务问题，一定并不只有一种方案。”敏捷，就是极致地缩短每一次尝试、反馈和决策的时间。

**在持续交付2.0中有两个关键点：一是要做增值活动，二是浪费。**

精益思想的核心是“消除一切浪费”。软件产品的生产浪费在哪里呢？就是那些无用功能、等待、工作项的移交、库存、软件缺陷等。从客户价值出发，“测试活动”就是一种浪费，但对于企业来说，它是有价值的活动。所以它应该被归到“必要的浪费”这一类。项目管理也是一样，我们要消除所有不必要的浪费，同时，尽可能减少必要的浪费。如果你秉持消除一切浪费的思想，就可以从另外一个视角审视当前的软件开发流程和生产流程。

解决问题的办法很多，比如如何快速修复”：修复的方式有两种：一是再次提交代码，把问题修复，二是把引起问题的代码删除，这也是最快的修复方法。



**在持续交付2.0里面有四大基本工作原则。**

1. **坚持少做**：并不是说100个人做一点点事情，而是说当你在想到底哪一件事情对你最重要的时候，你才能够真正思考要投入多少资源把它完成。作为一个管理者，唯一要思考的就是怎么少做。只有坚持这一个观点，才能发现真正重要的东西。
2. **持续分解任务**：想要“少做”，就必须将大问题持续分解成多个小问题，在其中找到那几个真正最重要的问题，把它们挑出来，然后去做。而在做的过程中，因为问题比较小，所以也能够得到快速反馈，从而验证自己最初所定义的问题是否正确。
3. **坚持快速反馈**：快交付频率，获取更多的机会将功能及时发布到生产环境中，但是在保障软件质量的前提下。事实上他的产品研发理念改变了整个的研发模式和思考模式，即：只有高频短周期发布，才有很多机会可以试错。对于大团队来说，尽量减少没有必要的沟通，可以减少很多浪费。

4、**持续不断改善**：持续不断的改善“这一点来自于丰田管理。任何时候、任何地方你都可以改善。

我们必须学习，如果识别这些浪费，并在下一个目标里程中，将消除这些浪费做为正式目标。从而不断提升能力，消除它们，这也就相应增加了更多的增值活动时间。持续交付2.0本身关注的是端到端的业务协作，并不仅仅是产品开发、测试和运维，还要与市场、业务联动。可持续性并不是每周冲刺加班，那样很难具有可持续性。因为最开始可以拼体力，但一段时间之后，只拼体力就无法做到可持续了。

持续交付2.0应该成为企业的组织能力，应该具有可持续性。提供真正的业务价值，这个才是目标。

**三大领域：基础设施、组织管理和软件架构，这其中的基础设施又可以细分五个子领域。**

1. 组织管理机制可理解为：文化的建设、组织架构的搭建、人员结构的分布、激励体系的建立
2. 软件架构目标是：易测试性、易部署性、易扩展性、易监控性
3. 基础设置的五个子领域就要容易理解一些，主要是：业务需求协作管理、分支与配置管理、构建与环境管理、自动化验证管理、部署发布与监控管理

目前一般把这三大领域（包括五个子领域）称作“七巧板”，可理解为每一个企业都要根据自己的目标和实际情况，**拼出自己不同阶段的不同解决方案，递进发展，适合自己的才是最好的。**

**DevOps理论应用实践**

在当前持续交付的思想下，原有的经验或者会成为一种枷锁，如何灵活的处理，去改变团队成员的行为，需要以终为始，设计实施过程。不要落入陷阱：“文化”不落地，“我们只能做能完成kpi的事情”。

**实施文化变革的方法可以分四步走**：1、定义想要做的事。2、定义期望的做事方法。3、提供相应的培训。4、做些必要的事情，强化那些行为（主要就是实践）。

在Google是如何看待软件开发及开发人员生产力的呢？从整个介绍过程可以理解为：看整体的，不看个人的，看宏观视图，更好定位问题环节。关注内容例如：

1. 一个问题或者需求需要多久才能提交到代码仓库
2. 有多少次是-f（force），绕过来正常的提交流程
3. 发布频率如何
4. 一个部署包要花多久时间才能部署到生产环境上
5. 有多少次的Cherrypick，有多少次回滚
6. 代码测试覆盖率是多少
7. 有多少次Pager
8. CodeReview所花费的时长
9. 多少缺陷被部署到了生产环境中

**可参考案例：Google发展历程**

2001年谷歌

200名开发人员，3名测试人员

测试人员不受重视，加班加点手工较多

那些善于写自动化的人很快被拉入开发行列

测试团队当时实际称呼“测试服务”

重点在UI验证上，并要随时响应不同项目的测试需求

2005年的Google（开始文化宣传）

实际情况：

开发人员招收，有能力做功能开发的人，就应当做功能开发

测试人员：喜欢沉浸与老的做事方法，抱怨在SWE面前地位低下，又无力去改变自己的做事方式。

2007年的Google

测试团队有了更好的定位：**做产品线上的质量控制人**

仍旧是支持性质团队：局限于传统QA模型，仍然处于发布过程的后期才介入

**后期实行测试认证：**

一个团队完成一系列的测试任务，这个团队会得到一个通过认证的标识；

所有团队最初都是0级；如果掌握了基本的优秀代码习惯，就达到了级别1；后续继续梯度评估水平，最终达到级别5。

2008年~2011年

Google通过认证的团队：0~50~100~more~0

2014年的Google，有14000名工程师

每天提交了25000个ChangeList，尽早且频繁的合并与集成；有问题容易定位和回滚。

80万次持续构建，2PB的构建产出物

运行7500万个测试用例

**总结**：

从以上历程可得出，1、起始框架搭建基础；2、文化宣传入人心；3、测试团队定位：做产品线上的质量控制人；4、项目团队测试认证机制

不同时期所做的事不同，起始阶段处于框架搭建，人员配置期间，待人员较为完备，开始在项目团队进行文化宣传，培训介绍。测试团队需要转变后期介入思维，做产品线上的质量控制人，在产品立项开发前期便需要一同介入，对产品线全流程进行质量把控，代码规范，流程质量控制，需求完善情况监控等。对项目团队的自测和行为进行测试认证，形成固有习惯，调高研发开发质量，使更多的测试人员投入自动化研究，更多精力投入产品线质量控制中，减少沉浸式做事方法，对完成和达标的项目团队授予测试认证等级。

## DevOps流水线构建

##### 电子看板

1、看板的创建，主要对产品、开发、测试三方人员配置项目内容，实现流程搭建

2、需求分配管理，完成项目所面对的各项需求内容，建立独立分支

3、维护可视化流程，在项目过程中自动补充所面对的新增条件

4、自定义流程关注项，灵活关注重点流程、重点需求，需要项目经理进行时刻关注

5、可根据单个需求创建并关联分支，建立意识区分特性分支

6、泳道创建，bug优先、特殊需求、紧急调整

##### 流水线构建

根据项目单项需求，进行构建流程，进行全流程测试，检查过程阻塞方向，直至部署运行。

1. 构建过程需要各方协同
2. 开发过程符合质量控制
3. 自动化测试实现监控
4. 自动化部署同步交付（产品）

##### 代码托管

确定代码托管地址，实现自有代码托管服务

##### 测试管理

1、完成测试数据与测试代码的分离管理，实现变量和不变量约束，可控和不可控判断。

2、做好测试环境及资料的数据备份与管理

3、提交解决的问题要同步可视化管理

##### 包管理（运维）

实现项目中应用的微服务管理，做到自动测试部署，实现自动化运维（该过程基本与测试管理绑定）。

综上三方面学习理解，DevOps应用中，持续集成是对整个项目及迭代需求而设置，强调的是双环快速响应，大量验证尝试，自动测试管理，迅速部署交付。就目前研发中心团队而言，产品经理这一块是独立区分，负责了多个项目，需求开发过程未参与或者参与不深，从而将双环的其中一环给阻塞，产生的结果就是开发阶段需求不明，未能及时处理，新需求产生未能及时同步，影响安排计划，软件问题暴露，提交解决滞后。形成开发过程的波浪前进，而不是加速前进，缺少了快速响应。即使说有成型的原型、prd、UI设计，但这些在整个过程中是为了最大程度避免沟通理解错误，而并非就不需要人为接入结果的产生，这样的情况产生的是减速而不是加速，过程中产生的是一些不必要的浪费。

自动化方面目前成型的两个项目，只能算雏形，需要不断深入开发流程，完成整个项目包含的参数验证项。起步就需要不断推进代码开发的持续规范，同步推进达到自动化程序与程序自动耦合，介入对接口输出的规范，质量控制过程中需要加强后端对功能需求的必要判断，当前前端控制过多，后端校验不足，对自动化测试的实现存在阻碍。

关于性能，接触和学习的方向比较单一，且目前现有的服务器设置了瓶颈，在整个DevOps体系中，性能关注介绍的并不多，简单粗暴的解决就是实现云端服务器的配置提升，要想在代码上和优化上提些建议，有点才疏学浅。

实现监控，个人觉得这个可以和今后自动化测试框架相结合，测试完成可实现对部署监控的联动控制，实现安全点的播报和全渠道消息广播提示。同时可结合现有的一些成熟监控系统，进行配置关联，形成自有的一套流程。

运维角度来说，在DevOps体系下，更多的问题和质量监控在该流程前基本过滤，解封传统运维过程中的复杂部署。需要关注的更多是大容量和超大容量的解决，网络生命的周期的自动化运维，分钟级流量调优、秒级故障识别、分钟级故障定位，达到全网负载均衡和预测性维护，自动备份等当前前沿的关注点。