极限编程（XP）

目录

什么是极限编程

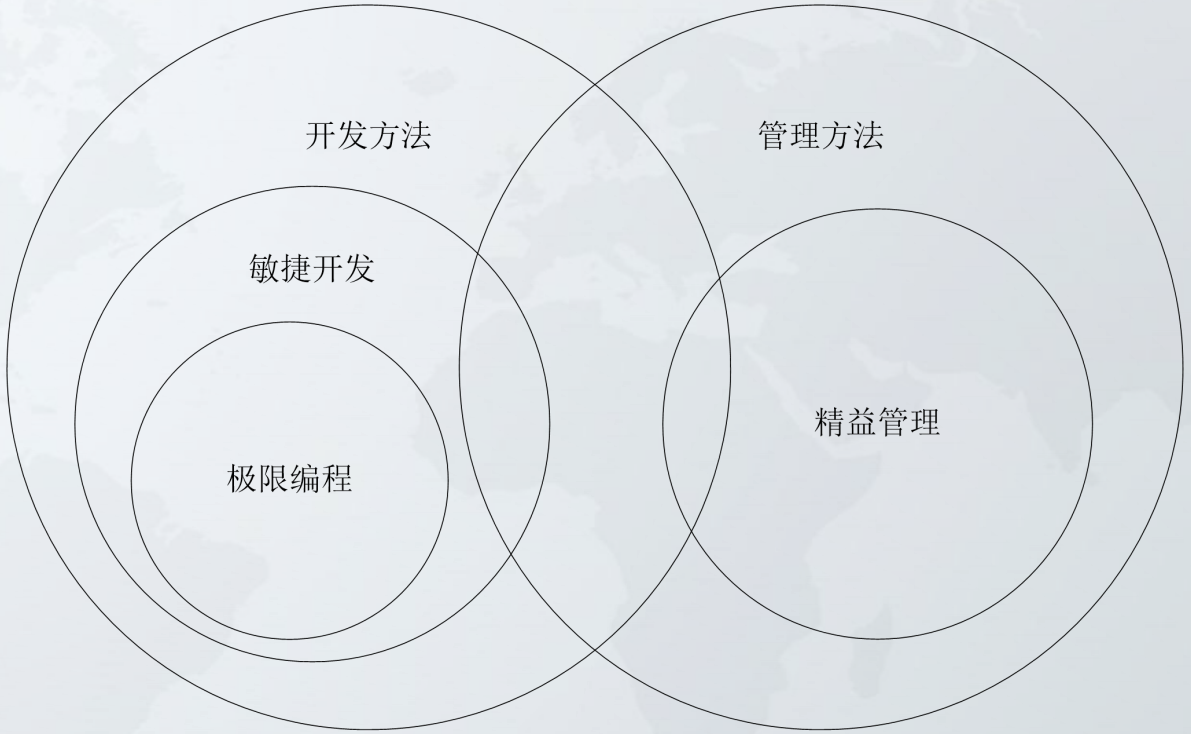
极限编程（eXtreme Programmin，简称xp）

- 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软件开发方法论。

——《解析极限编程》，Kent Beck.

- 其特点是：简单、快速、低缺陷率、适应需求变化。
- 是敏捷开发中最著名的方法。
- 极限编程中的“Extreme”（极限）是指将有效的软件开发原理和实践应用到极限。

极限编程与敏捷开发的关系

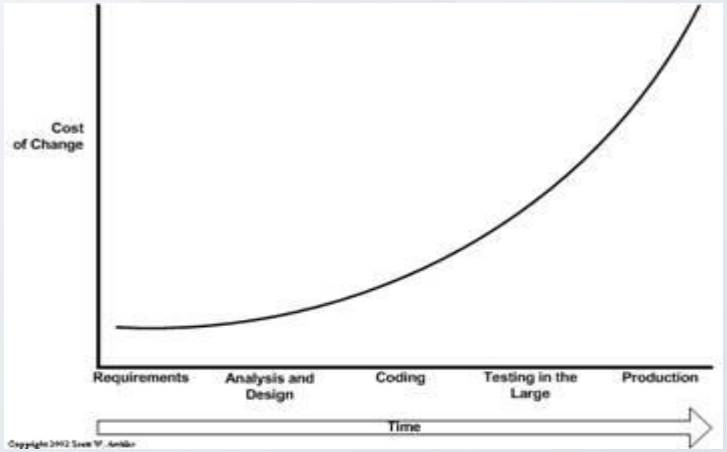


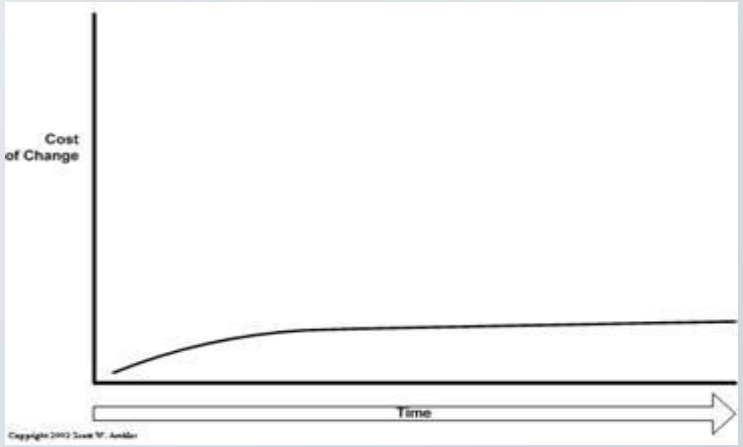
极限编程不研究管理，也没有严格的项目进度管理（就是白板+贴纸），scrum涉及一点项目管理。与敏捷开发相对应的有“精益管理”

XP 的目标

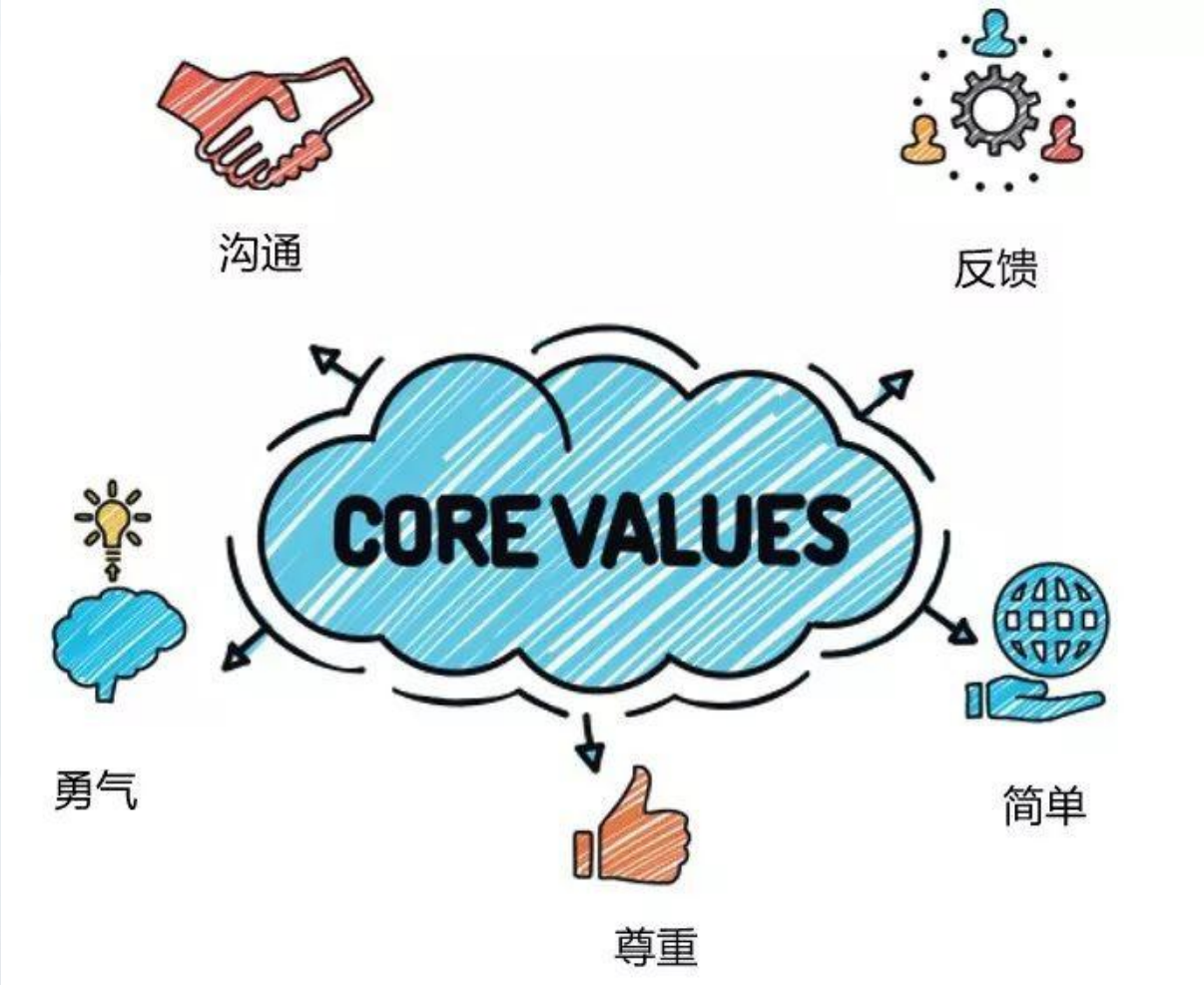
- 极限编程的主要目标在于降低因需求变更而带来的成本。
- 极限编程透过引入基本价值、原则、方法等概念来达到降低变更成本的目的。

传统　　V.S　　XP





极限编程的价值观



极限编程的价值观

- **沟通** — 把沟通看作项目中间协调与合作的主要推动因素。
- **为什么要沟通讨论？**

（1）因为你的问题别人可能已经知道答案，重复研究浪费时间

（2）讨论可以避免盲点

（3）要从多个待选方案中选取较优者，才是正确的工作方法

生活中的例子：三个臭皮匠顶个诸葛亮

项目中的例子：建领域模型（通过沟通将业务领域的知识融合起来）

极限编程的价值观

- **反馈** —— 获取系统反馈：编写单元测试，程序员直观得到系统修改后的状态；

  获得客户的反馈：获取客户对产品完成的反馈意见；

  获得小组的反馈：获取团队成员间协作的反馈意见。

- 即软件开发不是执行预定计划的过程。而是根据反馈不断改进的过程。

生活中的例子：做菜，开车

项目中的例子：频繁测试，结对编程（实时代码评审）

极限编程的价值观

- **简单** — 假定未来不能可靠地预测，所以不应该过多考虑未来的问题
  而是应该集中力量解决燃眉之急。
- 即在保证满足用户需求的前提下，尽量选择最简单的设计。
- 对应于设计原则“简洁为美”。
- 因为简单的系统才好维护，而维护成本是软件开发成本中比例最大的。

生活中的例子：自行车、电动车、汽车的维护成本

项目中的例子：链表、数组、全局变量、局部变量，尽量使用简单的方式

极限编程的价值观

- **尊重** — 团队成员间的互相尊重：保证每个人提交的改变不会导致编制无法通过或者现有测试案例失效或者其他方式导致工作延期。

  团队成员对工作的尊重：坚持追求高质量、坚持通过重构的手段寻找更好的解决方案。

极限编程的价值观

- **勇气** ——因为强调“拥抱变化”，因此对于用户的反馈，要有积极面对现实和修复问题的勇气，如放弃系统的代码，改进系统设计等。
- 勇敢的重构。
- 敢于所有人拥有代码。
- 敢于极限（把好的方法做到极至）

极限编程的优点

- **优点**
  - 拥抱需求变化。
  - 强调团队合作。
  - XP可以让开发者专注于编写代码，避免了不必要的文案工作及会议。
  - 增强代码和产品质量，并有效的减少BUG。
  - 程序员互相帮助，互相教对方，实现能力互补。
  - 从公司管理的角度来看，这种方法可以减少对牛人的依赖。同时它也提升了员工满意度。

极限编程的缺点

- **缺点**
  - 缺乏设计文档，局限于小规模项目
  - 缺乏质量规划
  - 没有提供数据的收集和使用的指导
  - 开发过程不详细
  - 全新的管理手法带来的认同度问题
- **适用对象**
  - 2-12人的小型团队

目录

为什么要使用极限编程

- **XP主要解决**
  - 不能适应需求变化——计划游戏
  - 软件缺陷多——结对编程、测试驱动开发、持续集成
  - 代码质量低——重构、代码规范
  - 设计不良——浮现式设计
  - 项目中浪费大——现场客户、简单设计
  - 开发效率低——以上所有、每周40小时工作制
  - 项目风险大——代码集体所有、小型发布

XP的最佳实践

1. 现场客户（On-site Customer）
2. 计划游戏（Planning Game）
3. 系统隐喻（System Metaphor）
4. 简单设计（Simple Design）
5. 代码集体所有（Collective Code Ownership）
6. 结对编程（Pair Programming）
7. 测试驱动（Test-driven）
8. 小型发布（Small Releases）
9. 重构（Refactoring）
10. 持续集成（Continuous integration）
11. 每周40小时工作制（40-hour Weeks）
12. 代码规范（Coding Standards）

1. 现场客户（On-site Customer）

- **XP**： 要求至少有一名实际的客户代表在整个项目开发周期在现场负责确定需求、回答团队问题以及编写功能验收测试。
- **评述**：现场用户可以一定程度地解决项目团队与客户沟通不畅的问题，但是不能保证有一定技术层次的客户常驻开发现场。解决问题的方法：一是可以采用在客户那里现场开发的方式；二是采用有效的沟通方式。
- **项目**：首先，在项目合同签署前，向客户进行项目开发方法论的介绍，使得客户清楚项目开发的阶段、各个阶段要发布的成果以及需要客户提供的支持等；其次，由项目经理每周向客户汇报项目的进展情况，提供目前发布版本的位置，并提示客户系统相应的反馈与支持。

2. 计划游戏(Planning Game)

- **XP**：要求结合项目进展和技术情况，确定下一阶段要开发与发布的系统范围。
- **评述**：项目的计划在建立起来以后，需要根据项目的进展来进行调整，一成不变的计划是不存在。因此，项目团队需要控制风险、预见变化，从而制定有效、可行的项目计划。
- **项目**：在系统实现前，我们首先按照需求的优先级做迭代周期的划分，将高风险的需求优先实现；同时，项目团队每天早晨参加一个15分钟的项目会议，确定当天以及目前迭代周期中每个成员要完成的任务。

3. 系统隐喻（System Metaphor）

- **XP**：通过隐喻来描述系统如何运作、新的功能以何种方式加入到系统。通常包含了一些可以参照和比较的类和设计模式。
- **评述**：XP在系统实现初期不需要进行详细的架构设计，而是在迭代周期中不断的细化架构。但对于大型系统或者是希望采用新架构的系统，就需要在项目初期进行系统架构设计，并在第一个迭代周期中进行验证，同时在后续迭代周期中逐步进行细化。
- **项目**：开发团队在设计初期，决定参照STRUTS框架，结合项目的情况构建了针对工作流程处理的项目框架。首先，团队决定在第一个迭代周期实现配件申请的工作流程，在实际项目开发中验证了基本的程序框架；而后，又在其它迭代周期中，对框架逐渐精化。

4. 简单设计（Simple Design）

- **XP**：认为代码的设计应该尽可能的简单，只要满足当前功能的要求，不多也不少。
- **评述**：XP认为需求是会经常变化的，因此设计不能一蹴而就，而应该是一项持续进行的过程。应该首先确定一个灵活的系统架构，而后在每个迭代周期的设计阶段可以采用XP的简单设计原则，将设计进行到底。
- **项目**：在项目的系统架构经过验证后的迭代周期内，始终坚持简单设计的原则。对于新的迭代周期中出现需要修改既有设计与代码的情况，首先对原有系统进行”代码重构”，而后再增加新的功能。

5.代码集体所有（Collective Code Ownership）

- **XP**：开发小组的每个成员都有更改代码的权利，所有人对于全部代码负责。
- **评述**：代码全体拥有并不意味者开发人员可以互相推委责任，而是强调所有的人都要负责。如果一个开发人员的代码有错误，其他开发人员也可以进行BUG的修复。但同时需要注意一定要有严格的代码控制管理。
- **项目**：要求开发人员不仅要了解系统的架构、自己的代码，同时也要了解其它开发人员的工作以及代码情况。这个实践与同级评审有一定的互补作用，从而保证人员的变动不会对项目的进度造成很大的影响。

  在项目执行中，一个开发人员缺席项目执行一周，由于实行了”代码集体所有”的实践，其它开发人员分担了该成员的任务，从而保证项目如期交付。

8. 小型发布（Small Releases）

- **XP**：强调在非常短的周期内以递增的方式发布新版本，从而可以很容易地估计每个迭代周期的进度，便于控制工作量和风险，及时处理用户的反馈。
- **评述**：小型发布突出体现了敏捷方法的优点。如果能够保证测试先行、代码重构、持续集成等最佳实践，实现小型发布也不是一件困难的事情。
- **项目**：项目在筹备阶段就配置了一台测试与发布服务器，在项目实施过程中，平均每两周进行一个小型发布；用户在发布后两个工作日内，向项目小组提交”用户接收测试报告”，由项目经理评估测试报告，将有效的BUG提交并分配给相应的开发人员。项目小组在下一个迭代周期结束前修复所有用户提交的问题。

9. 重构（Refactoring）

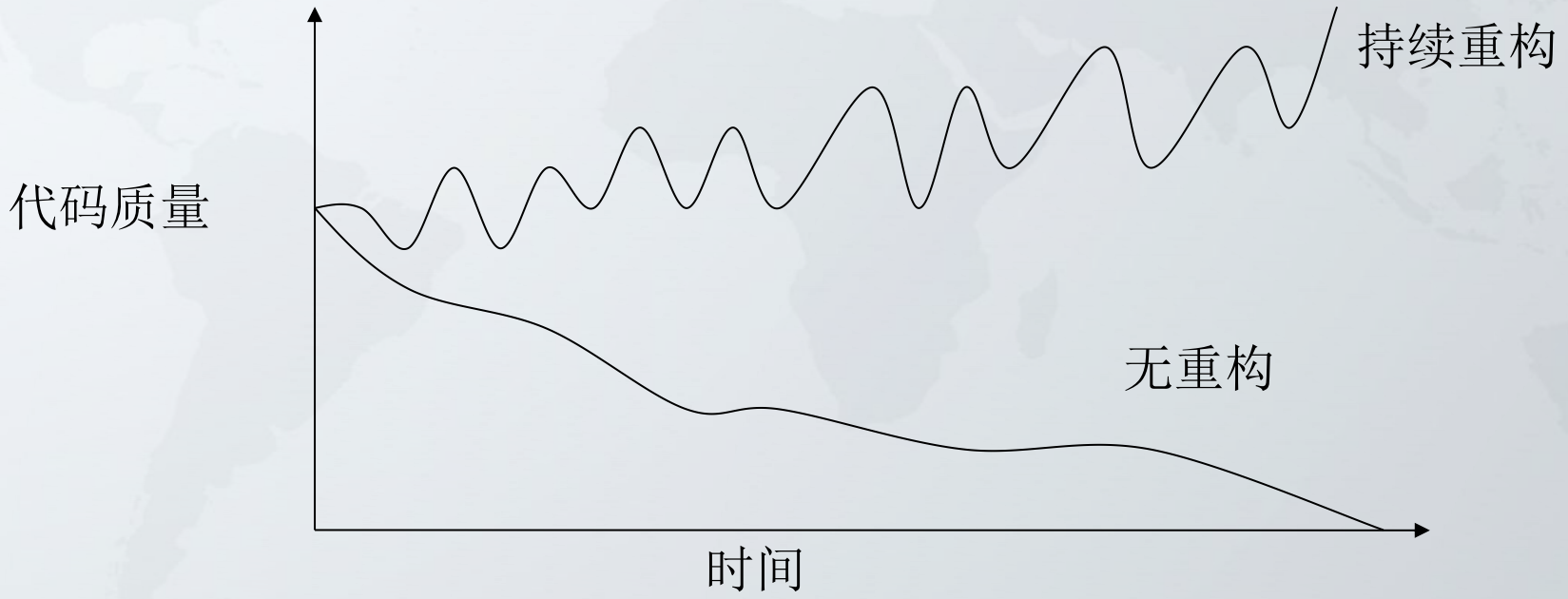
- **XP**：重构是在不改变软件外部行为的前提下改善其内部结构。
- **评述**：本质上说：重构就是在代码写好后改进它的设计。

  是一种整理代码的方法，以提高代码的质量。

为什么要重构

- **重构 vs 不重构**

（代码质量主要有正确性，可读性，可维护性，可扩展性、稳定性等）



如何重构

- **一次重构的步骤**
  - ①发现“代码坏味”（即代码中不符合设计原则的部分）
  - ②使用重构消除代码坏味
  - ③自动测试，保证重构没有改变代码外部行为

  不断重复以上三步，改进代码的设计

- **模块重构的级别**

  函数级整理、对象级整理、业务逻辑的整理

代码坏味有哪些？

- 不可读（13项）

  词不达意——函数改名　超长函数——抽取函数

  过长参数列——用函数取代参数　抽象层级不一致——抽取函数

- 不简洁（8项）

  重复代码——合并代码　重复逻辑——合并逻辑　过度设计——移除间隔层

- 不面向对象（7项）

  过程代码——生成对象

- 不单一职责（4项）

  发散式变化——切分代码　代码耦合——添加间隔层 or 依赖注入

  散弹式修改——重组代码　层间耦合——领域层与界面层分离 or 软件与硬件分离

重构实例

为什么要“增加函数减少参数”？具体如何操作？

- ①提高代码可读性
- ②减少代码间的耦合
- ③降低代码复杂度，易于维护

Set_type(A/B)

->set_type_A() 与set_type_B()

重构实例

什么是抽象层级不一致？

- 底层——抽象层级低，细节多，粒度小
- 高层——抽象层级高，细节少，粒度大

抽象层级的基本划分：分层、分模块、分对象、分函数（但注意对象之间、函数之间同样有抽象层级的关系）

生活中的例子：语言也是一种抽象；“早餐、中餐、晚餐、鸡蛋”、“ 外壳 引擎 底盘 螺丝钉“哪一个词抽象层级不符？；

24楼与1楼看到的景色有什么不一样？

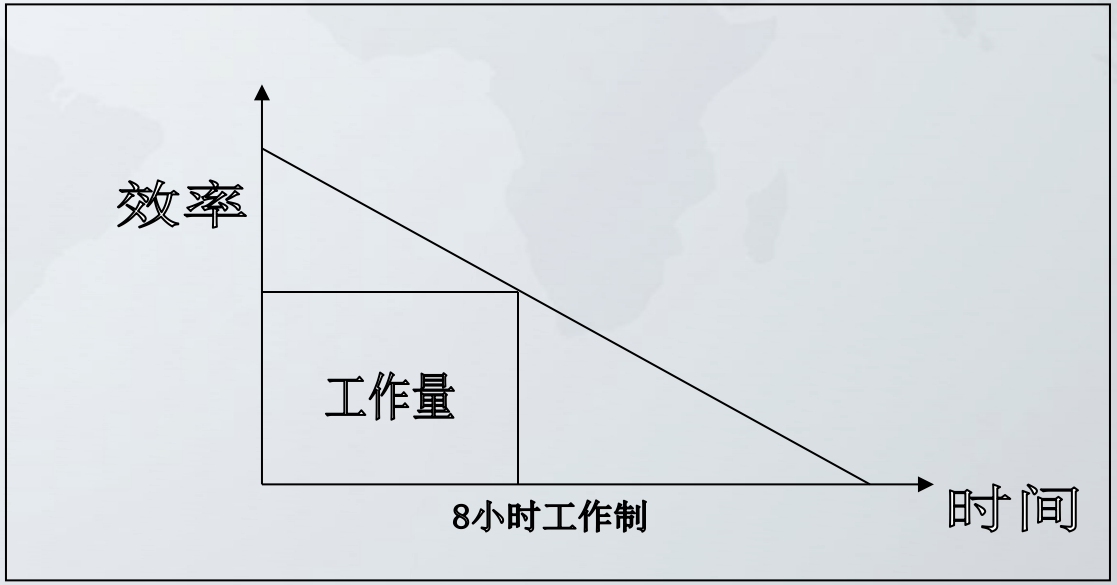
重构

什么时候重构？

- 任何修改代码的时候都需要重构。如添加新功能之前、修改bug之后都需要重构。
- 也可以抽专门的时间来重构。

11. 每周40小时工作制（40-hour Weeks）

- **XP**：要求项目团队人员每周工作时间不能超过40小时，加班不得连续超过两周，否则反而会影响生产率。
- **评述**：不加班，不熬夜，对于项目进度和工作量合理安排的要求比较高。



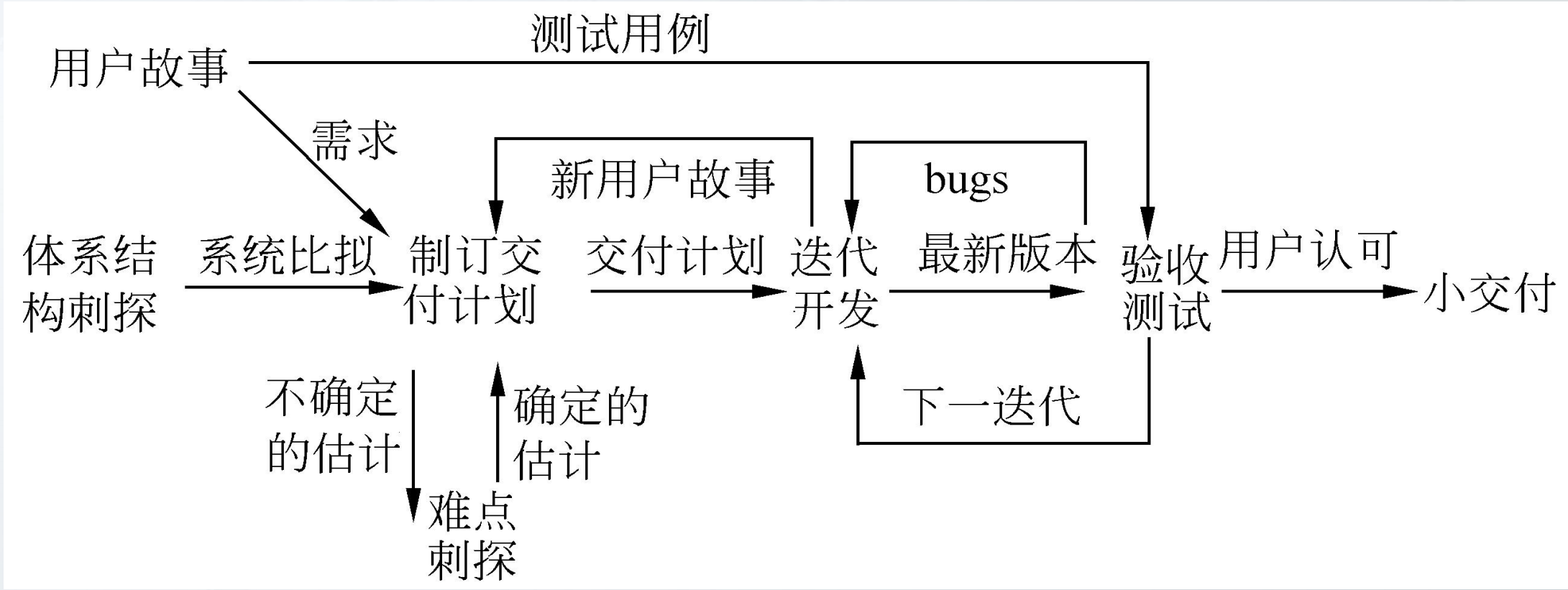
问题及XP解决方案

通常，项目只有在遇到麻烦时才会寻求极限编程等新方法的帮助

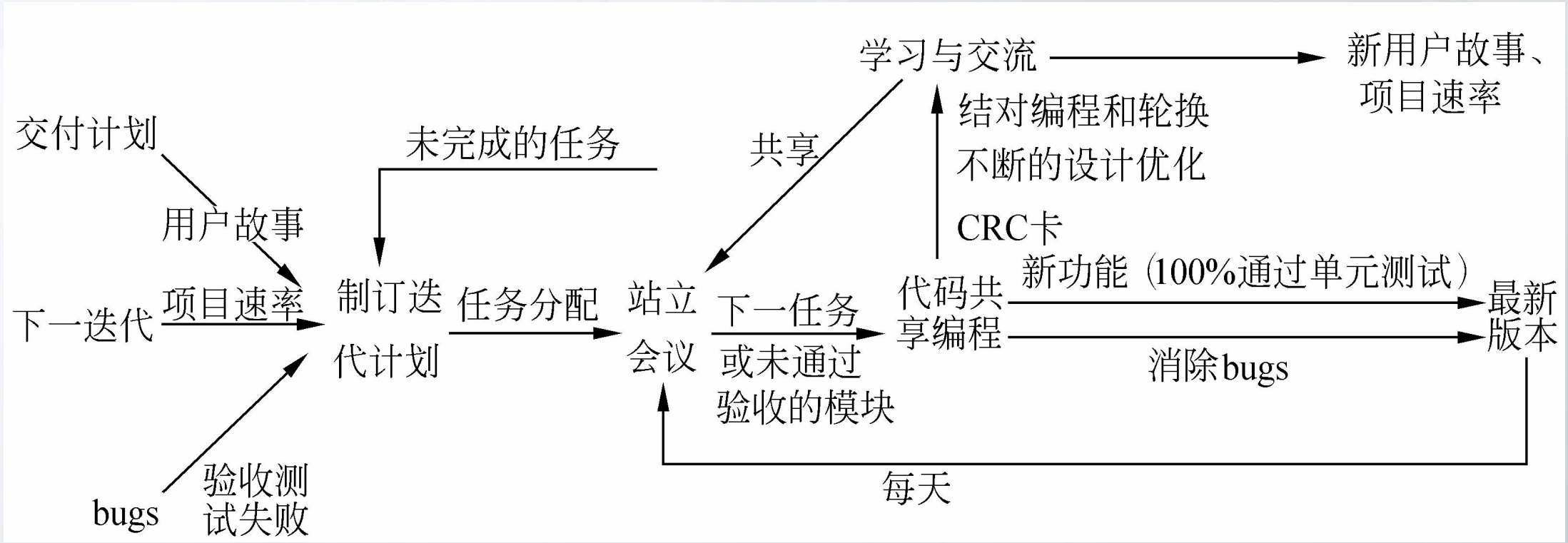
- 如果发现需求规范有25%是完全无用的，那么可以与客户一起编写用户故事。
- 如果经常遇到需求变更的问题，导致频繁地重新创建计划，那么尝试每隔几次迭代就召开一次简单发布计划会议，及时进行编程任务的规划。
- 如果遇到最大的问题是生产中的bug数量，那么尝试进行自动化验收测试，进行回归和验证测试。
- 如果最大的问题是集成bug，那么尝试自动化的单元测试。在将任何新代码发布到代码存储库之前，要求所有单元测试都通过（达到100%）。
- 如果一两个开发人员因为拥有系统中的核心类而成为瓶颈，必须进行所有的更改，那么尝试集体代码所有权。 让每个人都可以在需要时对核心类进行修改。

目录

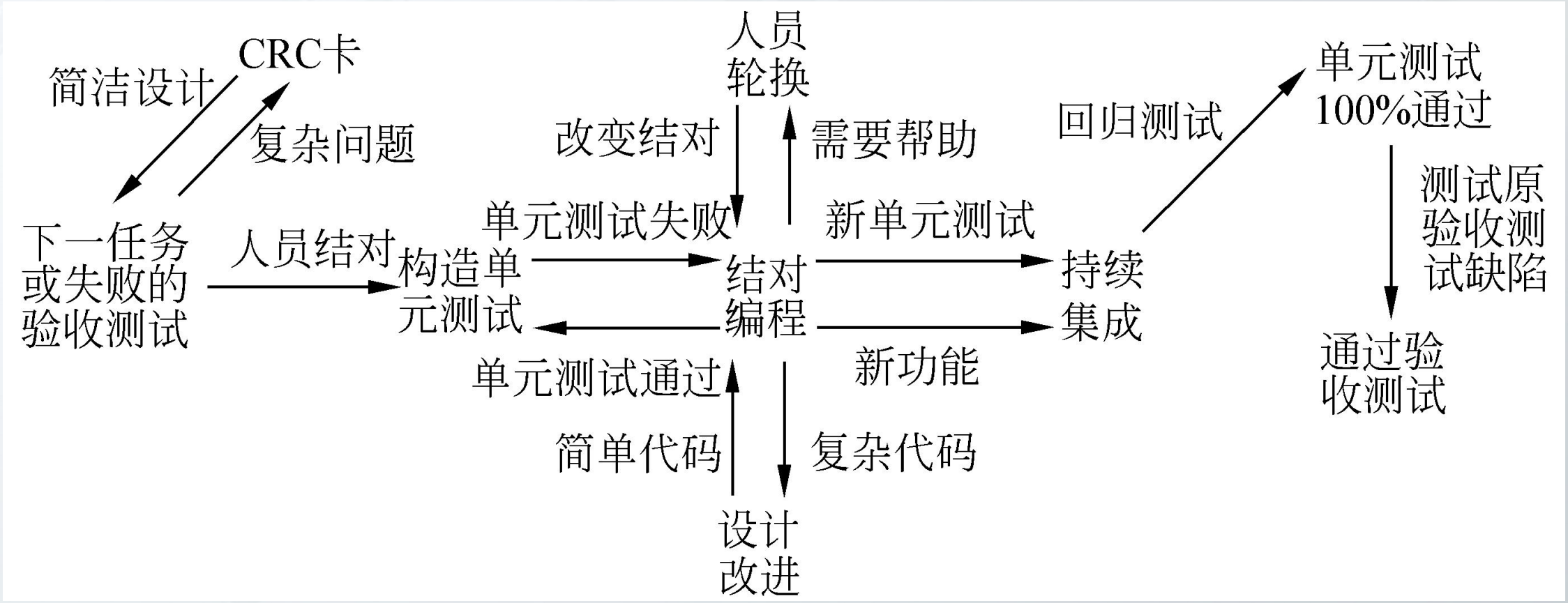
XP项目开发过程


测试用例
用户故事
需求
新用户故事
bugs
体系结构刺探
系统比拟
制订交付计划
交付计划
迭代开发
最新版本
验收测试
用户认可
小交付
不确定的估计
确定的估计
下一迭代
难点刺探

XP迭代开发过程


学习与交流
新用户故事、
项目速率
结对编程和轮换
不断的设计优化
CRC卡
交付计划
未完成的任务
共享
用户故事
项目速率
下一迭代
制订迭
代计划
任务分配
站立
会议
下一任务
或未通过
验收的模块
代码共
享编程
新功能（100%通过单元测试）
最新
版本
消除bugs
bugs
验收测
试失败
每天

XP的代码共享编程


CRC卡
简洁设计
复杂问题
下一任务
或失败的
验收测试
人员结对
构造单
元测试
人员
轮换
改变结对
需要帮助
单元测试失败
结对
编程
新单元测试
持续
集成
回归测试
单元测试
100%通过
测试原
验收测
试缺陷
通过验
收测试
单元测试通过
新功能
简单代码
复杂代码
设计
改进

XP与Scrum的异同

相同点

- 都属于敏捷开发
- 都追求短周期交付，持续交付
- 都注重交付产品的质量
- 都注重反馈
- 都使用迭代的方式进行开发
- 都有固定的迭代周期
- 都使用用户故事描述
- 都积极响应客户变化

XP与Scrum的不同点

类别	Scrum	XP
范围	不止应用于软件开发领域	主要解决软件开发中的问题
应用	是一套理论框架，没有具体实践	也是框架，但规划了大量实践
迭代周期	1-4周	周期相对更短
用户故事	PO划分优先级	客户划分优先级
迭代内容	团队成员决定做哪些故事	优先做高优先级高价值的故事
测试力度	注重质量，但没有规定测试的实践	测试与开发同等重要，强调自动化测试
团队	完全的自组织管理，团队自由发挥	应用一系列实践
实施	有明确的角色和事件（各种会议）	没有明确会议，更关注开发人员的实践

在管理模式上启用Scrum， 而在实践中，创造一个适合自己项目组的XP

总结

1. XP的价值观是什么？
2. XP不编写文档吗？团队成员之间如何沟通？
3. 什么是重构？
4. 极限编程的适用情景。
5. 极限编程的开发过程。

THANKS