摘要

关键字

Abstract

Key word

1. 绪论
   1. 写作背景
   2. 传统软件开发面临的问题
2. 敏捷软件开发概述

简单的说，敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。

* 1. 敏捷开发的产生

2001年，在美国犹他州雪鸟滑雪圣地的一次敏捷方法发起者和实践者的聚会中，17名编程大师分别代表极限编程（eXtreme Programming ，XP）、Scrum、动态系统开发方法（Dynamic Systems Development Method，DSDM）、自适应软件开发（Adaptive Software Development，ASD）、Crystal方法、特性驱动开发（Feature-Driven Development，FDD）、实用程序设计等开发流派，发表“敏捷软件开发”宣言。敏捷软件开发是一个开发软件的管理新模式，用来替代以文件驱动开发的瀑布开发模式。

* 1. 价值观

敏捷建模（Agile Modeling，AM）的价值观包括了XP的四个价值观：[沟通](http://baike.baidu.com/view/54445.htm)、[简单](http://baike.baidu.com/view/66543.htm)、[反馈](http://baike.baidu.com/view/80120.htm)、[勇气](http://baike.baidu.com/view/53936.htm)，此外，还扩展了第五个价值观：谦逊。

敏捷开发是针对传统的[瀑布开发模式](http://baike.baidu.com/view/4169795.htm)的弊端而产生的一种新的开发模式，目标是提高开发效率和响应能力。除了原则和实践，模式也是很重要的，多研究模式及其应用可以使你更深层次的理解敏捷开发。

* + 1. 沟通

建模不但能够促进你团队内部的开发人员之间沟通、还能够促进你的团队和你的project stakeholder之间的沟通。

* + 1. 简单

画一两张图表来代替几十甚至几百行的[代码](http://baike.baidu.com/view/41.htm)，通过这种方法，建模成为简化软件和软件（开发）过程的关键。这一点对开发人员而言非常重要－它简单，容易发现出新的想法，随着你（对[软件](http://baike.baidu.com/view/37.htm)）的理解的加深，也能够很容易的改进。

* + 1. 反馈

Kent Beck在Extreme Programming Explained中有句话讲得非常好：“乐观是编程的职业病，反馈则是其处方。”通过图表来交流你的想法，你可以快速获得反馈，并能够按照建议行事。

* + 1. 勇气

勇气非常重要，当你的决策证明是不合适的时候，你就需要做出重大的决策，放弃或[重构](http://baike.baidu.com/view/120901.htm)（refactor）你的工作，修正你的方向。

* + 1. 谦逊

最优秀的开发人员都拥有谦逊的美德，他们总能认识到自己并不是无所不知的。事实上，无论是开发人员还是客户，甚至所有的 project stakeholder，都有他们自己的专业领域，都能够为项目做出贡献。一个有效的做法是假设参与项目的每一个人都有相同的价值，都应该被尊重。

* 1. 核心原则
     1. 主张简单

当从事开发工作时，你应当主张最简单的解决方案就是最好的解决方案。不要过分构建（overbuild）你的软件。用AM的说法就是，如果你现在并不需要这项额外功能，那就不要在模型中增加它。要有这样的勇气：你现在不必要对这个系统进行过分的[建模](http://baike.baidu.com/view/44500.htm)（over-model），只要基于现有的需求进行建模，日后需求有变更时，再来重构这个系统。尽可能的保持模型的简单。

* + 1. 拥抱变化

需求时刻在变，人们对于需求的理解也时刻在变。项目进行中，Project stakeholder可能变化，会有新人加入，也会有旧人离开。Project stakeholder的观点也可能变化，你努力的目标和成功标准也有可能发生变化。这就意味着随着项目的进行，项目环境也在不停的变化，因此你的开发方法必须要能够反映这种现实。

* + 1. 递增的变化

和建模相关的一个重要概念是你不用在一开始就准备好一切。实际上，你就算想这么做也不太可能。而且，你不用在模型中包容所有的细节，你只要足够的细节就够了。没有必要试图在一开始就建立一个囊括一切的模型，你只要开发一个小的模型，或是概要模型，打下一个基础，然后慢慢的改进模型，或是在不在需要的时候丢弃这个模型。这就是递增的思想。

* + 1. 高质量的工作

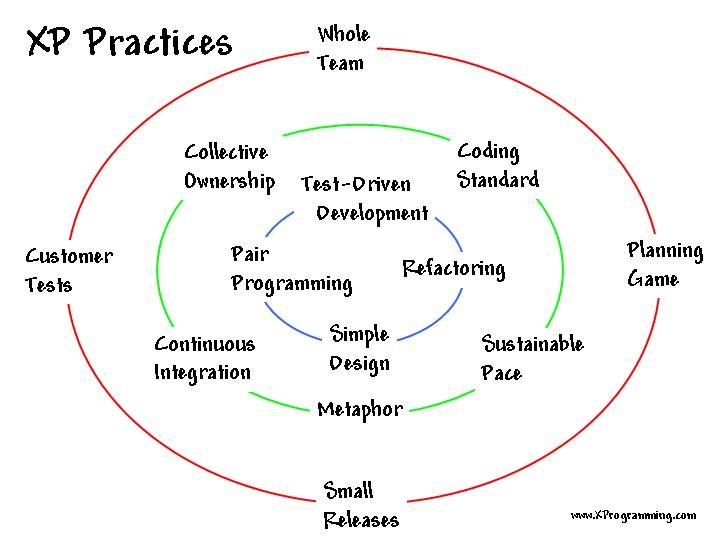
没有人喜欢烂糟糟的工作。做这项工作的人不喜欢，是因为没有成就感；日后负责重构这项工作（因为某些原因）的人不喜欢，是因为它难以理解，难以更新；最终用户不喜欢，是因为它太脆弱，容易出错，也不符合他们的期望。

* + 1. 快速反馈

从开始采取行动，到获得行动的反馈，二者之间的时间至关紧要。和其他人一共开发模型，你的想法可以立刻获得反馈，特别是你的工作采用了共享建模技术的时候，例如白板、CRC卡片或即时贴之类的基本建模材料。和你的客户紧密工作，去了解他们的的需求，去分析这些需求，或是去开发满足他们需求的用户界面，这样，你就提供了快速反馈的机会。

* 1. 优秀的实践

极限编程（eXtreme Programming，简称XP）是一种轻量级、高效、低风险、柔性、可预测的、科学的软件开发方法，其特性包含在12个最佳实践中。



* + 1. 计划游戏 （ Planning Game ）

    （1）快速制定计划、随着细节的不断变化而完善；

    （2）详解：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。当计划赶不上实际变化时就应更新计划。

* + 1. 小型发布 （ Small Release ）

    （1）系统的设计要能够尽可能早地交付；

    （2）详解：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；同时，也可以及时处理用户的反馈。

* + 1. 系统隐喻（ System Metaphor ）

    （1）找到合适的比喻传达信息；

    （2）详解：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。它通常包含了一些可以参照和比较的类和设计模式。

* + 1. 简单设计（ Simple Design ）

    （1）只处理当前的需求使设计保持简单；

    （2）详解：任何时候都应当将系统设计的尽可能简单。不必要的复杂性一旦被发现就马上去掉。

* + 1. 测试驱动（ Test-driven ）

    （1）先写测试代码再编写程序；

    （2）详解：程序员不断地编写单元测试，在这些测试能够准确无误地运行的情况下开发才可以继续。

* + 1. 重构（ Refactoring ）

     （1）重新审视需求和设计，重新明确地描述它们，以符合新的和现有的需求；

     （2）详解：代码重构是指在不改变系统行为的前提下，重新调整、优化系统的内部结构以减少复杂性、消除冗余、增加灵活性和提高性能。

* + 1. 结对编程（ Pair Programming ）

（1）由两个程序员在同一台电脑上共同编写解决同一问题的代码。

（2）详解：通常一个人负责写编码，而另一个负责保证代码的正确性与可读性。

* + 1. 集体所有权（Collective Ownership）

    （1）任何人在任何时候都可以在系统中的任何位置更改任何代码。

    （2）详解：每个成员都有更改代码的权利，所有的人对于全部代码负责。

* + 1. 持续集成（ Continuous Integration ）

    （1）可以按日甚至按小时为客户提供可运行的版本；

    （2）提倡在一天中集成系统多次，而且随着需求的改变，要不断的进行回归测试，避免了一次系统集成的恶梦。

* + 1. 每周工作40小时 （ 40-hour Week ）

    （1）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。

* + 1. 现场客户（ On-site Customer ）

    （1）在团队中加入一位真正的、起作用的用户，他将全职负责回答问题。

     （2）详解：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。

* + 1. 编码标准（ Code Standards ）

1. 强调通过指定严格的代码规范来进行沟通，尽可能减少不必要的文档。
   1. 敏捷开发的中国之旅

敏捷软件运动在中国经历了三个阶段。2002年前后为第一阶段，这一时期国际上发布了《敏捷宣言》，国内最早的一小批探路者开始关注敏捷，标志性事件是翻译了一套七本的《极限编程》。到了2007年，以ThoughtWorks发起敏捷中国大会（Agile China）为标志进入第二阶段，开始系统导入国际主流的思想和方法，实践应用集中在一些新兴的互联网企业。

2011年，以中国敏捷软件开发联盟（CAA）成立为标志，敏捷进入全面落地的第三个阶段，诸如中国移动、华为、用友等国内代表性大企业都相继开启了敏捷转型之路。

敏捷运动一波三折，但终究形成势不可挡的局面，十年敏捷运动史，动力何在？ 有IT业竞争加剧和客户面临的市场压力，有程序员对自身价值体现的永恒追求，有对软件危机的不断追问和应对危机道路上一次次挫败的失落，但还有一个重大因素闪现其间，构成驱动敏捷运动蓬勃发展的关键因素，这就是互联网！

多数人都相信互联网的时代才刚刚开始，更大的精彩还在后面。当今IT业的三大趋势——云计算、物联网和移动互联网，无不是互联网概念的延续和升级。这些IT技术、商业模式变革及由此汇集而成的产业重大趋势，才是业界对软件研发不断提出拷问的最重要幕后背景和推动软件研发新方法产生的不竭动力。事实上，我们正无奈地面临两面夹击的尴尬处境:传统软件危机暴露出的问题还未真正解决，新的挑战却已层层堆叠在眼前。面临的工作量很大，挑战也很大。

谈到在IT业三大趋势下，软件研发面临的挑战及对策，人们的思考才刚刚开始。（1）软件质量以服务质量形式展现，对质量的投资可获得更高的投资回报。（2）软件过程扩展到用户端，用户深入参与到软件全生命周期。（3）云时代适于敏捷交付。（4）功能至上远远不够，用户体验得到空前重视。（5）系统集成模式面临变革，软件、服务、终端、基础IT设施将形成更紧密的价值体系。（6）研发要更多关注非功能性需求，如安全性质量、性能、可靠性、可扩充性、可伸缩性、可用性等。

因而，敏捷方法无疑是应对三大趋势挑战的方法之一。可以预见，从2011年起的这一波新的敏捷运动热潮，还会持续升温。敏捷方法将从敏捷大师的高谈阔论落地到企业实践中，从以互联网企业应用为主延伸到传统软件企业应用，从小企业和小团队应用到大团队和大企业整体敏捷转型。

1. 敏捷开发流程及相关技术介绍
   1. 前期准备
      1. 集中办公

怎样才能极大地提高团队的生产效率呢？答案是让每个人都坐在一起。

集中办公的团队效率就是要高一些。问题不仅可以很快地在现场得到解决，而且彼此间的交流也会更加顺畅，并能很快建立起信任。集中办公的小型团队竞争力是非常强的。那么，既然集中办公的团队这么好，是否意味着分布式团队就无法运作敏捷项目了呢？绝对不是。

对于很多人来说，分布式办公正在成为一种生活方式。虽然相比紧凑型集中办公的团队，分布式团队总会有些劣势，但仍可以用很多办法来加以弥补。比如，在项目的初始阶段，为了将所有人都召集在一起，你要预留一些时间。哪怕是只有几天也好（如果能延长到几周就更好了），大家在这段时间内互相认识一下、开个玩笑、一起吃个饭，这非常有助于你把形形色色的人聚合在一起，打造成一个紧凑高效的团队。因此，在开始阶段，就要将所有人都聚在一起。之后，即使没有集中办公，你也可以使用书本中提及的所有沟通工具和技巧（Skype、视频会议和社交工具），使你的分布式团队看起来像一个集中办公的团队。

* + 1. 专职客户

当今，仍有很多团队编写的软件没有专职客户。这很可悲，甚至可以说是一种犯罪。如果产品的使用者没有纳入进程中，怎么可能期望团队能够建设优秀而具有创造性的产品呢？

专职客户会在演示版中现身、解答问题、给出反馈，并为团队建设优秀软件提供必要指导和洞见。他们是团队的核心成员，是交付过程中的全面合作伙伴。

正是由于这个原因，诸如极限编程和Scrum这样的敏捷方法都尽量实现客户的专职化，比如采用现场客户，或像Scrum那样设立专门的产品负责人。这是一项很重要的工作。稍后我们会对这些角色加以详述。也正因为这样，任何成功的敏捷项目都需要专职客户。

* + 1. 自组织

敏捷团队常常会设定一个目标，然后退而结网，通过共同研究来实现目标。为此，敏捷团队需要进行自组织。

自组织首先要求个人放下架子，与团队协作，将自己作为团队的一分子，想出如何发挥出个人所有的独特技巧、激情和天赋，最好地交付项目的办法。这并不意味着开发者必须要精通可视化设计,或者立刻就要求测试者去管理项目。这只是说建设团队的最好方法是要让角色适应人，而不是让人适应角色。

那么，该如何使团队自组织呢？

* 要让他们自己创建计划、提出估算，并对项目全权负责。
* 不要担心所谓的头衔和角色，而要更关注于不断交付出可运行且经过测试的软件。
* 你要找的人应该具有主动性，喜欢掌控自己命运而非惟命是从。

简言之，你要摆脱束缚，信任他们并授以权力，这样工作才能完成。

* + 1. 勇于承担和授权

优秀的敏捷团队会对其所产生的后果勇于承担。他们知道自己是客户成功的关键，并且从一开始，他们就要交付有价值的东西，这是他们义不容辞的责任。

当然，只有团队真正地被授予权力，勇于承担才会发挥效力。让团队自己去抉择，去做他们认为正确的事情，这可以激发他们的主动性并使其独立工作。他们会解决自己的问题，不必再等待他人的许可。当然，你偶尔也会犯错。但由于这样做优点颇多，故而值得冒险。

* + 1. 跨职能

跨职能团队要能够为客户提供全面的服务。这意味着团队要具备必要的技巧和技能，才能完成客户需要的所有特性，成功地完全交付。

当为团队招募员工时，你要招聘那些多面手，因为他们能够轻松地完成多种任务。如果想招聘程序员，就要找到那些精通整套技术栈之人（不只限于前端技术或者后端技术）。而对于测试人员和分析师来说，他们在做深度需求分析工作时，应与做测试一样轻松自如。

团队缺乏某种专业技巧（比如数据库性能调优）时才会偶尔用到专家。但通常在项目的整个过程中，团队会作为一个整体互相支持，同舟共济。

当然，跨职能团队的真正妙处在于其运转的速度。不必再等什么许可，也不必与其他人讨价还价争取某种资源，而是从一开始他们自己就可以交付有价值的东西，没有人会去挡他们的路。

* 1. 团队中的角色划分

在ThoughtWorks一个典型的敏捷[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)中，大致有四种不同角色：[项目经理](http://www.leadge.com/knowledge/klmore.asp?type=198)、业务分析师、开发工程师、测试工程师。同时，根据项目不同可能还需要：迭代经理，美术设计师、数据库工程师、系统工程师、交互设计师等不同人员。虽然在项目中不同的人需要确定一个角色，并担负相应的责任，但在ThoughtWorks内部，人与人之间是完全平等没有级别区分的。公司这种平等的文化，使得人与人之间的交流不会因为等级差距而丧失。同时，公司鼓励每个人向其感兴趣的其他领域发展，成为综合性人才。例如某个人现在是开发人员，但他也可以通过帮助[项目经理](http://www.leadge.com/knowledge/klmore.asp?type=198)做一些辅助工作，来学习[项目管理](http://www.leadge.com/)方法，从而最终成为独当一面的[项目经理](http://www.leadge.com/knowledge/klmore.asp?type=198)。

* + 1. Project Manager

作为[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)的精神支柱存在。与[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)的每个人进行必要的[沟通](http://www.leadge.com/knowledge/klmore.asp?type=2170218&art_type=%B9%B5%CD%A8%CC%B8%C5%D0)以保障项目成员的士气和稳定性。维持开发秩序，保障[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)间交流的效率和效果，负责主持必要的活动。消除外部干扰，负责与客户进行协调和协作。管理来自与客户的scope变更。跟踪[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)的开发效率，维持开发速率，进行适当调整以保证开发的顺利进行。管理项目[风险](http://www.leadge.com/knowledge/klmore.asp?type=2170223)，维护项目[风险](http://www.leadge.com/knowledge/klmore.asp?type=2170223)日志，识别[风险](http://www.leadge.com/knowledge/klmore.asp?type=2170223)并采取措施防治[风险](http://www.leadge.com/knowledge/klmore.asp?type=2170223) ，负责最终的项目交付成功。

* + 1. Business Analyst

需求获取与管理，与客户持续交流获取新的需求，并保持良好的客户关系。管理需求的优先级。保障下一个迭代需要开发的需求能够预备到位。提前准备好需要的Story卡片，在Iteration Kickoff会议解释每个Story的具体需求给Developer。主持必要的会议，例如Iteration Kickoff和需求的评估活动。对需求进行初步的功能验收，保证功能的交付符合原始需求。

* + 1. Developer/Architect

了解系统业务和需求，设计和演进系统整体架构，能够做出适当的技术决策。编码，并对系统的每行代码负责，保持代码的干净，保持较高的测试覆盖率。维护项目基础设施如持续集成服务器、版本控制服务器等。评估需求，并在开发完成后演示开发的需求。

* + 1. Quality Assurance

负责了解需求并编写需求验收条件，负责制定测试计划。负责测试开发人员完成的需求，并报告错误。负责对[软件](http://www.leadge.com/software/index.asp)进行性能、压力、容量、负载测试等，负责项目的手工功能测试和发布测试。

* + 1. Iteration Manager

小[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)多由[项目经理](http://www.leadge.com/knowledge/klmore.asp?type=198)或分析师兼任，负责项目过程的顺利进行，协调项目资源。主持各种迭代会议，如Standup和Retrospective。负责跟踪需求的状态，负责项目的其他日常事务。

* + 1. User Interaction Designer

多和分析师为同一人。在项目初期参与前期需求的收集，提出可行的交互设计方案，保证[软件](http://www.leadge.com/software/index.asp)的可用性。负责指导项目的界面开发原则，进行用户测试，持续改进系统的可用性。

* + 1. Database Administrator

维护[软件](http://www.leadge.com/software/index.asp)所需的数据库，定期进行数据备份。了解数据库重构和演化方法，负责维持数据库的每一条更新脚本

* + 1. System Engineer/Webmaster

维护[软件](http://www.leadge.com/software/index.asp)所需的各种硬件和网络系统。了解敏捷开发中的发布过程，保证每次迭代发布的实施 。

* + 1. Art Designer

一般[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)最缺少优秀的Art Designer，了解敏捷的Art Designer更甚。了解项目的远景和规划，了解迭代方法，应用增量式美术设计方法。了解[软件](http://www.leadge.com/software/index.asp)的交互设计，能够设计出可用性良好的系统。有一类角色，在敏捷[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)中至关重要，不得不重视起来的，就是Customer。

* + 1. Customer

理解迭代开发的过程，与[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)进行频繁和和谐的交流，参与[团队](http://www.leadge.com/knowledge/more.asp?type=2170219)的各种必要的活动如Showcase。理解需求和排序需求的优先级，验收需求，并及时的提出反馈。为了项目的成功，尽可能多的做一些事情，对个人和对项目都是有利的。

* 1. 敏捷团队成员
     1. 多面手

由于敏捷要求人们能够自始至终地追随并抓住机遇，所以敏捷项目中多面手才会有用武之地。对于程序员来说，这意味着可以学贯整套技术栈（从前端到后端）。对于分析师和测试人员来说，这意味着作分析和测试的时候很惬意。

多面手也会欣然接受不同的角色。他们也许某一天做编码，然后又做分析，继而又做测试。

* + 1. 能够欣然接受模糊角色之人

不是所有东西在敏捷项目中都是准备妥当的。需求并不是都在那里等着——你需要去发现它们。计划改变时，你也要去适应并随着而改变。

要寻找那些泰山崩于前而不动色的人，他们能够承受冲击，即使列车出轨，也能从容应对。

* + 1. 能够放下架子的人

这听起来有些老生常谈，但确定是员工要有大局观并能放下自己的架子，敏捷才能发挥其最大的功效。

并非所有人都喜欢敏捷引发的角色模糊。有些人会保护自认为属于“他们的”地盘。

要寻求那些能够安然面对自己处境之人，他们不畏分享，发自内心地喜欢与他人共同学习和成长。

* 1. 明晰需要取舍的因素（质量、时间、范围、预算）

所有的项目从一开始就被四种交织在一起、互相联系的因素所约束和监督。它们分别是：时间、预算、质量、范围。我们要尊重项目中的某些规则和因素。预算和时间通常都固定。范围经常被扩大，但人们又会将其随便放弃。质量永远是第一位的。

* + 1. 时间

时间是有限的。我们无法对其进行创造或者储存。我们必须在规定的时间内尽最大的努力。这就是为什么敏捷项目总是将其交付限定在一定时间内。不断地推迟交付项目和延迟交付有价值的软件会降低客户的投资回报率，并且还会让自己面临永远也发布不了任何东西的危险，而这对于任何软件项目来说都是灭顶之灾。

* + 1. 预算

预算与时间是一对双胞胎。预算也是固定的、有限的，通常到后来都不会很充裕。

对客户来说，最困难的事莫过于再去找项目发起人要钱了。这种情况偶尔会发生，但经历一定不会愉快。为了避免此类不愉快事件的发生，敏捷项目会给时间和预算同等待遇：把两者都限定。

* + 1. 质量

有些人认为可以牺牲质量换取时间，这就错了。任何以牺牲质量为代价、追求速度而获得短期收益的行为都是错误的，不能长久。

降低质量犹如杀鸡取卵，虽能暂时加快速度，但日后必定要付出更大的代价。因此，质量也应该固定，并且永远要执行最高标准。

* + 1. 范围

时间、预算和质量都被固定了，项目中唯一能变动的就只剩下一个因素了：范围。

如果要做的工作太多，我们就需要从中选取业务价值最高的，优先级最高的先做。如果现实与计划不符，一般会对计划而不是现实进行调整。但是大多数我们被告知：技术是固定的、不可移动的和僵化的，永远也不应改变。事实并非如此。因此，面对四个因素，时间、预算和质量都是固定的，但在项目的范围方面要进行灵活的处置。

* + 1. ……
  1. 需求管理

Martin Fowler曾经失望地说，自己倾注多年心血写了一本书，却总是惊讶地发现人们经常误解他要表述的核心思想。多数情况下，在描述和收集客户对软件的预期方面，需求文档所起到的作用真的很小。用文档形式收集需求的问题不在于量大，而在于沟通不畅。首先，你无法与文档进行对话（起码不方便）。而在某些极端情况下，句法上的错误还会使公司损失几百万。

因此，在敏捷项目中，无论是团队内还是团队间，传递信息最有效的方法就是面对面的交流。

* + 1. 创建用户故事（user story）

敏捷用户故事是对特性的简短描述，我们的客户希望软件具有这些特性。通常用户故事会写在小索引卡片上，目的就是鼓励我们走出去，与客户对话。

优秀的用户故事第一要素就是它要对客户有价值。怎么才算有价值呢？客户愿意为之掏腰包的东西就算是有价值。用户故事要对业务有意义。正因为如此，我们在记录时总是尽量使用客户能听懂的简单术语，避免技术问题把他们搞得云里雾里。第二要素就是要有始有终。一个优秀的用户故事会从头至尾贯穿整个架构的所有层面，传达一些有价值的东西。另外，优秀的用户故事还应具备如下几个特征：独立、可协商、可测试、小型与可估算等。

归纳起来，用户故事和文档在收集需求方面可以对比如下。

|  |  |
| --- | --- |
| 用户故事 | 规格与需求文档 |
| 精益、准确、及时 | 过度不准确、过时 |
| 鼓励面对面交流 | 鼓励猜测 |
| 简化的设计 | 复杂的设计 |
| 廉价、迅速而容易创建 | 昂贵、缓慢而很难创建 |
| 永不过时 | 经常过时 |
| 以最新知识为基础 | 极少或没有吸纳新知识 |
| 支持实时反馈 | 不支持实时反馈 |
| 避免了准确度上的错觉 | 增加了准确度上的错觉 |
| 允许基于团队的协作和创新 | 不鼓励开放的协作和创新 |

* + 1. 点数系统（story point）

科学证实，人类实际上擅长的是相对估算。所以，我们在对一张故事卡估算点数的时候，也进行相对估算。通过故事之间的相对估算，测量出团队的开发速度，我们就能得出所有的要素，形成敏捷计划。

通过点数系统，我们可以追踪进程并进行相对评估，同时又不用担心实际情况和估算有何出入。在实际操作中，一般我们会把整个迭代的故事卡都展示出来，组内的成员会选出他们认为复杂程度最小的一张卡，点数定位1。其他的卡都相对于这张卡，各自估算出一个点数（story point）。对于点数的取值，我们一般采用斐波拉契数列中的数值（1，2，3，5，8，…）。因为，我们认为估算出来的点数越高，这张卡的复杂度和不确定性就会越高。所以，建议一张卡的点数最好不要超过5。如果超过5点了，我们就要考虑这张卡是否要被拆分几张更小的卡。

* + 1. 迭代模型（iteration）

传统的瀑布模型是由Royce在1970年最初提出的软件开发模型，瀑布式开发是一种老旧的计算机软件开发方法。

瀑布模型式是最典型的预见性的方法，严格遵循预先计划的需求分析、设计、编码、集成、测试、维护的步骤顺序进行。步骤成果作为衡量进度的方法，例如需求规格，设计文档，测试计划和代码审阅等等。瀑布式的主要的问题是它的严格分级导致的自由度降低，项目早期即做出承诺导致对后期需求的变化难以调整，代价高昂。瀑布式方法在需求不明并且在项目进行过程中可能变化的情况下基本是不可行的。

敏捷项目中采用增量式迭代开发，是一种应对快速变化的需求的一种软件开发能力。它们的具体名称、理念、过程、术语都不尽相同，相对于“非敏捷”，更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重软件开发中人的作用。

在迭代式开发方法中，整个开发工作被组织为一系列的短小的、固定长度（如2周）的小项目，被称为一系列的迭代。每一次迭代都包括了需求分析、设计、实现与测试。采用这种方法，开发工作可以在需求被完整地确定之前启动，并在一次迭代中完成系统的一部分功能或业务逻辑的开发工作。再通过客户的反馈来细化需求，并开始新一轮的迭代。

这种开发方法有如下一些的优点：

* 降低风险
* 得到早期用户反馈
* 持续的测试和集成
* 使用变更
* 提高复用性
  + 1. ……
  1. 版本控制
     1. 集中化的版本控制系统

集中化的版本控制系统（ Centralized Version Control Systems，简称 CVCS ），诸如 CVS，Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。多年以来，这已成为版本控制系统的标准做法。

这么做最显而易见的缺点是中央服务器的单点故障。若是服务器当机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。如果中央服务器的磁盘发生故障，并且没做过备份或者备份得不够及时的话，还会有丢失数据的风险。最坏的情况是彻底丢失整个项目的所有历史更改记录，被客户端提取出来的某些快照数据除外，但这样的话依然是个问题，你不能保证所有的数据都已经有人提取出来。本地版本控制系统也存在类似问题，只要整个项目的历史记录被保存在单一位置，就有丢失所有历史更新信息的风险。

* + 1. 分布式版本控制系统

于是分布式版本控制系统（ Distributed Version Control System，简称 DVCS ）面世了。在这类系统中，诸如 Git，Mercurial，Bazaar 还有 Darcs 等，客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。因为每一次的提取操作，实际上都是一次对代码仓库的完整备份。

* + 1. ……
  1. 协作开发
     1. 敏捷沟通

故事计划会议（SPM，story planning meeting），这就是适时分心检查点会议。在会议期间，我们将与客户一起对接下来的用户故事的测试标准进行检查，与开发者一起对估算进行检查。从总体上来保证已经完成了下一批次迭代故事的准备工作。

迭代计划会议（IPM，iteration planning meeting），就是与客户一起为下次迭代的工作做计划。需要检查团队的速率，检查接下来的故事，然后共同确定出你和团队在下一次迭代中所承诺完成的工作量。

回顾会议（Retro）,这个会议一般在一个迭代快要结束的时候开，回顾一下在本次迭代中，我们有哪些是做的比较好的，有哪些是做的不够好的，还有哪些是需要改进的。主要是围绕以上三个方面讲述。

站立会议（Standup），日常站立会议的目的就是与团队成员迅速的分享一些重要信息，一般持续5到15分钟，主要围绕三个方面讲述。昨天做了些什么，今天准备要做什么，在做的过程中遇到了哪些障碍。

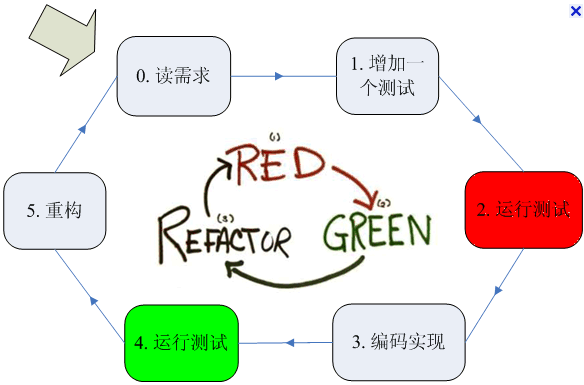
* + 1. 重构（偿还技术债）

技术债，即我们经常以赶时间和进度为借口，在编写代码时走捷径，写下的代码晦涩难懂，冗赘重复，所有这些错误就是我们所要背负的技术债务。表现形式有多种，如面条式代码、过度复杂、重复等。

在不影响整体外部行为的前提下，不断地对软件进行细微的设计和改进，我们把这种渐进式的实践叫做重构。在对代码进行重构时，我们不会增加新功能，甚至也不会去修复Bug。相反，我们会通过将代码变得更易于理解和改变来提升代码的可读性。

* + 1. 测试驱动开发（单元测试、功能测试、集成测试……）

测试驱动开发（TDD，test driven development）是一种软件开发技巧，能在非常短的开发周期内对软件进行渐进式的设计。其工作原理如下：首先，在编写任何系统的新代码之前，都要编写失败的单元测试，展示新代码的预期作用。然后，不管付出什么代价，你都要保证测试通过。如果发现测试可以全面执行，那就添加新代码。如果不能就再多做一些工作，直到测试通过。然后再回过头来解决有问题的代码，同时想尽各种办法将测试通过。此时，你就是在消除重复，确保一切都是精益的、简洁的、并保持尽可能整洁。



* + 1. 结对编程

两个有经验的程序员同时在一起工作，看起来好像浪费了一个人的时间：但实际上的效果确实完成了更高质量的代码。程序编的不那么容易出BUG，而且代码也写得更为优雅和紧凑。更重要的是，结对编程可以更好的进行知识的传递，还可以降低项目的风险，当有成员请假或者离职时，他的工作很容易就可以托付给其他人。因为，在不断的轮换结对的人员之后，团队内的成员对整个系统的了解更深入，达到了共享知识的效果。

* + 1. ……
  1. 持续集成
     1. 持续集成的特点

源代码编译。持续的源代码编译是CI系统中最基本、最常见的特征之一。编译指的是根据人们可读的源代码生成可执行代码。

数据库集成。一些人认为源代码集成和数据库集成是完全独立的两个过程，它们总是由不同的小组来完成。很不幸，因为数据库是软件的一部分，利用CI系统，可以确保利用单一的来源集成数据库。

测试。没有自动化的/持续的测试的CI不能算是CI。没有自动化测试，开发者或其他项目风险承担者对软件的变更很难有信心。因此，在使用CI系统的项目中，大部分开发会使用JUnit、NUnit或其他xUnit框架等单元测试工具来进行测试。还可以通过CI运行不同类型的测试，加快构建过程，比如单元测试、功能测试、集成测试等。

审查。自动化的代码审查可以通过强制遵守规则来增加软件的品质。其中使用Checkstyle就会生成很详细的审查报告。利用这样的报告可以持续监控编码标准和品质方面的测量指标。

文档与反馈。许多开发者深信文档是属于源代码的，实际上，清楚、简洁的代码和选择得当的类名、变量名和方法名就是最好的文档。

* + 1. 自动化构建

通过编写自动化的构建脚本，减少了软件项目中手工的、重复的、容易出错的过程。有许多构建工具可供选择，比较流行的构建工具有针对Java的Ant和针对.NET的NAnt。使用专门为构建软件设计的脚本工具，而不是一组自定义的shell脚本或者批处理脚本，能够以更有效的方式来开发一致的、可重复的构建解决方案。对于每次构建的自动集成，我们可以选择的CI工具有很多，比如Jenkins、Go等。

构建软件的主要步骤大致如下：

1. 利用NAnt、Rake、Ant、Maven或者Gradle这样的构建脚本工具创建构建。先让它保持比较简单，在右后可以添加更多的过程。
2. 在构建脚本中添加集成按钮所包含的每个过程（清理、编译等）。
3. 从IDE或从命令行运行构建脚本，构建软件。

清理

编译源代码

集成数据库

执行测试

执行审查

部署软件

* + 1. 数据库集成

持续数据库集成（CDBI，Continuous Database Integration）是一个过程，即每次项目的版本控制库中发生变更时，重建数据库和测试数据。对数据库的变更也可进行版本管理，推荐使用liquibase，支持多种类型的数据库，使用XML文件进行配置的时候可以很好的兼容不同的数据库。

下面列出了项目中通常执行的数据库集成活动，这些活动可以自动化。

|  |  |
| --- | --- |
| 活动 | 描述 |
| 删除数据库 | 删除数据库和相关的数据，以便用同样的名字创建一个新数据库 |
| 创建数据库 | 利用数据定义语言创建数据库 |
| 插入系统数据 | 插入系统在交付时应该具备的初始化数据 |
| 插入测试数据 | 在多个测试实例中插入测试数据 |
| 其一数据库和数据 | 定期迁移数据库schema和数据 |
| 在多个环境下简历数据库实例 | 建立独立的数据库以支持不同的版本和环境 |
| 修改列属性和约束条件 | 根据需求和重构，修改表的列属性和约束条件 |
| 修改测试数据 | 更改多个环境下需要的测试数据 |
| 修改存储过程 | 在开发过程中多次修改并测试存储过程 |
| 取得对不同环境的访问权 | 利用ID、口令和数据库标识符登录到不同的数据库环境 |
| 备份/恢复大量数据 | 为特别打的数据集或整个数据库创建特殊的功能 |

* + 1. 自动化测试（包含check style、测试覆盖率、判断重复代码……）

如果我们要构建真正可靠的软件系统，我们就必须确保对象层面的可靠性，这只能通过成功的单元测试来实现。否则，我们就不能期望构建出高可靠性的应用程序。当然，只是为对象编写单元测试不一定能保证可靠性。测试必须有效地使用了被测对象，而且必须经常执行。因为软件系统中的对象相互之间进行通信，每次当系统发生变更时，测试都必须执行。在CI系统中包含持续的测试让我们能够做到这一点。为了能够方便快捷的执行测试，让单元测试、组件测试、系统测试等自动化是非常有必要的。

* + 1. 反馈机制（包含邮件、短信、弹窗提示、build灯、声音……）

反馈是集成按钮的关键输出。如果没有反馈，CI的所有方面都不会有用。我们希望快速构建并让构建快速失败，其目的就是为了获得反馈信息。在正确的时间，以正确的方式，将正确的信息发送给正确的人——CI是让这种反馈信息自动化、目标化和实时化（持续化）的最好工具。

反馈信息可能是手机里收到的一条短信息，计算机音响里播放的一段定制的声音、一封电子邮件、一个构建灯的颜色、系统弹出的一个提示消息等。

* + 1. ……
  1. 持续部署

高效地创建能工作的软件是专业软件开发者存在的理由。没有成功的开发，软件就不可能存在。今天，我们创建和发布软件的频率要快很多，所以我们必须像开发过程一样，把这个过程做正确。持续集成因此需要“持续部署”，它是这些实践和步骤的顶点，让我们随时随地都可以发布能工作的软件，同时使得发布的工作量最小。

* 为资产打上标签

创建一个版本控制库的标签有利于标识并追踪资产，因为它清楚地说明了一组文件是在一起的。而且，标签使我们能够追踪一组文件的历史，而不仅仅是单个文件的历史。在某一时刻，单个文件的版本可能是不一样的。

* 清扫部署环境

在构建软件设计时，重要的是要确保没有漏掉文件或配置设置，漏掉这些文件或配置将导致软件失败。有不同的方法可以做到这一点。第一种方法是从什么都没有的一台计算机开始，每次加上“一层”，直到装上完整的系统。这通常是在一台测试计算机或分阶段构建计算机上执行的。在理想情况下，我们应该自动实现每一层的删除和安装。

* 为每一个构建版本打上标签

为每一个构建版本创建一个唯一的标识符，即“构建版本标签”，这通过两步完成：首先，版本控制库中的代码需要一个标签；其次，对代码的构建动作需要一个标签。这些构建标签让大家知道在特定的环境下使用了什么版本的代码，而且通过构建标签，缺陷、功能和新的需求都可以以这一份代码作为依据。但是，请注意版本控制库标签和构建标签之间的不同。版本控制库说明一组文件是相关的。构建标签说明构建的二进制输出是唯一的。

* 执行所有测试（保证通过率100%）

虽然子啊开发的某些阶段中可能只需要执行部分测试，但在打包部署之前，必须执行所有的测试并成功通过。就这么简单，执行所有的自动化测试，从单元测试到功能测试，保证通过率是100%。

* 生成报告（包含功能、缺陷、需求）

生成自动化构建的反馈信息有助于大家理解将要发布的构建版本中的确切情况，包括这个构建版本中那些文件不同，修复了哪些缺陷，实现了哪些功能。通过记录这些信息，感兴趣的人可以验证他们期望的东西是否实现。

* 版本的回滚

能够“撤销”部署是有效的开发中的重要一环。许多开发团队都遇到过这种情况，即需要有以前的版本取代新的有缺陷的代码，因为以前的版本工作的更好。通过构建版本标签和版本控制库标签，这个过程只需要索取想要的版本就可以了。

* + 1. ……
  1. 敏捷运维
     1. ITIL（Information Technology Infrastructure Library）介绍
     2. DevOps （Development & Operations，公认为下一代的"ITIL"）介绍
     3. ……

1. 基于敏捷开发的图书分享系统的实现
2. 总结与展望