1. 敏捷联盟

敏捷联盟是 17 位不同敏捷开发方法的提倡者共同成立的，目的是推进敏捷开发方法的研究和应用，他们并不要求强制使用某种开发方法，而是提出了敏捷开发的几个核心价值和基本原则：

www.agilealliance.com

2. 敏捷开发横空出世

多数软件开发仍然是一个显得混乱的活动，即典型的"边写边改"（code and fix）。设计过程充斥着短期的，即时的决定，而无完整的规划。这种模式对小系统开发其实很管用，但是当系统变得越大越复杂时，要想加入新的功能就越来越困难。同时错误故障越来越多，越来越难于排除。一个典型的标志就是当系统功能完成后有一个很长的测试阶段，有时甚至有遥遥无期之感，从而对项目的完成产生严重的影响。

我们使用这种开发模式已有很长时间了，不过我们实际上也有另外一种选择，那就是"正规方法"（methodology）。这些方法对开发过程有着严格而详尽的规定，以期使软件开发更有可预设性并提高效率，这种思路是借鉴了其他工程领域的实践。这些正规方法已存在了很长时间了，但是并没有取得令人瞩目的成功，甚至就没怎么引起人们的注意。对这些方法最常听见的批评就是它们的官僚繁琐，要是按照它的要求来，那有做太多的事情需要做，而延缓整个开发进程。所以它们通常被认为是"繁琐滞重型"方法，或 Jim HighSmith 所称的"巨型"（monumental)方法。

作为对这些方法的反叛，在过去几年中出现了一类新方法。尽管它们还没有正式的名称，但是一般被称为"敏捷型"方法。对许多人来说，这类方法的吸引之处在于对繁文缛节的官僚过程的反叛。它们在无过程和过于繁琐的过程中达到了一种平衡，使得能以不多的步骤过程获取较满意的结果。 敏捷型与滞重型方法有一些显著的区别。其中一个显而易见的不同反映在文档上。敏捷型不是很面向文档，对于一项任务，它们通常只要求尽可能少的文档。从许多方面来看，它们更象是"面向源码"（code-oriented）。事实上，它们认为最根本的文档应该是源码。

但是，并不是说文档方面的特点是敏捷型方法的根本之点。文档减少仅仅是个表象，它其实反映的是更深层的特点：

1、敏捷型方法是"适配性"而非"预设性"。 重型方法试图对一个软件开发项目在很长的时间跨度内作出详细的计划，然后依计划进行开发。这类方法在计划制定完成后拒绝变化。而敏捷型方法则欢迎变化。其实，它们的目的就是成为适应变化的过程，甚至能允许改变自身来适应变化。

2、敏捷型方法是"面向人"的(people-oriented) 而非"面向过程"的(process-oriented)。 它们试图使软件开发工作顺应人的天性而非逆之。它们强调软件开发应当是一项愉快的活动。

以上两个特点很好的概括了敏捷开发方法的核心思想：**适应变化和以人为中心**

3. 敏捷开发的定义

从广义上来给敏捷开发下定义，敏捷开发（agile development）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。简言之，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。

4. 核心价值 core value

- ✧ Individuals and interactions over processes and tools

  个人和交流重于过程和工具
- ✧ Working software over comprehensive documentation

  正在运行的软件本身重于复杂的文档
- ✧ Customer collaboration over contract negotiation

  与客户的沟通和交流重于使用合同约束客户
- ✧ Responding to change over following a plan

  对变化的快速响应重于跟随计划

5. 基本原则 principles

- ✧ 最高目标是通过快速的和经常的发布软件满足客户的需要
- ✧ 提交软件的周期为几个星期到几个月
- ✧ 产生正确的软件是衡量进度的首要标准
- ✧ 主动接受需求的改变而不是拒绝
- ✧ 商务人员和开发人员工作在一起
- ✧ 个人必须有动力，要创造环境支持他们的要求，信任他们
- ✧ 最有效的交流方法是面对面的交流
- ✧ 最好的结构，需求和设计来自于自组织的团队（self-organizing team），允许任何人提出想法和建议
- ✧ 持续改进设计和编码
- ✧ 鼓励正常工作，减少长时间加班
- ✧ 保持简单，减少不必要的部分，认识到简单的设计比复杂的设计更难（simple design is harder to produce）
- ✧ 定期调整过程，获得更高效率

6. 敏捷开发技术的特点和优势

6.1. 个体和交互胜过过程和工具

人是获得成功的最为重要的因素。如果团队中没有优秀的成员，那么就是使用好的过程也不能从失败中挽救项目，但是，不好的过程却可以使最优秀的团队成员失去效用。如果不能作为一个团队进行工作，那么即使拥有一批优秀的成员也一样会惨败。团队的构建要比环境的构建重要得多。许多团队和管理者就犯了先构建环境，然后期望团队自动凝聚在一起的错误。相反，应该首先致力于构建团队，然后再让团队基于需要来配置环境。

6.2. 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。代码不是传达系统原理和结构的理想媒介。团队更需要编制易于阅读的文档，来对系统及其设计决策的依据进行描述。然而，过多的文档比过少的文档更糟。编制众多的文档需要花费大量的时间，并且要使这些文档和代码保持同步，就要花费更多的时间。如果文档和代码之间失去同步，那么文档就会变成庞大的、复杂的谎言，会造成重大的误导。虽然从代码中提取系统的原理和结构信息可能是困难的，但是代码是惟一没有二义性的信息源。在团队成员的头脑中，保存着时常变化的系统的脉络图（road map）。人和人之间的交互是把这份脉络图传授给他人的最快、最有效的方式。

6.3. 客户合作胜过合同谈判

不能像订购日用品一样来订购软件。你不能够仅仅写下一份关于你想要的软件的描述，然后就让人在固定的时间内以固定的价格去开发它。所有用这种方式来对待软件项目的尝试都以失败而告终。有时，失败是惨重的。告诉开发团队想要的东西，然后期望开发团队消失一段时间后就能够交付一个满足需要的系统来，这对于公司的管理者来说是具有诱惑力的。然而，这种操作模式将导致低劣的质量和失败。成功的项目需要有序、频繁的客户反馈。项目的需求基本处于一个持续变化的状态。大的变更是很平常的。在这期间，也会出现整个功能块被减掉，而加进来另外一些功能块。然而，合同和项目都经受住了这些变更，并获得成功。成功的关键在于和客户之间真诚的协作，并且合同指导了这种协作，而不是试图去规定项目范围的细节和固定成本下的进度。

6.4. 响应变化胜过遵循计划

响应变化的能力常常决定着一个软件项目的成败。当我们构建计划时，应该确保计划是灵活的并且易于适应商务和技术方面的变化。计划不能考虑得过远。

7. 敏捷开发技术的适用范围

✧ 项目团队的人数不能太多

- ✧ 项目经常发生变更
- ✧ 高风险的项目实施
- ✧ 开发人员可以参与决策

8. 敏捷开发小组的工作方式

前面提到的这 4 个核心价值观会导致高度迭代式的、增量式的软件开发过程，并在每次迭代结束时交付经过编码与测试的软件。接下来几节覆盖了敏捷开发小组的主要工作方式，包括：

- ✧ □□增量与迭代式开发
- ✧ □□作为一个整体工作
- ✧ □□按短迭代周期工作
- ✧ □□每次迭代交付一些成果
- ✧ □□关注业务优先级
- ✧ □□检查与调整

8.1. 增量与迭代式开发

增量开发，意思是每次递增的添加软件功能。每一次增量都会添加更多的软件功能。使用增量法逐渐地增进功能，那么如果开发花费的时间多于预期，就可以将迄今为止已经增量构建的功能发布出去。（在软件开发的实践中往往项目开发花费的时间大于预期。）

增量地释放版本，可以实际地得到已创造的商业价值。因为在人们开始使用构建好的软件之前，软件开发方是不会真正地得到投资回报的。在那之前，预期的商业价值只是一种估计。如果估算软件开发是困难的，那么就尝试估算投资回报率吧。

迭代式开发：在软件开发的早期阶段就想完全、准确的捕获用户的需求几乎是不可能的。实际上，软件开发经常遇到的问题是需求在整个软件开发工程中经常会改变。迭代式开发允许在每次迭代过程中需求可能有变化，通过不断细化来加深对问题的理解。迭代式开发不仅可以降低项目的风险，而且每个迭代过程以可以执行版本结束，可以鼓舞开发人员。

笔者对迭代开发的理解是先构建软件，然后评估它是否能够正常的工作，然后对其作出修改。而且是根据用户需求的不断变化，不停的对其进行修改，不停的校验是否满足客户的需求，通过迭代找到正确的解决方案。

从 XP、Crystal、RUP、ClearRoom 等方法学中对比，体会迭代设计的精妙之处：每一次的迭代都是在上一次迭代的基础上进行的，迭代将致力于重用、修改、增强目前的架构，以使架构越来越强壮。在软件生命周期的最后，除了得到软件，还得到了一个非常稳定的架构。对于一个软件组织来说，这个架构很有可能就是下一个软件的投入或参考。

8.2. 敏捷小组作为一个整体工作

项目取得成功的关键在于，所有的项目参与者都把自己看作朝向一个共同目标前进的团队的一员。“扔过去不管”的心理在敏捷开发项目中是没有市场的。分析师不会把需求“扔”给设计师；设计师和架构师不会把设计“扔”给编码人员；编码人员不会把只经过部分测试的代码“扔”给测试人员。一个成功的敏捷开发小组应该具有“我们一起参与其中”的思想。虽然敏捷开发小组是以小组整体进行工作，但是小组中仍然有一些特定的角色。有必要指出和阐明那些在敏捷估计和规划中承担一定任务的角色。

第一个角色是产品所有者(product owner)。产品所有者的主要职责包括：确认小组的所有成员都在追求一个共同的项目前景，确定功能的优先级以便总是在处理最具价值的功能，以及做出决定使得对项目的投入可以产生良好的回报。在商业软件开发中，产品所有者通常是公司的市场部门或者产品管理部门的人员。而在开发内部使用的软件时，产品所有者则可能是用户、用户的上级、分析员，也可能是为项目提供资金的人。

第二个角色是客户(customer)。客户是做出决定为项目提供资金或者购买软件的人。在一个开发内部使用的软件的项目中，客户通常是来自另一个团组或者部门的代表。在这样的项目中，产品所有者和客户的角色常常是重合的。而对一个商业产品来说，客户就是购买这个软件的人。无论是哪种情况，客户都可能是，也可能不是软件的用户(user)。用户当然也是一个主要的角色。

另一个值得注意的角色是开发人员(developer)。这里使用开发人员来概指所有开发软件的人。它包括了程序员、测试人员、数据库工程师、可用性专家、技术文档编写者、架构师、设计师，等等。使用这一定义，即使是产品所有者在很多项目中也可以被看作是开发人员。

最后一个角色是项目经理(project manager)。如 Highsmith(2004a)所述，在敏捷开发项目中，项目经理的角色发生了变化。敏捷开发项目经理会更多地关注领导而不是管理。在某些敏捷开发项目中，承担项目经理角色的人也会承担其他的角色，通常是作为开发人员，少数时候也会担任产品所有者。

8.3. 敏捷小组按短迭代周期工作

在敏捷开发项目中，对开发阶段没有什么重要的分隔——没什么先期需求阶段，然后是分析阶段，然后是架构设计阶段，等等。根据您实际选择的或者定义的敏捷开发过程，您可以在项目的最开始设置一个很短的设计、建模或其他阶段。但只要项目真正开始，在每次迭代中都会同时进行所有的工作(分析、设计、编码、测试，等等)。

迭代是受时间框(timebox)限制的，意味着即使放弃一些功能，也必须按时结束迭代。时间框一般很短。大部分敏捷开发小组采用 2～4 周的迭代，但也有一些小组采用长达 3 个月的迭代周期仍能维持敏捷性。大多数小组采用相对稳定的迭代周期长度，但是也有一些小组在每次迭代开始的时候选择合适的周期长度。

8.4. 敏捷小组每次迭代交付一些成果

比选择的特定迭代周期长度更重要的是，开发小组在迭代中把一个以上不精确的需求声明变成经过编码、测试，实际可以交付的软件。当然，大多数小组不会把每次迭代的结果都交付给用户；敏捷开发的目标只是让他们可以交付。这意味着开发小组在每次迭代中都会增加一些小功能，但是增加的每个功能都经过编码、测试，达到了可以发布的质量。

在每次迭代结束的时候让产品达到潜在可交付状态是很重要的。实际上，这并不是说小组必须全部完成发布所需的所有工作，因为他们通常并不会每次迭代都真的发布产品。例如，我曾经参与一个小组的工作，他们需要在发布产品之前对软硬件都进行 2 个月的 MTBF(Mean Time Between Failure，平均无故障时间)测试。他们不能缩短这 2 个月的时间，因为这

是他们的客户通过合同约定的，而且检查硬件故障也需要这么多时间。这个小组按照 4 周的迭代周期工作，他们的产品在每次迭代结束的时候除了没有进行这 2 个月的 MTBF 测试，都达到了确实可以发布的状态。

由于单次迭代并不总能提供足够的时间来完成足够满足用户或客户需要的新功能，因此我们需要引入更广义的发布(release)概念。一次发布由一次或以上(通常是以上)相互接续，完成一组相关功能的迭代组成。最常见的迭代一般是 2～4 周，一次发布通常是 2～6 个月。例如，在一个投资管理系统中，一次发布可能包括所有与买入和卖出共同基金和货币市场基金有关的功能。这需要 6 次 2 周的迭代来完成(大约 3 个月)。第二次发布可能增加了股票和债券交易，需要 4 次 2 周的迭代。可以按不同的间隔进行发布。也许需要 6 个月来完成第一次发布，而接下来的发布则可能在 3 个月以后，等等。

8.5. 敏捷小组关注业务优先级

敏捷开发小组从两个方面显示出他们对业务优先级的关注。首先，他们按照产品所有者所制定的顺序交付功能，而产品所有者一般会按照使机构在项目上的投资回报最大化的方式来确定功能的优先级，并将它们组织到产品发布中。要达到这一目的，需要根据开发小组的能力和所需新功能的优先级建立一个发布计划。要让产品所有者在确定功能的优先级时具有最大的灵活性，就必须在编写功能时使它们相互之间的技术依赖性最小化。如果选择一个功能要求先开发另外的 3 个功能，产品所有者就很难在发布计划中确定功能的优先级。开发小组不太可能达到绝对没有任何依赖的目标；但是，把依赖性控制在最低程度则是相当可行的。

其次，敏捷开发小组关注完成和交付具有用户价值的功能，而不是完成孤立的任务(任务最终组合成具有用户价值的功能)。达到这个目的的最佳方法之一就是采用一种表示软件需求的轻量级技术(Cohn 2004)，即用户故事。一个用户故事(user story)是从系统用户或者客户的角度出发对功能的一段简要描述。用户故事的形式很自由，没有什么强制性的语法。但是，按照大致符合这样一个形式来考虑用户故事是比较有益的："作为〈用户的类型〉，我希望可以〈能力〉以便〈业务价值〉。"以这样的模板作为例子，您可以得到一个用户故事说："作为购书者，我希望可以根据 ISBN 找到一本书，以便更快找到正确的书。"

用户故事是轻量级的，因为并不需要一开始就把它们全部收集和记录下来。与写下一篇冗长的需求说明相比，敏捷开发小组发现采用刚好及时的需求分析方法更有效。通常可以通过在记录卡片上写下一个用户故事的简短说明来开始，对较大的或者分布式的小组，则可以把它录入计算机中。不过故事卡片只是个开始，对每个用户故事，开发人员和产品所有者都应根据需要进行交流。这些交流在需要时进行并且应包含所有必需的人。使用基于用户故事的需求分析方法时，仍可能需要编写文档。不过，工作重点在很大程度上从文档编写转移到了口头的交流。

8.6. 敏捷小组进行检查和调整

任何项目开始的时候所建立的计划都不是对将来会发生些什么事情的保证。事实上，它只是一个当前的猜测。有很多事情可以让这个计划失效——项目成员的增减、某种技术比预期的效果好或差、用户改变了想法、竞争者可能会迫使我们做出不同的或者更快的反应，等等。对每个这样的变化，敏捷开发小组都把它看作为了更好地反映当前的实际情况而更新计划的机会和需要。

在每次新迭代开始的时候，敏捷开发小组都会结合在上一次迭代中获得的所有新知识作出相应的调整。如果小组认识到一些可能影响到计划的准确性或是价值的内容，他们就会调整计划。小组可能发现他们过高或过低地估计了自己的进展速度，或者发现某项工作比原来以为的更耗费时间，从而影响到计划的准确性。

计划的价值也可能由于产品所有者获得了期望用户或是可能用户的想法而发生改变。也许，基于用户看到以前的迭代交付的软件后产生的反馈，产品所有者认识到用户可能更想具备某类功能，而对另一个功能则没有之前所认为的那么看重。通过在发布中增加更多用户期待的功能而减少一些低价值的功能，可以增加计划的价值。

以上这些并不是说敏捷开发小组对优先级的改变是随兴而为的。在一次迭代到另一次迭代间优先级实际上是相当稳定的。但是，具有在迭代之间改变优先级的机会对使项目投资回报最大化是大有益处的。

9. 敏捷开发中对文档的要求

有些人认为敏捷不需要文档，甚至不支持任何形式的文档化，这是一个误解。

回想一下水晶方法是如何处理文档：

水晶方法把开发看作是一系列的协作游戏，而写文档的目标就是只要能帮助团队在下一个游戏中取得胜利就行了。水晶方法的工作产品包括用例、风险列表、迭代计划、核心领域模型，以及记录了一些选择结果的设计注释。水晶方法也为这些产品定义了相应的角色。然而，值得注意的是，这些文档没有模板，描述也可不拘小节，但其目标一定要清晰，那就是满足下次游戏就可以了。我总是将这些思想以下面的方式向我的团队成员表达：通过它们，你只要了解你明天加入这个团队所要知道的内容就行了。

而极限编程是如何对待文档的呢？

与水晶方法相比，XP 认为在团队内外，文档都不必太多，并把它看作是需要交付给客户并由客户付费的故事。我觉得这样做的目标就是通过与其它特性集合进行对照评估，来减少多余的文档数量。XP 更依赖于开发人员之间的直接交流来传递相关的知识而不是依靠写好的文档，而结对编程使其成为可能：因为通过结对，你就可以和其它人分享对系统的理解，而消灭"死角"，也就很少需要文档来记录这些知识了。另外，代码与测试也被看作是描述软件实现细节的文档，它没有更新不及时的问题.....总而言之，就是团队必需的东西或者客户想要的东西。

敏捷宣言指出有价值且可工作的软件胜于详尽的文档。敏捷宣言是一套基本原则和标准，用于检验某个过程是否敏捷，而不是一个具体的方法论。

要理解这个警示，必须记住非敏捷方法论常有文档驱动开发的特征，因为在写代码之前，它需要以大量的文档作为输入。很多情况下，软件开发团队执行相应的过程步骤，只是因为方法论要求他们这样做，尽管它们几乎没有什么价值。结果，很多开发团队完全放弃了这些方法论。敏捷试图避免漫无目标的文档产物，而再次把焦点放在软件开发过程的关键产物上，即代码。不幸的是，很多人没有认识到应该抛弃什么，从而使那种即兴而为式或纸上谈兵式的有限文档成了目标，这些人基本上没有对瀑布式过程（如 SSADM）进行完整的实践。

10.敏捷开发与传统软件工程的区别和联系

那么敏捷开发是全新理论吗？答案莫衷一是。笔者认为，敏捷开发其实借鉴了大量软件工程中的方法。例如迭代与增量开发，这两种在任何一本软件工程教材中都会被提到的方法，在敏捷开发模式中扮演了很重要的角色。

改善，而非创新。敏捷开发可理解为在原有软件开发方法基础上的整合—取其精华，去其糟粕。因此敏捷开发继承了不少原有方法的优势。

11.敏捷开发带来的好处

对于业务人员或者软件系统的最终用户来说，敏捷开发带来的好处是显而易见的：敏捷开发能使项目团队在更快地获取投资回报的同时，构建出更高质量的系统。团队一旦进行构建，客户就可以了解其情况，而不必在一开始就"一步到位"。固有的短反馈周期能够迅速提供满足客户真正需求的特性。同时敏捷开发也比传统项目管理方法提供了更多的管理变更和风险的可选方案。此外，它们还允许项目团队以组织——客户、用户和其他部门——能够真正看见、评估和使用的方式，展现他们的创造力和解决问题的能力。

12.敏捷在软件开发实践中的定位

敏捷的核心是什么？敏捷给软件企业（以及软件开发者个人）带来的好处究竟在哪里？这个问题有很多不同的答案。例如"重视个人和交流"，软件开发者喜欢这样的态度，这是毫无疑问的。例如"重视可工作的软件"，它的价值是显而易见的。但在这一切的背后，敏捷的核心是什么？时下流行的观点是：敏捷就是软件行业里的精益（lean）生产，它的核心是消除浪费。ThoughtWorks 中国公司的高层在近日接受采访时明确指出了这一点。

首先考虑质量问题。一些软件企业为了降低成本而忽视质量，但质量低下的软件会造成返工的浪费，反而提高成本。相反，在日常工作中投入更多的精力来保证质量，反而能够为企业节约成本。ThoughtWorks 中国公司技术总监 Michael Robinson 用软件工程的经典理论来分析这个问题：

任何一本软件工程教材都会告诉你：假设在分析阶段找到并解 决一个错误的成本为 1，在设计阶段解决同一个错误的成本就变成 10，在实现阶段就变成 100，在维护阶段就变成 1000。敏捷软件开发中的众多实践正是为了避免低质量和返工的浪费。尽管它们一开始看起来似乎有些麻烦，但它们带来的收益是实实在在的。

另一种常见的浪费则是"为将来准备的投资"。例如为了应付将来可能出现的需求变化而提前引入的灵活设计，如果需求没有发生变化，这些灵活设计就会成为浪费：不仅浪费了将它设计出来的成本，而且浪费了继续维护它的成本。制造业为了降低库存成本而创造出"Just In Time"的生产和决策方法，ThoughtWorks 中国公司总经理郭晓认为这些方法同样适用于软件行业：

如何消除预测错误的浪费？避免预测错误的 根本办法就是推迟决策：决策下得越晚，就越不容易因为预测失准而造成浪费。当然也不能晚到错过了时机、耽误了工作才下决策，这就像丰田制造的 Just In Time，决策也要 Just In Time。过早的、含有太多预测成分的决策也会造成浪费，其危害丝毫不亚于过晚的决策。

敏捷的、精益的、实用主义的决策往往是符合中庸之道的：它们往往是各种因素、选择权衡之后的结果。敏捷方法极端重视提升客户价值，为了达到这个目标而采取的手段通常都不可能是极端的。

中庸之道常常有效的深层原因是边际效用递减律：对一个方面的东西重视到一定程度以后，再加入更多的重视，收到的边际效用递减；同样的重视度放到另一个方面上，能够收到更大的边际效用。让每一分投入收到最大的回报，尽可能地消除浪费，这是精益的追求。

设计方案的选择说到底应该是一次成本与收益的计算，而不是个人审美取向的衡量——当然，优秀的程序员能够把这种计算变成本能，笔者认为这就是"软件开发的艺术"所在。敏捷方法强调"简单设计"，同样是经过计算的结果。

在面对一个复杂并且灵活的设计时，首先要衡量的不是实现它的收益，而是"现在实现它"与"将来实现它"之间成本的差额。不论一个灵活的设计的收益和成本如何，只要这个差额非常小——等到未来实现它也没有什么额外的困难，就应该毫不犹豫地推迟决策，等到真正需

要的时候再引入灵活的设计。感谢现代化的 IDEs，很多时候我们讨论的这个成本差额确实非常小，这是敏捷设计通常取简单方案的原因所在。

值得注意的是，随时进行这种成本与收益的计算并不是一件易如反掌的事。计算本身也有成本。这是最佳实践和工具支持存在的意义所在：你可以用较低的成本得到前人积累的知识。

13. 敏捷开发方法要避免的过程设计的几个常见错误

- ✧ 对所有的项目使用同一种过程
- ✧ 没有弹性
- ✧ 过于沉重
- ✧ 增加不必要的"必须完成"（"should do" is really should？）
- ✧ 没有经过实践检验

14. 敏捷开发方法过程设计的几个原理

1 、交互的面对面的交流是代价最小，最迅速的交换信息的方法

2 、超过实际需要的过程是浪费的

3 、大的团队需要重量级方法

4 、处理重大问题的项目需要重量级方法强调

5 、增加反馈和交流可以减少中间产品和文档的需求

6 、轻量级方法更强调理解（understanding）,自律（discipline）和技能(skill)，重量级方法更强调文档（documentation），过程（process）和正式（formality）

understanding 指整个团队关于项目的全部知识，包括讨论的过程，documentation 只能记录其中的一部分

discipline 是指个人主动的完成工作，process 指个人根据指令完成工作

skill 指具有良好技能的人可以省略中间的产品，formality 指必须按照规定步骤完成工作

7 、确定开发中间的瓶径，提高它的效率

对于瓶径处的工作应该尽量加快，减少重复，（使用更熟练的人，使用更多的人，使用更好的工具，使瓶径处的工作的深入尽量稳定）对于非瓶径处的工作可以多一些重复，在输入还不确定的情况下也可以尽早开始。

这些原理的几个结论：

1 、向一个项目增加人员要花费较大代价（constly），因为原有人员和新人员之间的交流要花费大量时间

2 、团队的规模经常是跳跃的，例子：需要 6 个熟练的程序员，但是只有 4 个，于是增加不熟练的程序员，结果团队的大量时间花费在培训不熟练的程序员上面，最后增加到了 20 个不熟练的程序员。

3 、应该侧重于提高团队的技能而不是扩充团队

4 、对不同的项目使用不同的过程

5、 在适用的条件下，轻量级的方法优于重量级的方法

6 、对不同的项目要裁减过程

敏捷开发方法的原则是“刚刚好”（Light and Sufficient）

15. 敏捷开发方法背后的思想

Alistair Cockburn 在 Agile Software Development 中讲述了敏捷开发方法背后的思想

人们掌握过程（process）可以分为 3 个阶段：

1 、following 遵循一个定义好的 process

2 、detaching 知道不同 process 的适用范围，在不同的场合使用不同的 process

3 、fluent 不关心是否遵循特定的 process，知道在什么情况下采用什么动作

软件开发是一个充满发明和交流的协作性游戏(cooperative game of invertion and communication)。软件开发的首要目标是生产出软件，遵循特定的过程和模型只是手段，

只要传递了足够的信息，手段是次要的。交流的效果要远远重于交流的形式（Effect of communication is more important than the form of communication）。
一般软件开发有两个目标：1 尽快的生产出软件 2 为下一个 team 或项目做准备，有时这两个目标是矛盾的，我们要在这两个目标之间寻求平衡

在软件开发中，人的因素要远远大于过程和技术。人是有缺陷的：
1 容易犯错误，因此必须在错误扩散之前找到并改正错误
2 当觉得可能失去较多的时候，不愿意冒险
3 重新构造而不愿意重复使用已有的东西
4 难于坚持一个习惯

针对个人因素的几个建议：
1 具体的模型较抽象的模型更容易理解
2 从一个例子开始是容易的
3 通过观察他人的成果学习
4 要有足够的不受打扰的时间
5 分配的工作要与个人意向，能力匹配
6 不正确的奖励会有坏作用，从长期看个人兴趣比奖励更重要，培养在工作中的自豪感：
1) pride in work 参与工作的自豪感，通常参与一个重要的工作会有自豪感
2) pride in accomplishment 完成工作的自豪感，长期未完的工作会使士气低落
3)pride in contribution 为他人贡献的自豪感
7 鼓励关心其他人的工作和整体的工作

在一个团队之间，交流是最重要的，实践证明面对面的实时的交流是最有效的，对交流的延误会损失信息，白板是最好的交流工具，交流工具的先进并不能提高交流效果。文档的作用是记录和备忘，不能通过文档交流。

16.其他

人高于过程，过程的影响只是第二位的。只有人，它们的情感、品质和沟通能力才更有影响力。