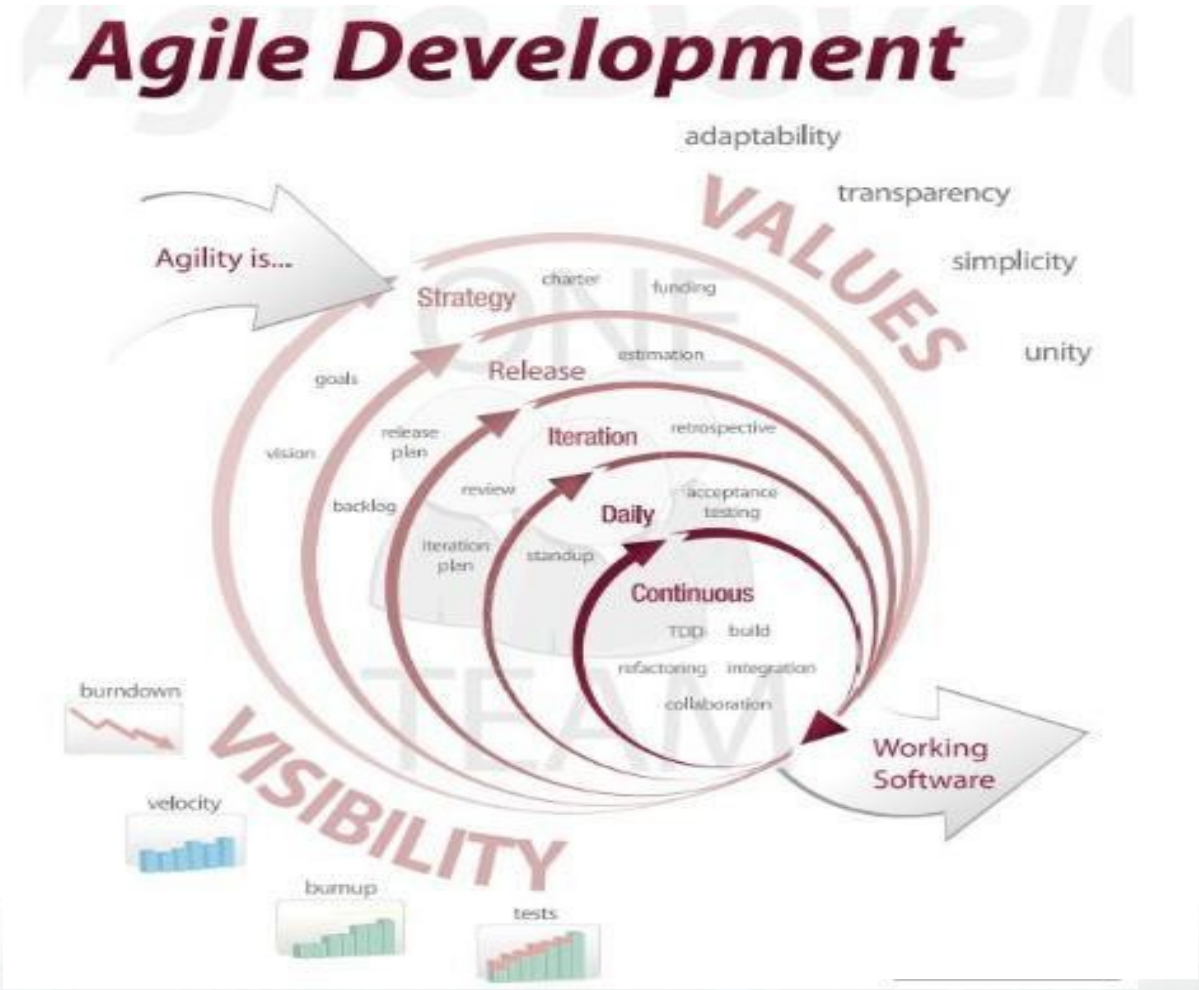
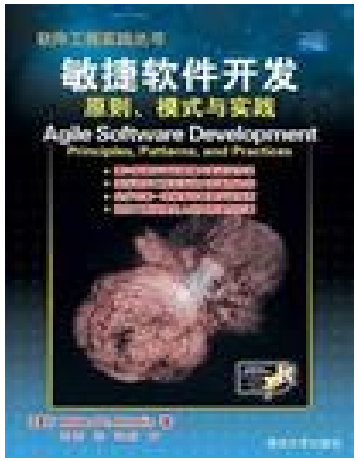
敏捷软件开发


Agile Development
adaptability
transparency
simplicity
unity
VALUES
Agility is...
Strategy
charter
funding
Release
estimation
goals
vision
release plan
Iteration
retrospective
backlog
review
Daily
acceptance testing
iteration plan
standup
Continuous
TDD
build
refactoring
integration
collaboration
ONE TEAM
Working Software
VISIBILITY
burndown
velocity
burnup
tests

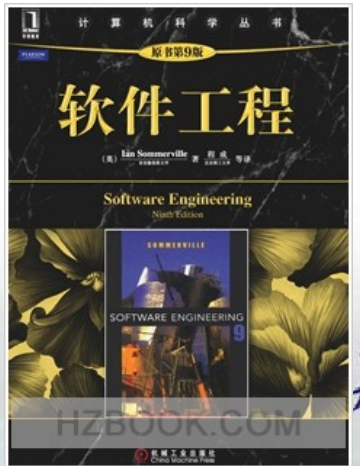
From:

1、《Agile Software Development》
--Robert C. Martin



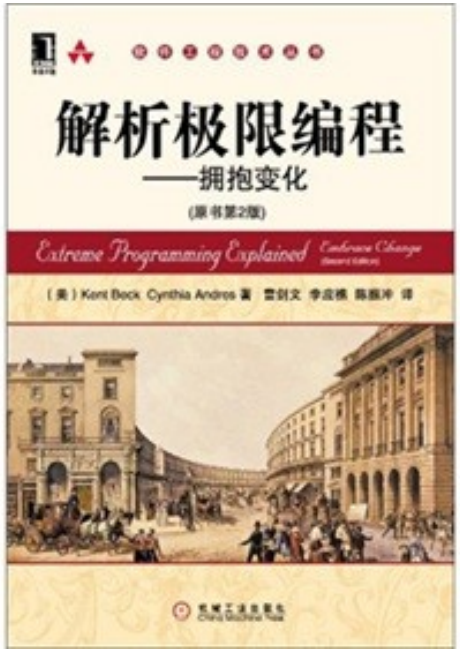
2、《Software Engineering-A Practitioner' Approach》 Seventh Edition
--Roger S.Pressman

3、《Software Engineering》 Ninth Edition
--Ian Sommerville



From:

4、解析极限编程—拥抱变化（第 2 版）
（美） Kent Beck 等著，雷剑文等译



5、 http://zh.wikipedia.org/wiki/ 敏捷软件开发

一、概念

敏捷软件开发又称敏捷开发，是一种从 1990 年代开始逐渐引起广泛关注的一些新型软件开发方法，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。

－－维基百科

敏捷软件工程是哲学理念和一系列开发指南的综合。这种哲学理念推崇让客户满意和软件的早期增量发布，小而高度自主的项目团队，非正式的方法，最小化软件工程产品以及整体精简开发。开发的指导方针强调超越分析和设计（尽管并不排斥这类活动）的发布，以及开发人员和客户之间主动和持续的沟通。

－－《 Software Engineering-A Practitioner’s Approach 》 Seventh Edition

--Roger S.Pressman

二、 敏捷联盟

2001 年初，由于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（ Agile ）联盟。

敏捷联盟宣言：

我们正通过亲身实践以及帮助他人实践，揭示更好的软件开发方法。通过这项工作，我们认为：

- 个体和交互　胜过　过程和工具
- 可以工作的软件　胜过　面面俱到的文档
- 客户合作　胜过　合同谈判
- 响应变化　胜过　遵循计划

- 个体和交互　胜过　过程和工具
  - 人是获得成功最重要的因素
  - 一个优秀的团队成员能很好地和他人合作，即合作、沟通以及交互能力要比单纯的编程能力更重要
  - 合适的工具对成功很重要，但不要过分夸大工具的作用
  - 团队的构建比环境的构建重要得多
- 可以工作的软件　胜过　面面俱到的文档
  - 没有文档的软件是一种灾难，但过多的文档比过少的文档更糟糕
  - 对团队而言，需要编写并维护一份系统原理和结构方面的文档

- 客户合作　胜过　合同谈判
  - 成功的项目需要有序、频繁的客户反馈。不是依赖于合同或者关于工作的陈述，而是让软件的客户和开发团队密切地在一起工作，并尽量经常提供反馈
  - 那些为开发团队和客户的协同工作方式提供指导的合同才是最好的合同
- 响应变化　胜过　遵循计划
  - 响应变化的能力常决定一个软件项目的成败
  - 计划不能考虑过远
  - 较好的做计划的策略是：为下两周做详细的计划，为下三个月做初略的计划，再以后就做极为初略的计划

三、敏捷原则

---- 这是敏捷实践区别于重过程的特征所在

1）最优先要做的是：通过尽早地、持续地交付有价值的软件来使客户满意。—获取有质量软件的理念

2）即使到了开发后期，也欢迎改变需求。敏捷过程利用变化来为客户创造竞争优势。 —关于态度的声明

3）经常交付可工作的软件，其时间间隔可以是几周到几个月。 交付的时间间隔越短越好。

— 项目规划的理念（涉及如何处理文档和软件项目开发之间的关系）

4）在整个项目开发期间，业务人员和开发人员必须天天在一起工作。—团队组成和精神问题

5） 不断激励开发人员，开展项目的有关工作。给他们提供所需要的环境和支持，并信任他们能够完成所承担的工作。 —“领导”的含义 - 涉及管理功能

6）在团队内部，最有效果的、最有效率的传递信息的方法，就是面对面的交谈。

—获取开发信息（需求、技术信息和项目信息等）的途径

7） 首要的进度度量标准是工作的软件。

— 进度度量的理念

8）敏捷过程提倡可持续的开发速度。责任人、开发者和用户应该能够保持一个长期的、恒定的开发速度。

— 项目“持续发展”的能力

9） 不断关注优秀的技能和设计，增强敏捷能力。

—提高敏捷能力的一种途径

10） 简单是根本的

— 使未完成的工作最小化的艺术

11） 最好的体系结构、需求和设计，出自自己组织的团队。

— 团队观念 ---- 一种软件项目管理的理念

12） 每隔一段时间，团队对如何才能有效的工作进行反省，然后对自己的行为进行适当的调整。

— 自我调整和适应

注：以上 12 条是敏捷开发的实践原则。实践的语义比过程更宽泛，包扩活动以及与活动相关的人和基础设施。

四、极限编程

极限编程（eXtreme Programming，简称 XP）是敏捷方法中最显著的一个。它由一系列简单却相互依赖的实践组成。

“如果你的组织准备好了要改变开发软件的方式，有缓慢的增量方法：一个一个地解决问题；同样也有快速的途径：跳进 XP 来。不要被名字吓到，它根本不是那么极限。大部分是多年积累的老处方和常识，被很好地整合起来，去除了这些年来积累的多余脂肪”

---Philippe Kruchten，加拿大 UBC 大学教授

1、极限编程包含的实践

（1）客户作为团队的成员

□□□客户与开发人员一起紧密的工作，相互了解所面临的问题，并共同解决之。其中，客户（人或团队）的主要责任是定义产品特征、并对这些特征进行优先排序。

注意：“如出现不能一起紧密工作的情况，应该寻找可以代表真正客户的、并可一起工作的人”。

（2）“用户素材”(user stories）

含义：为了了解与项目需求有关的内容，采用“用户素材”(user stories）——一种规划工具，作为在进行关于需求谈话时所使用的助记工具。通常，在客户的索引卡上记录认可的一些词语， 与之同时，在该卡上写下关于需求的估算。

（3）短的交付周期

含义：短的交付周期是指每隔两周，就交付一次可工作的软件。这意味着每两周的迭代都实现了“涉众”的一些需求；并在每次迭代结束时，可给“涉众”演示由迭代所生成的系统，以得到他们的反馈。

显然，这一实践涉及迭代计划和交付计划的制定。

(4）验收测试

- 作用：通过验收测试，捕获用户素材的有关细节。
- 编写时间：在要实现该用户素材之前或实现该用户素材期间进行编写。
- 编写工具：编写验收测试，使用能够让它们自动、反复运行的某种脚本语言。

(5) 结对编程

结对编程的含义是：共同设计、共同编写、功劳均等，以促进知识在全队的传播。

注：Laurie Williams 和 Nosek 的研究表明，结对不但不会降低团队的开发效率，而且还会减少缺陷率。

(6) 测试驱动的开发

含义：首先对产品的某一功能编写一个单元测试。由于该功能还不存在，因此它一定会运行失败。然后编写这一功能的代码，使该测试得以通过。

益处：为了测试用例通过而编写代码，这样的代码称为可测试的代码。优点是：由于要独立对它进行测试，因此可激励解除模块之间的耦合，使模块之间具有低的耦合。

(7) 集体所有权

集体所有权的含义：

① 编程中的每一结对，都具有“检出”（check out）任何模块的权力；

② 没有程序员对一个特定的模块或技术单独负责；

③ 每一个人都参与 GUI 工作，每一个人都参与中间件方面的工作，每一个人都参与数据库方面的工作。

没有人对结对所编写的模块和技术具有更多的权威。

注意，这并不意味着开发人员没有自己的专业知识。

(8) 持续集成

持续集成的含义是：

程序员每天可以多次检入（check in）他们的模块进行集成。

其中，最重要的是，要确保所有的测试都能通过：

① 可以把新的代码集成到代码库中，可以对代码进行合并；

② 必要时，可以和检入的程序员进行协商。一旦集成了他们的更改，就构造了新的系统，从而要运行系统中的每一个测试，包括当前所有运行的验收测试。一旦所有的测试都通过，这才算完成这一次的检入工作。

(9) 可持续的开发速度

含义：团队必须有意识地保持稳定、适中的速度。

(10）开放的工作空间

含义：团队在一个充满工作气氛 (例如“墙壁上挂满了状态图表、任务明细表、 UML 图等) 的房间中一起工作，结对的每个人都可了解对方的工作状态，可得知另一个人何时遇到了麻烦，并可适宜地进行交流和讨论。

密歇根大学一项研究表明：

在“充满积极讨论的屋子里工作，生产率不但不会降低，反而会成倍的提高。”

(11) 规划游戏 (planning game)

含义： 业务人员（客户）决定特征以及特征的重要性；

开发人员决定实现一个特征所花费的代价。

即：在每一次发布和迭代开始时，开发人员基于最近一次提过

所完成迭代或最近一次发布所使用的工作量，为客户提供一个预算；客户选择那些所需的、成本合起来不超过该预算的用户素材。

可见，其本质是：业务人员和开发人员之间的职责划分．

益处是：采用短周期的迭代和发布，客户和开发人员很快就会适应项目开发节奏；客户会了解开发人员的速度，

(12) 简单的设计

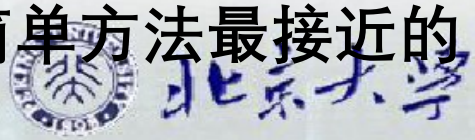
仅关注计划中本次迭代所要完成的用户素材，并在一次次迭代中，不断变迁系统设计，使之对正在实现的用户素材始终保持最佳状态，使团队所进行的设计尽可能的简单、具有表现力．

其中，三条指导原则是：

① 尽可能寻找最简单的方法，实现当前的用户素材。例如：

- 如果能够使用文件，就不去使用数据库或 EJB；
- 如果能使用简单的 socket，就不去使 ORB 或 RMI （远程方法调用）；
- 如果能够不使用多线程，就别去使用之。

然后，选择一种能够实际得到的、和该简单方法最接近的解决方案



② 延迟基础设施的需求决策

有关基础设施（数据库、 ORB 等）的引入，团队应对此进行认真的考虑，只有在有证据，或至少有十分明显迹象表明引入这些基础设施比继续等待更加合算时，才能引入之。

③ 一次代码，并且只有一次

不允许出现重复的代码。一旦出现，必须消除之，以减少代码之间的耦合。其中，消除重复代码的最好办法是抽象。

引起代码重复的因素很多，一个最明显的例子是：用鼠标选中一段代码四处粘贴。

(13) 重构

① 重构的含义：

重构就是在不改变代码行为的前提下，对其进行的一系列改造（transformation），旨在改进系统结构的实践活动。通过重构，可逐步改进系统设计和体系结构。

② 重构方式：

❶ 在进行每次改造时，要进行单元测试，确保这次改进没有造成任何破坏，保持系统可以工作。

❷ 重构要不断进行，每隔一个小时或更短的时间。这样可以不断地保持尽可能干净、简单且具有表现力的代码。

(14) 隐喻（metaphore）

隐喻是 XP 中形成系统一个全局视图的重要实践。隐喻是每个人（客户、设计人员以及管理者）可以讲述的系统如何工作的故事。

例如要开发一个网络流量分析系统。该系统每 30 分钟，轮询一些适配器，从中获取监控数据。对此，可以把每个网络适配器所提供的由几个变量组成的数据，看作是“面包切片”，把其中的一个变量看作是“面包屑”，而把分析程序看作是“烤面包机”。

通过隐喻，可以将整个系统连接在一起，使所有单独模块的位置和外观变得明显、直观。如果模块的外观与整个系统的隐喻不符，则就知道该模块是错误的。

应用极限编程实践中应注意的两个问题

(1) 应将这组简单的实践融为一体，相互依赖。

(2) 可增加一些实践或对其中一些实践进行修改。

2、极限编程过程

XP 使用面向对象方法作为推荐的开发范型，它包含了策划、设计、编码和测试 4 个框架活动的规则和实践。

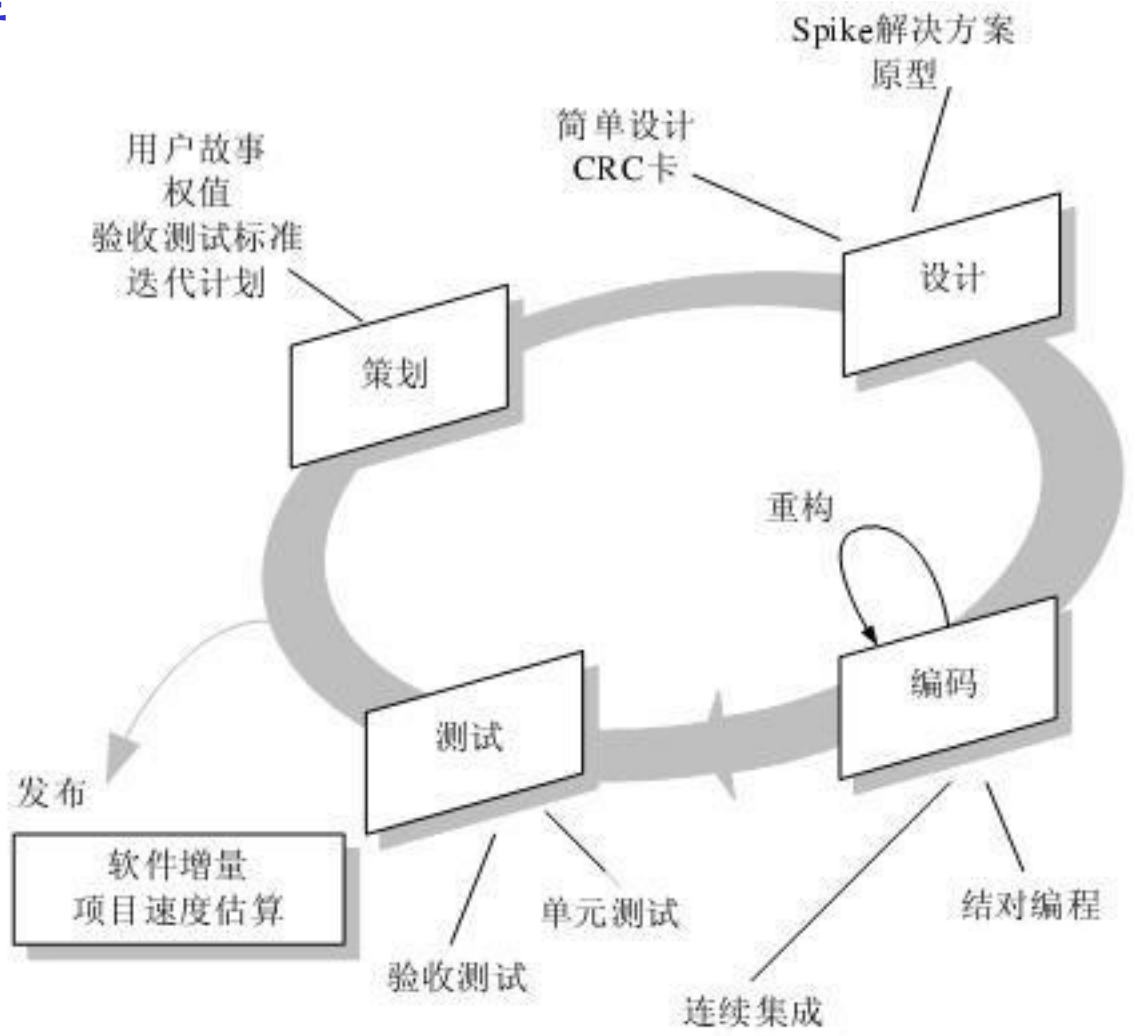


图　极限编程过程

2.1 策划

策划活动开始于倾听，是一个需求获取活动，该活动要使 XP 团队技术成员理解软件的商业背景以及充分感受需要的输出和主要特征及主要功能：

（1）产生一系列的用户故事，用于描述即将建立的软件所需要的输出、特征和功能。每个故事由客户书写并置于一张索引卡上，客户根据对应特征或功能的综合业务价值标明故事的权值（即优先级）。

（2）XP 团队成员评估每一个故事并给出以开发数周为度量单位的成本。如果某个故事的成本超过了 3 个开发周，则将请客户把该故事进一步细化，并重新赋值并计算成本。

（3）客户和 XP 团队共同决定把故事分组并置于 XP 团队将要开发的下一个发行版本中（下一个软件增量）。

（4）一旦认可对下一个发布版本的基本承诺（包括的故事、交付日期及其他项目事项），XP 团队将以下述三种方式之一对开发的故事进行排序：

(a) 所有选定故事将在几周之内尽快实现；

(b) 具有高优先级的故事将移到进度表的前面并首先实现；

(c) 高风险故事将首先实现。

在开发过程中，客户可以增加故事，改变故事的权值，分解或者去掉故事。然后， Xp 团队要重新考虑所剩余的发行版本并相应修改计划

2.2 设计

XP 设计严格遵守简洁原则。

（1） XP 设计鼓励使用 CRC （类－责任－协作者）卡确定和组织当前软件增量相关的面向对象的类。 CRC 卡也是 XP 过程中的唯一的设计工作产品。

（2）如果某个故事设计遇到困难， XP 推荐立即建立这部分设计的可执行的原型，实现并评估设计原型（被称为 Spike 解决方案），其目的是在真正的实现开始时降低风险，对可能存在的设计问题的故事确认其最初的设计。

2.3 编码

（1） XP 推荐在故事策划和初步设计完成之后，团队不是直接编码，而是开发一系列用于检测本次（软件增量）发布的所有故事的单元测试。

（2）建立单元测试，开发者就集中精力编码，开发必须实现的内容以通过测试。

（3）编码中强调结对编程。

(4) 完成结对编程的任务后，将所开发的工作与其他人的工作集成起来。

2.4 测试

（1）在编码之前建立单元测试是 XP 方法的关键。所建立的单元测试应当使用一个可以自动实施的框架，以便回归测试。

（2）一旦将个人的单元测试组织到一个“通用测试集”，每天都可以进行系统的集成和确认测试。

（3） XP 验收测试，也称为客户测试，由客户规定技术条件，并着眼于客户可见的、可评审的系统级的特征和功能。验收测试根据本次软件发布中所实现的用户故事而确定。

五、敏捷设计

1、问题的提出

为了应对需求变化，设计应尽力避免以下问题：

（1）僵化性（Rigidity）：是指难于对软件设计进行改动，

即使是简单的改动。例如：如果一个改动会导致有依赖关

系的连锁改动，那么设计就是僵化的。期间，必须改动的

模块越多，其设计就越僵化。

（2）脆弱性（Fragility）：是指在进行一个改动时，程序的

许多地方就可能出现问题，即使出现问题的地方与改动

（3）粘固性（Immobility）：是指在一部分的设计中包含了对其它部分有用的成分，但要想把这些成分分离出来就要付出很大的努力并具有相当大的风险，即设计难于复用。

（4）粘滞性（Viscosity）：存在 2 种表现形式：

❶ 软件粘滞性：是指当面临一个改动时，若想保持系统一致的设计方法是一件很困难的事情，很易采用一些破坏设计的方法，即难于做正确的事情。

❷ 环境粘滞性：是指环境的迟钝和低效。例如编译时间长。

- 也是难于做正确的事情。

（5）不必要的复杂性（Needless Complexity）：是指设计中包

含了当前没有用的成分，一方面可使软件变得复杂，另

一方面可使软件难于理解，即过分设计。

（6）不必要的复制（ Needless Repetition）：是指滥用“剪切”

和 “粘贴”等鼠标操作。

尽管 “剪切”和“粘贴”操作也许是有用的文本编辑操作，但对代码编辑来说，该操作却是灾难性的：

❶ 往往使开发人员忽略了抽象，从而使系统不易理解；

❷ 软件中的重复代码，使系统的改动变得更加困难

（7）晦涩性（Opacity）：是指模块难于理解。并且由于代码

随时不断演化，往往会模块变得越来越晦涩。

一般来说，若能持续地保持代码是清晰的和富有表现力的，

那么就可以减少代码的晦涩性。

2、防止软件腐化的基本途径

简言之，以变应变，尽力避免出现以上设计问题。

进一步说：

（1）团队几乎不进行预先（up-front）设计，因此不需要一个成熟的初始设计；

（2）团队通过多次使用单元测试和验收测试，支持系统的设计

尽可能的干净、简单，使设计保持灵活性和易于理解性；

（3）灵活、持续地改进设计，以便使每次迭代结束时所生成的系统具有满足那次迭代需求的设计。

3、采用敏捷设计的方法

包括：

使用一些设计原则，以保持软件是灵活的、可维护的；

掌握一些设计模式，以便针对特定问题权衡这些原则。

模式作为一种对经验的总结，
针对软件开发过程中一些反复出现、具有共性的问题
给出的良好的解决方案

例如：MVC （Model View Controller）

设计模式

3、采用敏捷设计的方法

敏捷设计是一个应用原则、模式和实践的过程，其间不断改善软件结构，保持系统设计在任何时间都尽可能的简单、干净（主要是指边界清楚，结构良好）和富有表现力，即：

❶ 它的功能　对于用户来说，通过直观、简单的界面呈现出恰当特征的程序；

❷ 它的内部结构　对于软件设计者来说，通过简单、直观的划分，使其具有最小耦合的内部结构；

❸ 创造过程　对于开发人员来说，每周都会取得一些重大进展，并生产出无缺陷代码的具有活力的团队过程.

六、一种敏捷过程模型— Scrum

Scrum(得名于橄榄球比赛) ： Jeff Sutherland 和他的团队在 20 世纪 90 年代早期发展的一种敏捷过程模型。

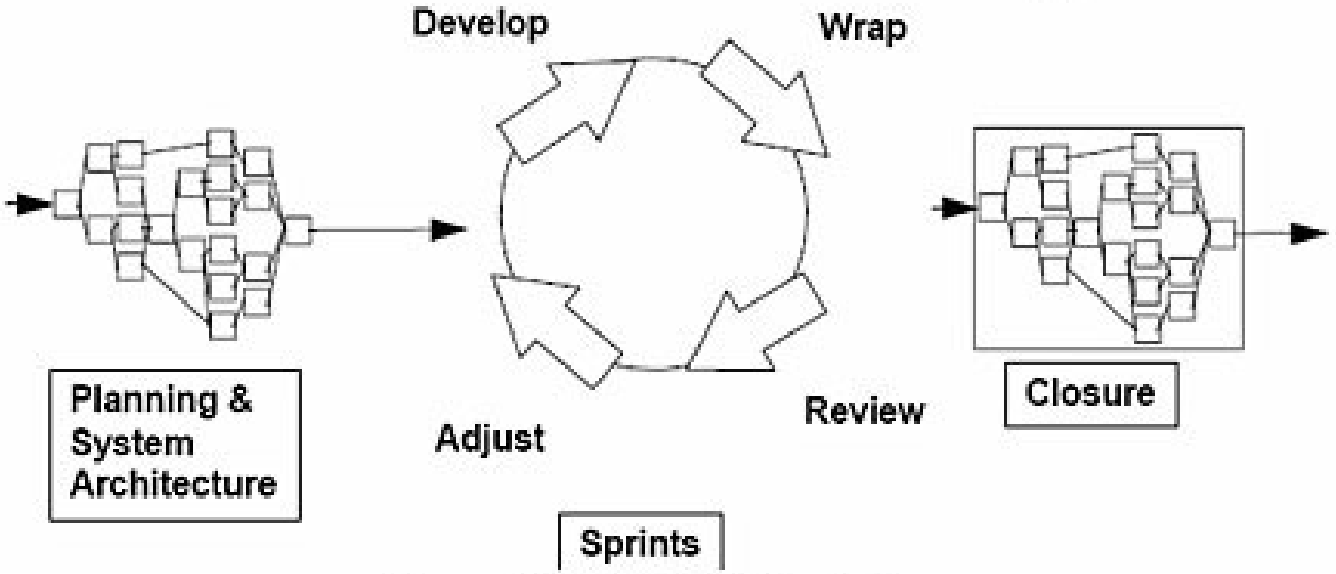


图 Scrum 过程

Scrum 有三个阶段：

- 规划纲要阶段：建立大致的项目目标和设计软件体系结构
- 一系列的冲刺循环，每个循环开发出一个系统增量
- 项目结束阶段总结项目，完善需要的文档，如系统帮助和用户手册，并总结从项目中获得的经验。

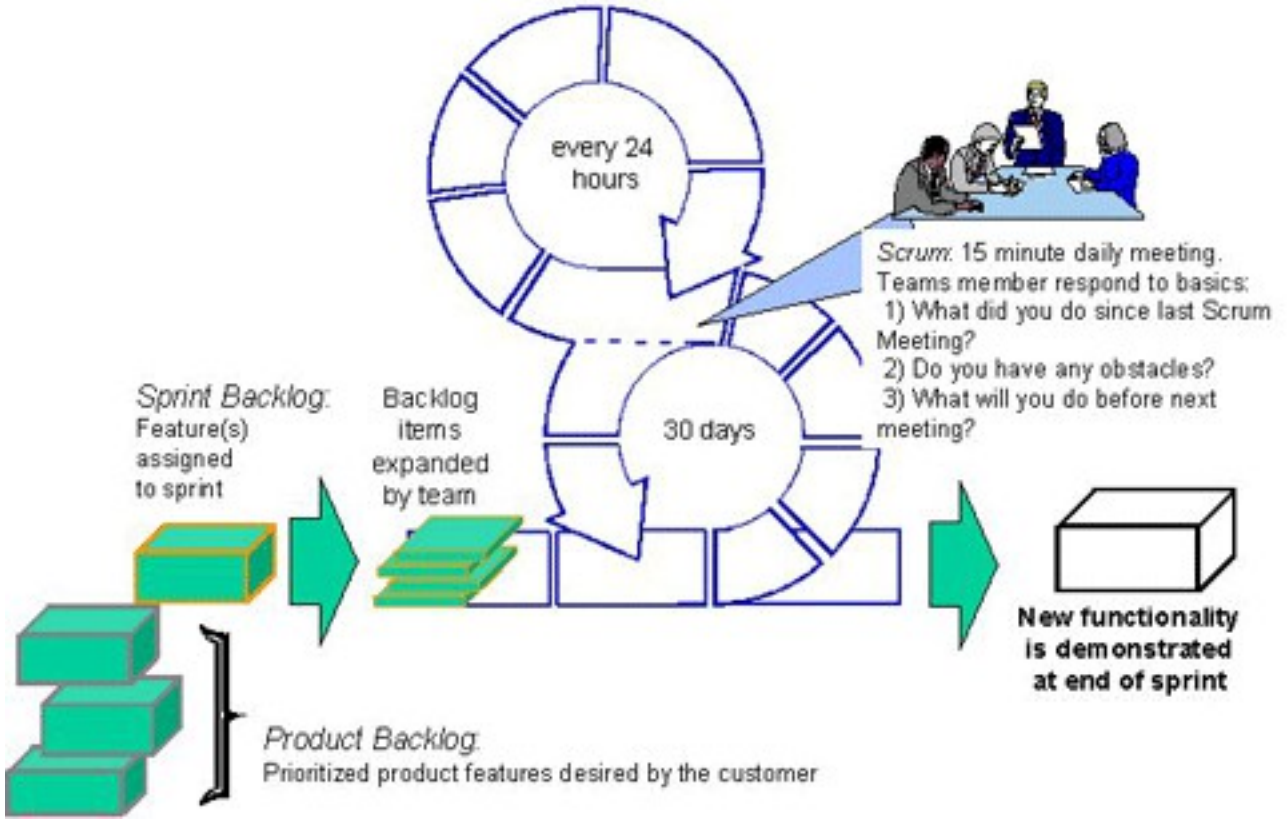


图　Scrum 过程流

- 待定项（backlog）：一个为用户提供商业价值的项目需求或特征的优先级列表
- 冲刺（sprint）：由一些工作单元组成，是达到待定项中定义的需求所必须的
- Scrum 例会：Scrum 团队每天召开的短会（一般为 15 分钟），会上每个成员回答三个问题（1）上次例会做了什么？（2）遇到什么困难？（3）下次例会前计划做什么？
- 演示：向客户交付软件增量

七、可扩展的敏捷方法

敏捷方法的开发是为在同一个房间办公与交流的小型团队使用的。但也可以将敏捷实践用到大型系统开发中。

（一）大型和小型系统开发的区别：

（1）大型系统经常由独立的、交互的子系统组成，不同的团队独立开发不同的子系统。

（2）大型系统包含了一系列已存在的系统与它们进行交互，所以许多系统需求关注这种交互，而不是如何适应于灵活性和增量式开发。

（3）当一个系统由多个系统集成时，开发工作中一个重要部分是系统配置而不是原始代码的开发。这不一定与增量式开发和频繁的系统集成兼容。

（4）大型系统和它们的开发流程通常受限于外部规则和规章限制，如要求写各种系统文档等。

（5）大型系统有很长的采购和开发时间，很难保持团队在整个周期中对系统连续的认识。

（6）大型系统通常具有不同的信息持有者，将不同的信息持有者加入到开发流程中很重要。

伸缩的敏捷方法：

1. 照搬放大的观点，关注如何将敏捷开发方法应用到哪些小型团队无法开发的大型项目中。
2. 渗透的观点，关注如何将敏捷方法介绍到拥有多年开发经验的大机构中。

（二）为了保留敏捷方法的基本内涵（弹性计划、频繁发布、持续集成、测试驱动开发、良好的团队沟通），敏捷方法应用到大型项目时必须做的调整：

（1）大型系统不可能只关注代码，必须做更多的预先设计（如软件体系结构）和系统文档（如数据库模式）。

（2）跨团队沟通机制必须建立和使用。

（3）当系统由很多个程序模块组成时，持续集成是不可行的，但需要保持频繁的系统构建和定期地发布系统。需要引入支持多团队软件开发的新的配置管理工具。

八、敏捷开发要点总结

1.敏捷方法是一种专注快速开发的增量式开发，频繁地发布软件、降低过程开销、生产高质量的代码。用户支持参与到开发过程中。

2.决断是否使用敏捷或计划驱动的方法取决于所开发系统的类型、开发团队的能力和开发系统的公司的文化。

3.极限编程是一种著名的敏捷方法，它集成了一系列的好的编程经验，如频繁地发布软件、连续地改善软件和让客户参与到软件开发团队中。

4.极限编程的一个特别长处是在创建程序特征之前开发自动测试，在增量集成进系统的时候所有的测试必须成功执行。

5. Scrum 是一种敏捷软件过程模型，它的核心是一组冲刺循环，开发一个系统增量有固定的时间周期。规划是基于积压的工作的优先权安排的，选择高优先权的任务开始冲刺循环。

6. 伸缩的敏捷方法对于大型系统是困难的。大型系统需要前期设计和一些文档。在实践中，当一个项目有几个独立的开发团队时，连续的集成是不可能的。