

敏捷软件开发方法理论与实战

<http://morningspace.51.net/>
<mailto:morningspace@126.com>

议 题

- 敏捷方法概述
- 极限编程简介
- 敏捷实践案例
- 敏捷游戏

敏捷方法概述

开场白

军事历史就是一个在装备和灵活性的相对优势之间来回摇摆的钟摆。

—— 卡尔·冯·克劳塞维茨 《战争论》

- 盔甲骑士 vs. 布衣士兵
- 盔甲骑士 vs. 轻骑兵
- 坦克 vs. 轻骑兵
- 坦克 vs. 反坦克导弹



在IT领域，我们正好都在从装备统治一切的时代走出来。现在我们正进入一个唯有灵活性才是至关重要的时代。

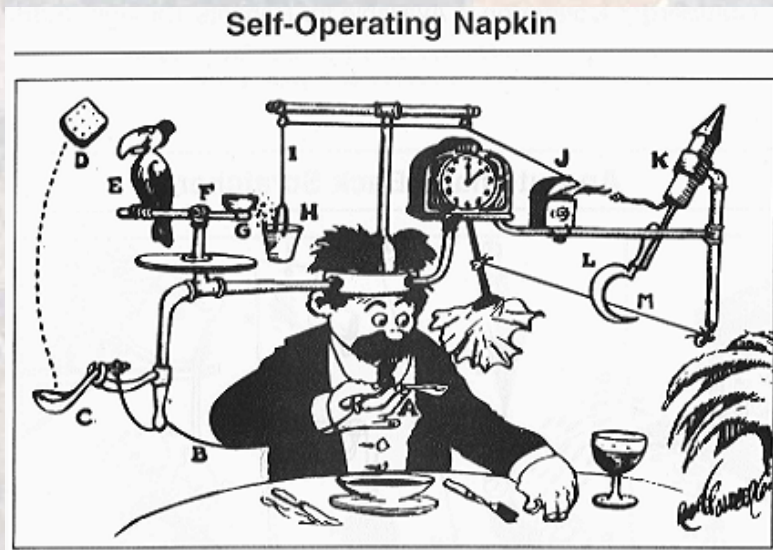
—— Tom DeMarco 《规划极限编程》序

- 工程方法 vs. 没有方法
- 工程方法 vs. 敏捷方法



工程方法 Engineering Methodology

- 借鉴了工程领域的实践，有着严格而详尽的规定，强调项目的可控性
- 官僚繁琐，要做太多的事情从而延缓开发进程
- 从泰勒主义，到精益制造



敏捷方法 Agile Methodology

- 以客户的商业价值为最终目标，更低的成本，更少的缺陷，更高的生产率，更高的投资回报，



- 对繁文缛节的官僚过程的反叛。轻装上阵，卸下包袱
- 只要求尽可能少的文档，认为最根本的“文档”应该是源码

敏捷联盟 Agile Alliance

2001年初，犹他州的Snowbird，由于看到许多公司的软件团队陷入了不断增长的过程的泥潭，一批业界专家聚集在一起概括出了一些可以让软件开发团队具有快速工作、响应变化能力的价值观和原则。他们称自己为敏捷联盟。敏捷联盟是一个非盈利性组织，其宗旨是推广敏捷方法和促进这方面的讨论。在随后的几个月中，他们创建出了一份价值观声明，也就是敏捷联盟宣言。

<http://www.agilemanifesto.org/>

<http://www.agilealliance.com/>

***Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler***

***James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick***

***Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas***

敏捷宣言 Agile Manifesto

我们通过亲身实践以及帮助他人实践，找到了更好的软件开发方法。
通过这项工作，我们认为：

- 个体和交流 胜过 过程和工具
Individuals and interactions over processes and tools
- 可运行的软件 胜过 面面俱到的文档
Working software over comprehensive documentation
- 客户合作 胜过 合同谈判
Customer collaboration over contract negotiation
- 响应变化 胜过 遵循计划
Responding to change over following a plan

虽然右项也有价值，但是我们认为左项具有更大的价值。

敏捷方法的两大核心理念

- 敏捷方法是“适应性”的而非“预见性”的

工程方法试图对一个软件开发项目在很长时间跨度内作出详细计划，然后依序开发。这类方法在需求和环境有变化时并不能发挥效力，其本质是拒绝变化的。而敏捷方法则欢迎变化，它通过迭代获得反馈。目的是成为适应变化的过程，甚至能允许改变自身来适应变化。

- 敏捷方法是“面向人”的而非“面向过程”的

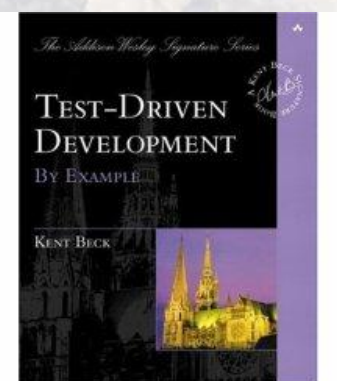
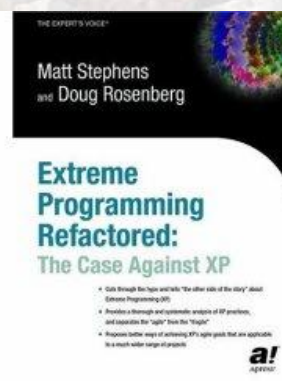
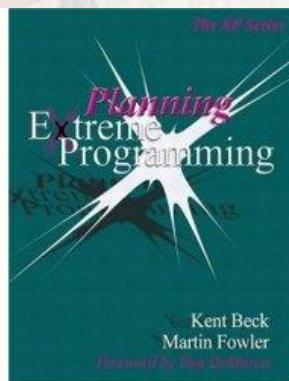
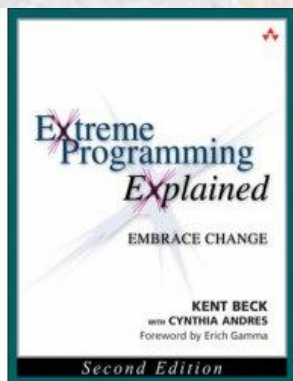
工程方法的目标是，定义一个过程，不管是谁用，都能运转良好，它隐含了实施者无需是高智商的。而敏捷方法认为，没有任何过程能代替开发团队的技能，过程所起的作用，是对开发团队的工作提供辅助支持。

—— Martin Fowler, New Methodology

有哪些敏捷方法（续）

• 极限编程（Extreme Programming）

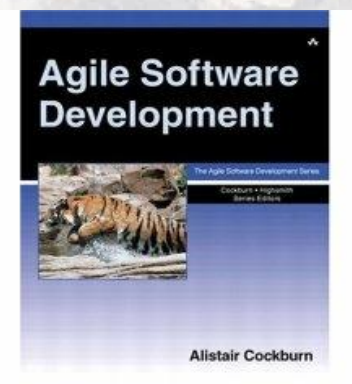
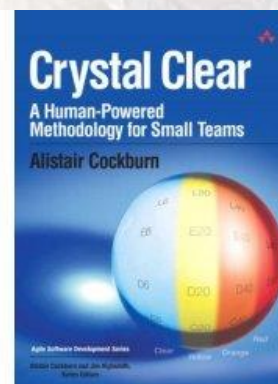
在所有敏捷方法中，XP是最具影响力的。XP起源于Smalltalk圈子，特别是Kent Beck和Ward Cunningham自上世纪80年代末的密切合作。目前，XP包括5条价值观和若干原则和参考实践。XP对测试极端重视，并且强调每次迭代中的代码重构，从而形成了“纪律性”与“适应性”的高度统一。



有哪些敏捷方法（续）

• 水晶系列方法（Crystal）

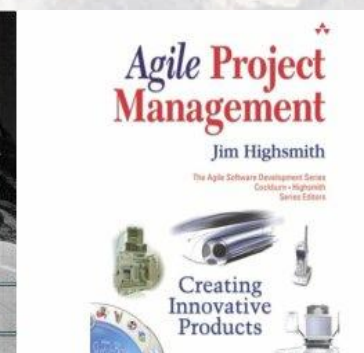
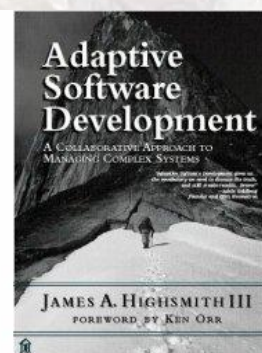
由Alistair Cockburn自上世纪90年代初开始发展而成。之所以是一个系列，是因为Alistair相信，不同类型的项目需要不同的方法，每种方法都有用武之地。与XP的高度纪律性不同，Alistair探索了用最少纪律约束而仍然能够成功的方法。虽然Crystal没有如XP那样的产出效率，但会有更多的人能够接受并遵循它。此外，Alistair也强调了每次迭代后的总结回顾，鼓励过程的自我完善。



有哪些敏捷方法（续）

• 适应性软件开发方法（ASD）

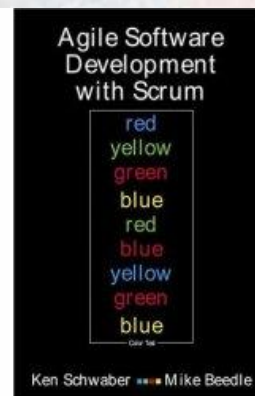
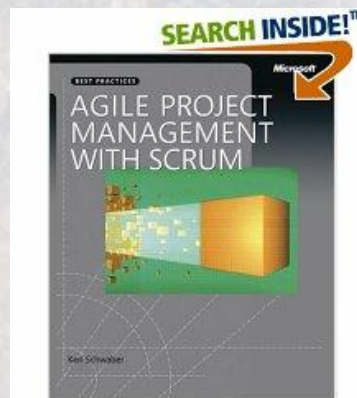
Jim Highsmith在他的书中探讨了如何将一些源自复杂适应性系统的思想应用于软件开发之中。他认为，在适应性环境里，偏离计划是在引导我们向正确的目标迈进。ASD的核心是三个非线性的、重迭的开发阶段：猜测、合作与学习。ASD侧重于“软”方法，这对那些从开发实践中提炼出来的方法如XP，FDD和Crystal而言，将是一个有益的互补。



有哪些敏捷方法（续）

• SCRUM

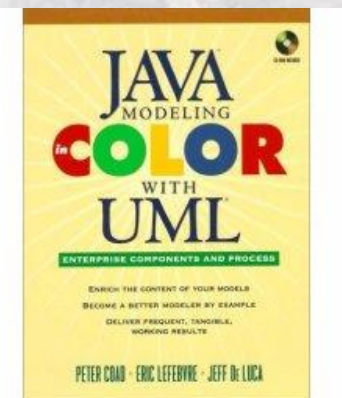
SCRUM把项目分成若干迭代阶段，一个迭代为期1个月，每次迭代之前明确要实现的功能，迭代期间，需求必须是固定的。人们每天召开一个15分钟左右的短会（称为一个scrum），会上大家报告目前的进展，以及第二天要干什么。SCRUM文献多集中于论述迭代阶段计划与进度跟踪。它与其他敏捷方法在许多方面都很相似，特别是它可以与XP很好地结合。



有哪些敏捷方法（续）

• 特性驱动开发（FDD）

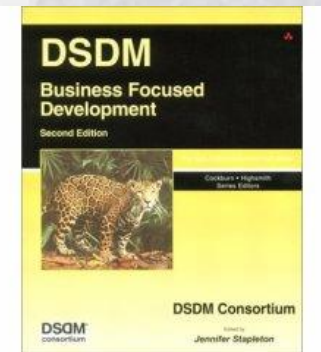
由Peter Coad和Jeff De Luca提出，致力于短期迭代和可用功能。FDD中一个迭代周期一般为两周。包括五项任务：建立总体模型，指定特性列表，针对特性逐项制订计划，特性设计和开发实现。其中，前三项在项目开始时完成，后两项则在每个为期两周的迭代中都要做。在FDD中，开发人员还被分成两类：Chief Programmer和Class Owner。前者负责设计和协调，后者负责具体实现。



有哪些敏捷方法（续）

- DSDM (Dynamic System Dev. Methods)

始于1994的英国。最早由一些想用RAD和迭代方式开发系统的公司组成了一个社团，开始时有17个成员，现在已超过了1000个，遍布英国内外。由于DSDM是由社团发展而来，与其他敏捷方法有所不同，它有专门的组织支持，有手册，培训课程，认证程序等。DSDM有一些基本原则，包括与用户积极的交流，频繁的交付。DSDM的一个周期在2-6周之间，它强调高质量的系统和对需求变更的高度适应性。



你是否应该使用敏捷方法

- 如果你已经习惯于没有过程，那么遵循简单的过程应该比遵循繁琐的过程更容易一些。
- 敏捷方法的一个主要局限也许是如何对付较大项目。但是，许多软件项目可以减少人数而不减少总体的生产率。
- 如果没有稳定的需求，就不可能进行稳定的设计，并遵循一个计划好的过程。这种情况下，敏捷方法是适用的。
- 使用敏捷方法最大的障碍或许来自客户。重要的是让客户理解：在一个需求不断变更的环境中，遵循可预见性过程对客户方是有风险的，同样对开发方也是有风险的。
- 如果你要采用敏捷方法，就需要信任开发人员，并让他们参与决策。如果你认为开发人员素质不够，那么你应该采用可预见性方法。

极限编程简介

第一个XP项目—— Chrysler C3

- 1995年，Chrysler公司的C3项目（Chrysler Comprehensive Compensation System）启动。其目的是，到1999年，新系统能替代原有的多个分散的工资系统，能够处理86000多名员工的工资支付。
- 1996年，项目陷入困境，公司聘请了Kent Beck作为项目的领导者。在找到问题症结之后，他与Ron Jeffries等人一起使用XP实践，重新启动项目。
- 期间，有关XP实践与原则的讨论，在Ward Cunningham的Wiki和OTUG的论坛中热烈的进行着。
- 1997年，C3系统发布了的第一个可用版本，虽然延误了几个月，但C3已经能支付10000人的工资了。不过，在随后的两年中，C3却没有发布新的版本。
- 1998年，Chrysler与Daimler-Benz合并，成立了新的Daimler Chrysler公司。
- 1999年，Kent Beck在XP的开篇之作《Extreme Programming Explained》中提出了极限编程这一创新的软件过程方法。
- 2000年，C3项目被遗憾的取消了，但XP却已经逐渐为人所熟知，并取得了长足的进步。包括Kent Beck, Ron Jeffries, Martin Fowler, Chet Hendrickson, Don Wells等人，都曾参与C3项目，他们也是今天流传于市的许多XP书籍的作者。
- XP还在不断演化中，2004年，Kent Beck的《Extreme Programming Explained》第2版面市。

什么是极限编程

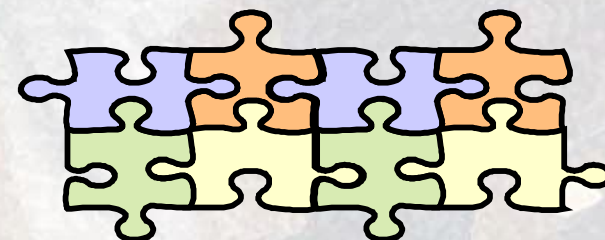
- 极限编程是一种适用于中小型团队在需求不明或快速多变的情况下进行软件开发的轻量级方法学。
- 极限编程是一种轻量、高效、低风险、柔性、可预测、科学而且充满乐趣的软件开发方式。
- 极限编程与其他方法论的不同之处在于：
 - 短周期，早期、具体和持续的反馈；
 - 递增的进行计划；
 - 依赖于口头交流、测试和源码来沟通系统的结构和意图；
 - 依赖于整个系统存在期间持续进化的设计过程；
 - 依赖于技术水平一般的程序员间紧密的协作；
 -

——《解析极限编程》第一版

为什么是极限的

XP将常识性原理和实践用到了极致

- 如果代码评审是好的，那么我们就始终评审代码（结对编程）
- 如果测试是好的，那么就让所有人都始终进行测试（单元测试）
- 如果设计是好的，那么我们就把它当作每个人日常工作的一部分（重构）
- 如果集成测试很重要，那么我们就在一天里多次集成并测试（持续集成）
- 如果迭代周期短些好，那么我们就让迭代时间以秒、分或小时来计，而不是周、月或年（计划游戏）
-



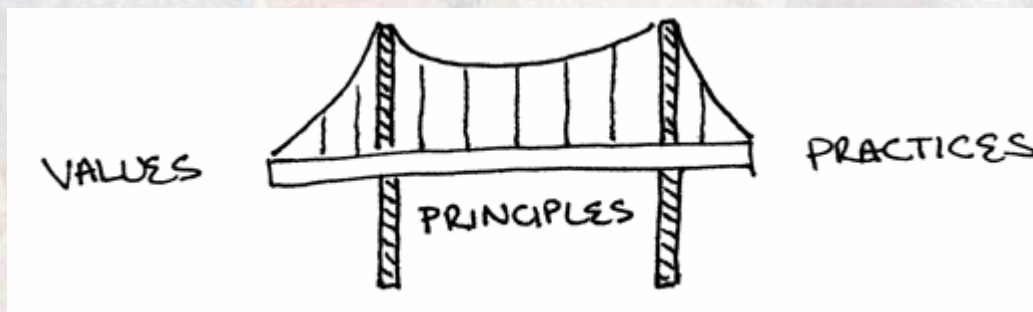
来自《解析极限编程》第2版的补充

- **XP is about social change.**
 - Giving up old, ineffective technical and social **habits** in favor of new ones that work.
 - Good, safe **social interaction** is as necessary to successful XP development as good technical skills.
- **XP can work with teams of any size.**
 - The values and principles behind XP are applicable at any scale.
 - The practices need to be augmented and altered when many people are involved.
- **Applying XP not adapt XP（下水的例子）**

XP的构成

- 实践 + 价值观 + 原则

- 实践是看得见摸得着的。可以不求甚解拿来就用。
- 价值观是某一情景下喜欢或不喜欢某事物的根源。显式的化价值观可以避免实践的生搬硬套。
- 原则介于价值观与实践之间。有助于更好的理解实践。
- 掌握了价值观和原则，实践是可以创造的



XP的核心价值观

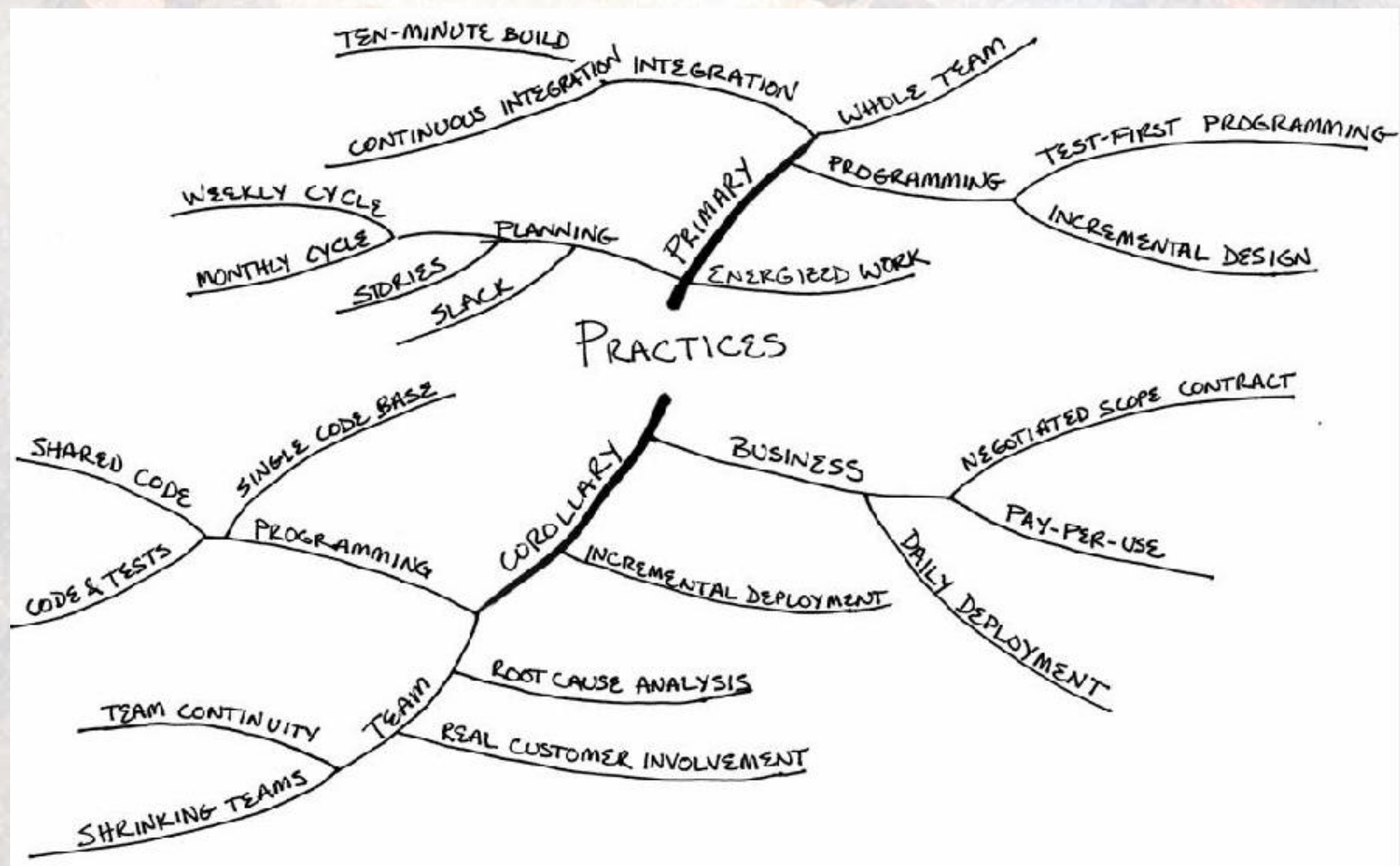
- 沟通（Communication）
- 简单（Simplicity）
- 反馈（Feedback）
- 勇气（Courage）
- 尊重（Respect）

所有价值观是一个有机的整体，而非彼此孤立
价值观也不是XP的专利，也许你有自己的价值观
在实际价值观与XP价值观不一致的组织中，XP是无效的

XP的原则

- **Humanity**
- **Economics**
- **Mutual benefit**
- **Self-similarity**
- **Improvement**
- **Diversity**
- **Reflection**
- **Flow**
- **Opportunity**
- **Redundancy**
- **Failure**
- **Quality**
- **Baby steps**
- **Accepted responsibility**

XP的参考实践



敏捷实践案例

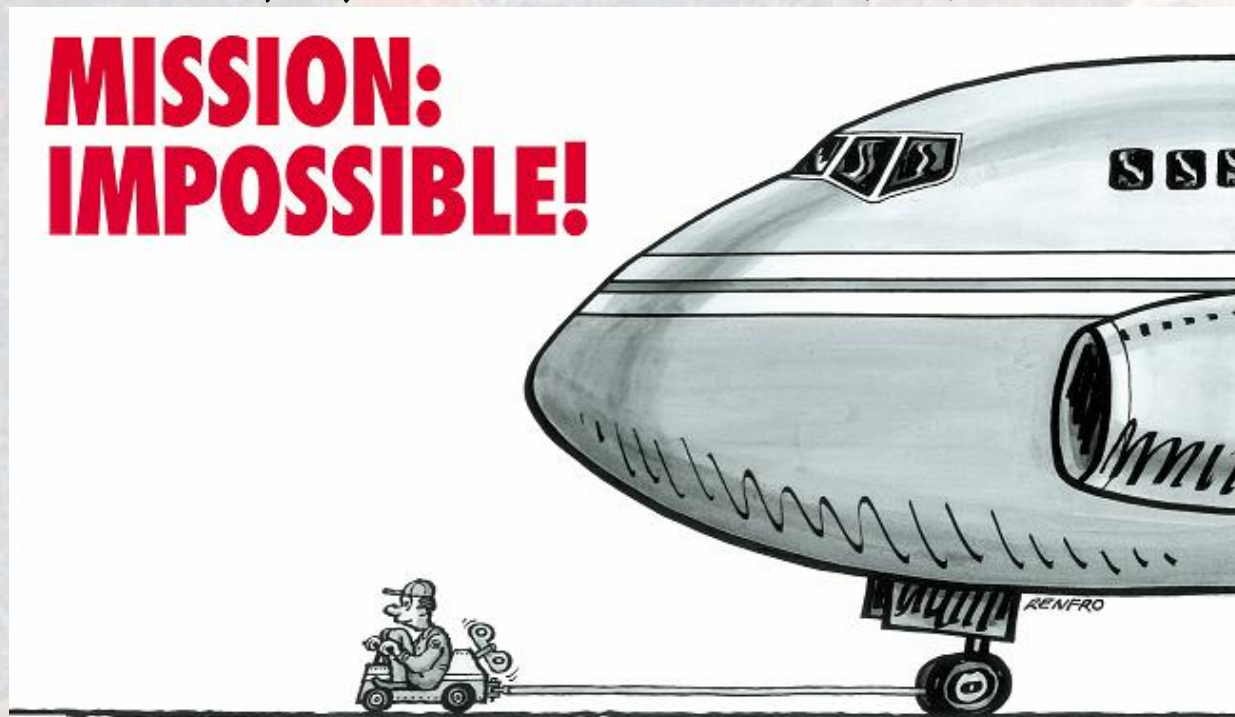
案例背景

- 开发延后，客户却不知情
- 软件系统
 - 基于第三方平台 + 部分外包（报表子系统）
 - 遗留系统（CMS子系统）
 - “从头做起”（办公子系统）
- 团队
 - 成员对新技术不熟悉（通过前期培训和迷你教学项目实战加以弥补）
 - 欠缺良好编码习惯
- 插曲：第一周的加班

案例背景（续）

- 结论：这是个风险颇大的项目
 - 我们真正能控制的只占系统的三分之一
 - 团队过程有待改进，而改进需要时间

**MISSION:
IMPOSSIBLE!**



第一个迭代周期

- 改进策略

- 简化问题，关注重点
- 引入迭代计划和用户故事编写
- 辅助实践：复查程序员当日代码

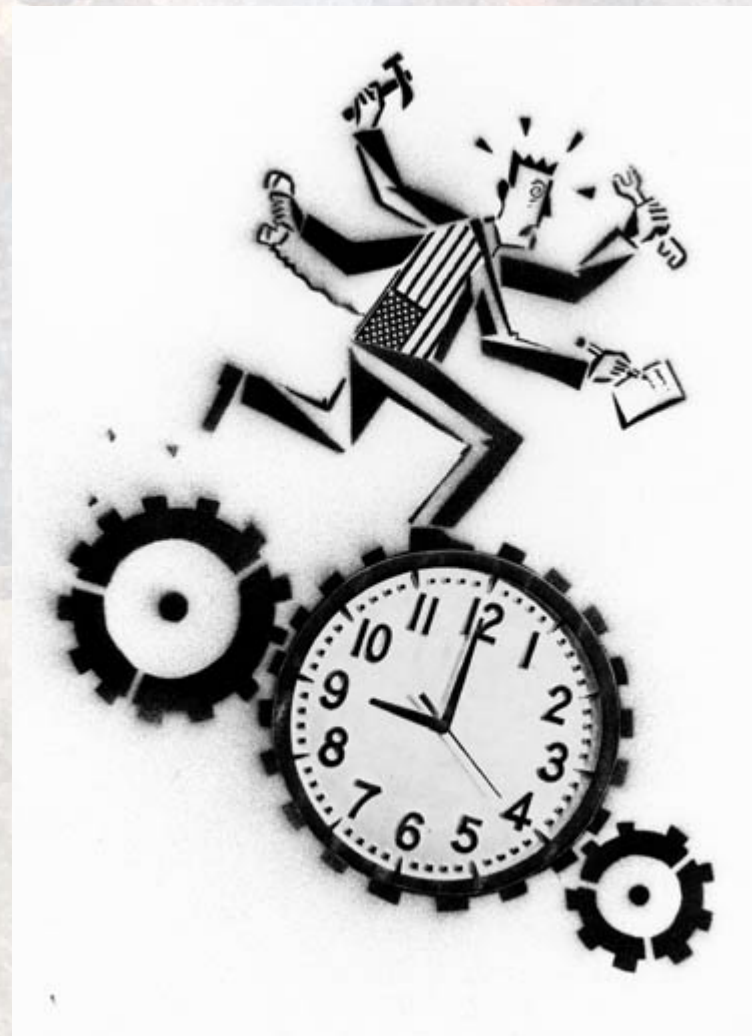
目的：通过持续复查一定程度矫正开发者编码习惯

理由：工作量不大，反馈速度足够快



第一个迭代周期（续）

- 关于加班
 - 加班成为习惯思维
 - 对未知事物的恐惧心理
 - 缺乏明确的反馈机制
 - 解决加班问题的三张牌
 - 时间
 - 人员
 - 范围（超市购物）



第二个迭代周期

- 情况

- 开发速度渐趋稳定，一切按计划进行，工作气氛轻松

- 进一步改进策略

- 将进度（非正式文档）发给决策者
 - 引入“车轮式”结对

为团队成员示范好的实践

鼓励充分沟通并细粒度把握项目风险

充分沟通可以减少孤独感，结对强化沟通

结对的两人中，需要有经验者做主导

结对和私人空间不冲突



第二个迭代周期（续）

- 插曲：形同虚设的Wiki公告
 - 公示项目状况的必要性
 - 电子化手段不一定起作用
 - 改用邮件，或面对面沟通
 - 敏捷团队中为何会有“故事墙”
- 插曲：人员更迭
 - 单人多任务，人员频繁调动
 - 频繁切换工作内容不利于团队整体感、连续性、生产率
 - 被池化的人
 - 虚幻的效率，从未得到证实
 - 忽略团队价值，切换的代价

小结

- 首先采用何种实践，最好根据实际环境，从认为最有机会改进的地方入手，循序渐进。
- 改变团队首先要从自身做起。用事实说服身边的人，说服团队外的人，达到一致认同的价值观。
- 要做好“迂回前进”的心理准备，注意人们面对变化的本能反应。同时请记住：经历痛苦挫折的人往往更加愿意尝试剧烈的变化。
- 接受XP的观点会让人失去掩蔽，不设防才是真正的安全。
- 软件开发不仅是程序员的活动，要平衡所有人的利害关系。同时还要将视野扩大到团队外，漠视等于回避充分的沟通（没有真正领会XP）
- 技术人员对“重构”、“持续集成”情有独钟，建议要有更开阔的视野。时刻注意协调技术观点和业务观点的矛盾。

谢谢