项目经历

智能配置软件开发平台

单机版 – 设计并实现新平台，解耦应用研发团队

动机：

1. 老平台代码难以维护（10+y，正常换代）。
2. 观察到团队研发流程之间的耦合性。

配置软件 – 终端类型兼容，版本后向兼容

终端应用 – 维护最新版本，终端类型繁多，终端应用的发布频率15~20次/年

导致配置软件发布、测试工作量巨大

提出“引擎+数据”（平台+发行包）的模式，避免频繁发布平台，节省了大量工作量

分布式智能终端管理系统

动机：统一配置、批量编程的需求，用户对智能终端使用情况的掌握

分布式CS结构，技术选型上，大量使用开源框架（JAVA, Tomcat, Hibernate, PostgreSQL），节省了研发成本

用户需求演进：

专用网络 -> 公共网络

移动端

**AWS部署**

**虚拟化**/用户为服务付费

**数据挖掘**（MongoDB）

作为主管系统架构师，负责系统架构设计，以及架构演进

作为产品负责人（PO），负责需求规划

协作为主管系统架构师，助部门经理安排项目计划和人员安排，

作为CSM和CSPO在部门内推动敏捷实践，持续集成，Hyper-V

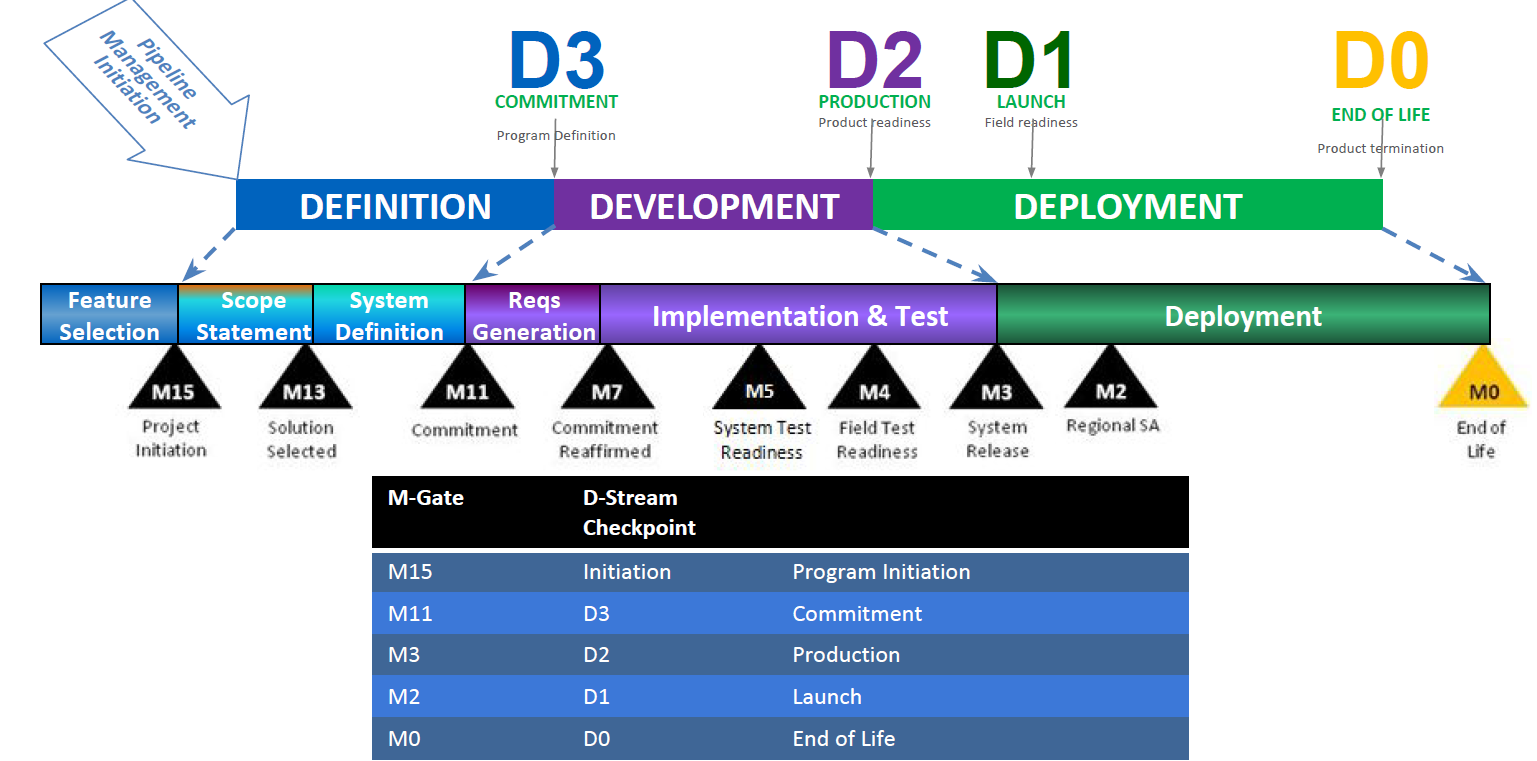
申请部门成立10+年来第一个专利，

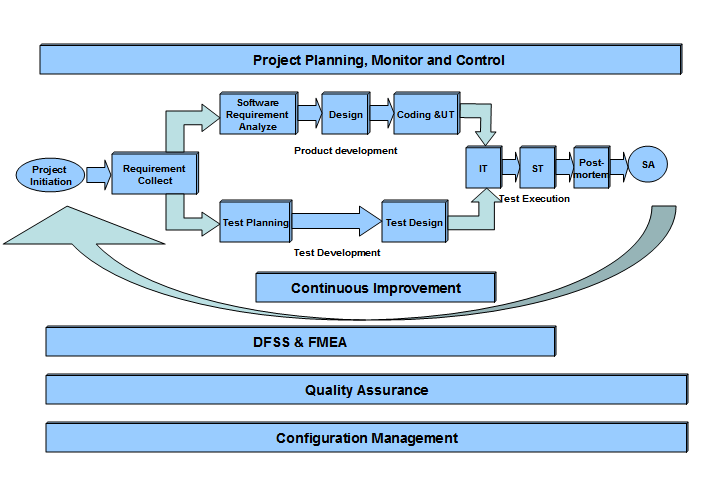
（人才梯队建设、组织转型、团队文化）

产品生命周期管理流程

2005 CMMI 5, 2009 ISO9001,

M Gates (M15 ~ M0) -> D Stream



[](http://zch49app07.sc.mcel.mot.com/spg/qms/Process%20Map/Process%20Map.bmp)

Plan phase – Review plan, risk plan, SCMP, coordination/dependency plan

Req phase – MRD, SRS

Design – design document

Test – test plan, test log

Audit report

Lesson learn

PDCA模式 - P(PLAN) D(DO) C(CHECK) A(ACT)

软件产品的复杂性决定了很难做一个“长期”的计划/预测。需求会变化，人员会更替，这些都给项目的管理带来很大的挑战。

对敏捷团队的合同可以理解成购买团队的“生产力”，实现的功能有客户在项目的执行过程中逐步确定。

从本质上讲，CMMI、ISO和Agile都是人们为了解决在软件生产过程中出现的**质量低下**、**进度延迟**、**预算超支**等问题，而产生的**标准或过程改进的模型**或**方法实践**，只是它们的关注点有所不同而已。CMMI和ISO关注为了实现组织软件生产目标，我们**应该做什么**？但却不关注如何做。而敏捷开发作为一个实践性方法，更关注**怎么做**。因此，在具体操作过程中，可以通过有效结合，能够使组织更快、更好地实现过程改进目标。

产品发布前的DT Scrub不是敏捷的实践，但是这个流程是产品发布的必须的。CMMI, ISO?

**敏捷宣言**

个体和交流重于过程和工具

可用的软件重于完备的文档 （**价值驱动**）

客户协作重于合同谈判

响应变化重于遵循计划 （**拥抱变化**）

预定义流程 vs. 经验主义流程

敏捷：在最短的时间内交付最高的用户价值

定期迭代，小版本交付，客户是最好的测试人员 – 互联网产品的灰度发布

Scrum提供一个用于组织和管理工作的框架，在此框架下，各组织可以添加相关工程实践特有的实现方法。

透明性 DoD

检视

调整

**敏捷实践** –

Sprint计划，需求细化，每日站会，Sprint回顾，Sprint评审

每日站会 – 提高项目的透明性，更加公平，人与人的差异会更加直接快速的反映

持续集成，自动化测试，随时有“可用的产品”，经常性的集成开发成员的工作，通过自动化的构建来验证，包括自动编译，测试，发布，从而尽快地发现集成错误。通过自动化工具，节省开发人员的精力投入。

**用“完成”来标识项目进度，提高透明度**

总的来讲，提高了团队的工作效率。建设了一种更加积极向上的工程师文化。

挑战：

分布式团队，市场/推广部以及高阶架构组在异地。

团队之间的依赖关系，很难真正完成一个完整的“用户场景”。

测试人员的角色转变，刚开始，每个sprint的最后几天加班加点，甚至认为不能按时交付是测试人员的责任。**敏捷测试** – 测试人员的角色需要从自己完成所有测试向训练整个团队养成良好的测试实践转变（写测试用例，自动化）。心态的转换，测试人员与开发人员从对立面转向一侧。测试人员绩效考核的变化。

度量质量 vs. 提高质量 用户质量 vs. 开发质量

发现产品的不可测试性，面向开发质量的测试，**内建质量**

​让变更更容易被验证

从软件经济学角度，一个团队是否敏捷转型成功了，要看它处理变更的单位成本是否下降了，这是一个更面向结果的敏捷说法

自组织 – 去PL化

SAFe – Agile Release Train 敏捷火车

跨部门跨团队的计划和投资组合（portfolio）

团队之间的协作和依赖

组织需要提升产品交付频率

虚拟化/卖服务（SaaS） -> **DevOps**

敏捷确保交付可用的功能，DevOps确保上线可用，拓展持续集成和发布功能到生产。

某云服务公司被思科收购的例子（4小时迭代）- Release Train

开发环境与生产环境的差异 – 虚拟机，自动化部署。

“精益软件”- 关注对客户有价值的上下文环境消除浪费。**敏捷主要关注具体的开发实践和项目管理。精益一体看待软件开发和它的整个业务环境**。

**价值流图**描述从开始到最终交付所有活动的一张图表。例如，从客户请求一个新功能到这个功能交付到用户手中的所有活动，每一步都必须对客户来说是有价值的。否则便是浪费的。

消除预测错误浪费的根本办法就是推迟决策。决策下得越晚，就越不容易失准而造成浪费。

挑战：

主干开发/Feature Flag，所有的feature作为整体发布，让上线变得有规律。

DevOps所倡导的高频部署给信息安全带来很大压力。信息安全审计通常需要以月为单文。即使采用了安全工具，通常也需要数小时至数天。

“分级检查”- 针对特定的安全隐患模块改动做检查。定期（天/周）检查。

持续集成的测试用例分Pre 和Post

相对于标准的单元测试，DevOps工作流依赖检测和恢复更多一点。

**快速**实现，**快速**发布，**快速**修复

**AWS的信条**，运维的需求具有最高优先级，功能要为性能让路。凡是会让运维变得复杂的需求都可能被工程师团队拒掉。

银行核心业务系统 -> 互联网分布式架构

**主机集中式架构** -> **互联网分布式架构**

集中，装有，封闭 开放平台，开源软件架构

**需求变更**风险，快速响应业务部门的需求，应用上线频繁，控制变更引起的运行事故变得越来越困难。

**挑战**：

系统安全性 IOE架构由于其专有和封闭的特点，在一定程度上更加安全。

运维管理 互联网分布式架构增加了系统的复杂性，设备和系统的数量也大大增加，无疑了运维工作的复杂性和工作量。

互联网 + 金融

金融 + 互联网

政策优势

资金优势 – 存款

差异化的场景

雷军互联网思维 - 专注、极致、口碑、快

**敏捷**：

1. 快速响应需求变化 – 拥抱变化
2. 在最短的时间里交付最高的用户价值 – 价值驱动

团队的透明度

挑战 – 分布式团队，团队之间的依赖性，测试人员转型

**精益/DevOps**：

4小时迭代例子，AWS信条

挑战 – 高频部署带来的安全隐患(DropBox)，主干开发策略

**敏捷主要关注具体的开发实践和项目管理。精益一体看待软件开发和它的整个业务环境**