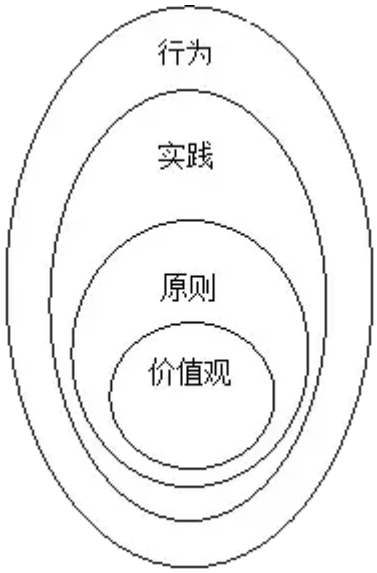
XP的极限编程（eXtreme Programming）

XP是一种轻量（敏捷）、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。与其他方法论相比，其最大的不同在于：

- 在更短的周期内，更早地提供具体、持续的反馈信息。
- 在迭代的进行计划编制，首先在最开始迅速生成一个总体计划，然后在整个项目开发过程中不断的发展它。
- 依赖于自动测试程序来监控开发进度，并及早地捕获缺陷。
- 依赖于口头交流、测试和源程序进行沟通。
- 倡导持续的演化式设计。
- 依赖于开发团队内部的紧密协作。
- 尽可能达到程序员短期利益和项目长期利益的平衡。

XP由价值观、原则、实践和行为四个部分组成，它们彼此相互依赖、关联， 并通过行为贯穿于整个生命期。



XP的五大价值观

沟通（Communication）、简单（Simplicity）、反馈（Feedback）、勇气（Courage）四大价值观，它们是XP的基础。

1. 沟通

XP方法论认为，如果小组成员之间无法做到持续的、无间断的交流，那么协作就无从谈起，从这个角度能够发现，通过文档、报表等人工制品进行交流面临巨大的局限性。因此，XP组合了诸如对编程这样的最佳实践，鼓励大家进行口头交流，通过交流解决问题，提高效率。

2. 简单

XP方法论提倡在工作中秉承“够用就好”的思路，也就是尽量地简单化，只要今天够用就行，不考虑明天会发现的新问题。

- 开发小组在开发时所做的规划，并无法保证其符合客户需要的，因此做的大部分工作都将落空，使得开发过程中重复的、没有必要的工作增加，导致整体效率降低。
- 另外，在XP中提倡时刻对代码进行重构，一直保持其良好的结构与可扩展性。也就是说，可扩展性和为明天设计并不是同一个概念，XP是反对为明天考虑而工作，并不是说代码要失去扩展性

而且简单和沟通之间还有一种相互支持的关系。当一个团队之间，沟通的越多，那么就越容易明白哪些工作需要做，哪些工作不需要做。另一方面，系统越简单，需要沟通的内容也就越少，沟通也将更加全面。

3. 反馈

在XP方法论中，通过持续、明确的反馈来暴露软件状态的问题。具体而言就是：

- 在开发团队内部，通过提前编写单元测试代码，时时反馈代码的问题与进展。
- 在开发过程中，还应该加强集成工作，做到持续集成，使得每一次增量都是一个可执行的工作版本，也就是逐渐是软件长大，整个过程中，应该通过向客户和管理层演示这些可运行的版本，以便及早地反馈，及早地发现问题。

同时，我们也会发现反馈与沟通也有着良好的配合，及时和良好的反馈有助于沟通。而简单的系统更有利于测试盒反馈。

4. 勇气

需要你有勇气来面对快速开发，面对可能的重新开发。勇气可以来源于沟通，因为它使得高风险、高回报的试验成为可能；勇气可以来源于简单，因为面对简单的系统，更容易鼓起勇气；勇气可以来源于反馈，因为你可以及时获得每一步前进的状态（自动测试），会使得你更勇于重构代码。

5个原则

1. 快速反馈

及时地、快速地获取反馈，并将所学到的知识尽快地投入到系统中去。也就是指开发人员应该通过较短的反馈循环迅速地了解现在的产品是否满足了客户的需求。这也是对反馈这一价值观的进一步补充。

2. 简单性假设

类似地，简单性假设原则是对简单这一价值观的进一步补充。这一原则要求开发人员将每个问题都看得十分容易解决，也就是说只为本次迭代考虑，不去想未来可能需要什么，相信具有将来必要时增加系统复杂性的能力，也就是号召大家出色地完成今天的任务。

3. 逐步修改

就像开车打方向盘一样，不要一次做出很大的改变，那样将会使得可控性变差，更适合的方法是进行微调。而在软件开发中，这样的道理同样适用，任何问题都应该通过一系列能够带来差异的微小改动来解决。

4. 提倡更改

在软件开发过程中，最好的办法是在解决最重要问题时，保留最多选项的那个。也就是说，尽量为下一次修改做好准备。

5. 优质工作

在实践中，经常看到许多开发人员喜欢将一些细小的问题留待后面解决。例如，界面的按钮有一些不平整，由于不影响使用就先不管；某一两个成员函数暂时没用就不先写等。这就是一种工作拖泥带水的现象，这样的坏习惯一旦养成，必然使得代码质量大打折扣。

而在XP方法论中，贯彻的是“小步快走”的开发原则，因此工作质量决不可打折扣，通常采用测试先行的编码方式来提供支持。

13个最佳实践

在XP中，集成了13个最佳实践。其主要创新点在于提供一种良好的思路，将这些最佳实践结合在一起，并且确保尽可能彻底地执行它们，使得它们能够在最大程度上相互支持。

1. 计划游戏

计划游戏的主要思想就是先快速地制定一份概要的计划，然后随着项目细节的不断清晰，再逐步完善这份计划。计划游戏产生的结果是一套用户故事及后续的一两次迭代的概要计划。

“客户负责业务决策，开发团队负责技术决策”是计划游戏获得成功的前提条件。也就是说，系统的范围、下一次迭代的发布时间、用户故事的优先级应该由客户决定；而每个用户故事所需的开发时间、不同技术的成本、如何组建团队、每个用户故事的风险，以及具体的开发顺序应该有开发团队决定。

- 客户编写故事：由客户谈论系统应该完成什么功能，然后用通俗的自然语言，使用自己的语汇，将其写在卡片上，这也就是用户故事。
- 开发人员进行估算：首先客户按优先级将用户故事分成必须要有、希望有、如果有更好三类，然后开发人员对每个用户故事进行估算，先从高优先级开始估算。如果在估算的时候，感到有一些故事太大，不容易进行估算，或者是估算的结果超过2人/周，那么就应该对其进行分解，拆成2个或者多个小故事。
- 确定迭代的周期：接下来就是确定本次迭代的时间周期，这可以根据实际的情况进行确定，不过最佳的迭代周期是2~3周。有了迭代的时间之后，再结合参与的开发人数，算出可以完成的工作量总数。然后根据估算的结果，与客户协商，挑出时间上够、优先级合适的用户故事组合，形成计划。

2. 小型发布

XP方法论秉承的是“持续集成，小步快走”的哲学，也就是说每一次发布的版本应该尽可能的小，当然前提条件是每个版本有足够的商业价值，值得发布。

由于小型发布可以使得集成更频繁，客户获得的中间结果也越频繁，反馈也就越频繁，客户就能够实时地了解项目的进展情况，从而提出更多的意见，以便在下一次迭代中计划进去。以实现更高的客户满意度。

3. 隐喻

- 寻求共识：也就是鼓励开发人员在寻求问题共识时，可以借用一些沟通双方都比较熟悉的事物来做类比，从而帮助大家更好地理解解决方案的关键结构，也就是更好地理解系统是什么、能做什么。
- 发明共享词汇：通过隐喻，有助于提出一个用来表示对象、对象间的关系通用名称。例如，策略模式（用来表示可以实现多种不同策略的设计模式）、工厂模式（用来表示可以按需“生产”出所需类得设计模式）等。
- 创新的武器：有的时候，可以借助其他东西来找到解决问题的新途径。
- 描述体系结构：体系结构是比较抽象的，引入隐喻能够大大减轻理解的复杂度。例如管道体系结构就是指两个构件之间通过一条传递消息的“管道”进行通信。

4. 简单设计

强调简单设计的价值观，引出了简单性假设原则，落到实处就是“简单设计”实践。XP的简单设计实践并不是要忽略设计，而且认为设计不应该在编码之前一次性完成。

Kent Beck概念中简单设计是这样的：

- 能够通过所有的测试程序。
- 没有包括任何重复的代码。
- 清楚地表现了程序员赋予的所有意图。
- 包括尽可能少的类和方法
- 他认为要想保持设计简单的系统，需要具备简单思考的能力，拥有理解代码和修改的勇气，以及为了消除代码的“坏味道”而定期重构的习惯。

5. 测试先行 / 测试驱动开发

测试先行是XP方法论中一个十分重要的最佳实践，并且其中所蕴含的知识与方法也十分丰富。

6. 重构

重构时一种对代码进行改进而不影响功能实现的技术，XP需要开发人员有重构代码的勇气。重构的目的是降低变化引发的风险，使得代码优化更加容易。通常重构发生在两种情况之下。

- 实现某个特性之前：尝试改变现有的代码结构，以使得实现新的特性更加容易。
- 实现某个特性之后：检查刚刚写完的代码后，认真检查一下，看是否能够进行简化。

重构技术是对简单性设计的一个良好的补充，也是XP中重视“优质工作”的体现，这也是优秀的程序员必备的一项技能。

7. 结对编程

结对编程技术被誉为XP保持工作质量、强调人文主义的一个典型的实践，应用得当还能够使得开发团队之前的协作更加流畅、知识交流与共享更加频繁，团队的稳定性也会更加稳固。

8. 集体代码所有制

由于XP方法论鼓励团队进行结对编程，而且认为结对编程的组合应该动态地搭配，根据任务的不同、专业技能的不同进行最优组合。由于每个人都肯定会遇到不同的代码，所以代码的所有制就不再适合于私有，因为那样会给修改工作带来巨大的不便。

也就是说，团队中的每个成员都拥有对代码进行改进的权利，每个人都拥有全部代码，也都需要对全部代码负责。同时，XP强调代码是谁破坏的（也就是修改后发生问题），就应该由谁来修复。

由于在XP中，有一些与之匹配的最佳实践，因此你并无须担心采用集体代码所有制会让你的代码变得越来越乱：

- 由于在XP项目中，集成工作是一件经常性得工作，因此当有人修改代码而带来了集成的问题，会在很快的时间内被发现。
- 由于每一个类都会有一个测试代码，因此不论谁修改了代码，都需要运行这个测试代码，这样偶然性的破坏发生的概率将很小。
- 由于每一个代码的修改就是通过了结对的两个程序员共同思考，因此通常做出的修改都是对系统有益的。
- 由于大家都坚持了相同的编码标准，因此代码的可读性、可修改性都会比较好，而且还能够避免由于命名法、缩进等小问题引发经常性得代码修改。

集成代码所有制是XP与其他敏捷方法的一个较大不同，也是从另一个侧面体现了XP中蕴含的很深厚的编码情节。

9. 持续集成

小型发布是指在开发周期经常发布中间版本，而持续集成的含义则是要求XP团队每天尽可能多次地做代码集成，每次都在确保系统运行的单元测试通过之后进行。这样，就可以及早地暴露、消除由于重构、集体代码所有制所引入的错误，从而减少解决问题的痛苦

10. 每周工作40小时 / 可持续的速度

Kent Beck认为开发人员即使能够工作更长的时间，他们也不该这样做，因为这样做会使他们更容易厌倦编程工作，从而产生一些影响他们效能的其他问题。因此，每周工作40小时是一种顺势行为，是一种规律。

11. 现场客户

为了保证开发出来的结果与客户的预想接近，XP方法论认为最重要的需要将客户请到开发现场。就像计划游戏中提到过的，在XP项目中，应该时刻保证客户负责业务决策，开发团队负责技术决策。因此，在项目中有客户在现场明确用户故事，并做出相应的业务决策，对于XP项目而言有着十分重要的意义。

12. 编码标准

不管是代表重型方法论的RUP，PSP，还是代表敏捷方法论的XP，都认为开发团队应该拥有一个编码标准。XP方法论认为拥有编码标准可以避免团队在一些与开发进度无关的细节问题上发生争论，而且会给重构、结对编程带来很大麻烦。

13. 配合是关键

有句经典名言“1+1>2”最适合表达XP的观点，Kent Beck认为XP方法论的最大价值在于在项目中融会贯通地运用12个最佳实践，而非单独地使用。当然可以使用其中的一些实践，但这并不意味着你就运用了XP方法论。XP方法论真正能够发挥其效能，就必须完整地运用12个实践

敏捷开发中XP与SCRUM的区别

前面说了敏捷它是一种指导思想或开发方式，但是它没有明确告诉我们到底采用什么样的流程进行开发，而Scrum和XP就是敏捷开发的具体方式了，你可以采用Scrum方式也可以采用XP方式；

区别之一： 迭代长度的不同

XP的一个Sprint的迭代长度大致为1~2周, 而Scrum的迭代长度一般为 2~ 4周.

区别之二: 在迭代中, 是否允许修改需求

XP在一个迭代中，如果一个User Story(用户素材, 也就是一个需求)还没有实现， 则可以考虑用另外的需求将其替换，替换的原则是需求实现的时间量是相等的。 而Scrum是不允许这样做的，一旦迭代开工会完毕, 任何需求都不允许添加进来，并有Scrum Master严格把关，不允许开发团队受到干扰

区别之三: 在迭代中，User Story是否严格按照优先级别来实现

XP是务必要遵守优先级别的。 但Scrum在这点做得很灵活， 可以不按照优先级别来做，Scrum这样处理的理由是：如果优先问题的解决者，由于其它事情耽搁，不能认领任务，那么整个进度就耽误了。 另外一个原因是，如果按优先级排序的User Story #6和#10，虽然#6优先级高，但是如果#6的实现要依赖于#10，则不得不优先做#10.

区别之四:软件的实施过程中，是否采用严格的工程方法，保证进度或者质量

Scrum没有对软件的整个实施过程开出工程实践的处方。要求开发者自觉保证，但XP对整个流程方法定义非常严格，规定需要采用TDD, 自动测试， 结对编程，简单设计，重构等约束团队的行为。因此，原作者认为，这点上，XP的做法值得认同的，但是却把敏捷带入了一个让人困惑的矛盾, 因为xp的理念，结合敏捷模式，表达给团队的信息是“你是一个完全自我管理的组织， 但你必须要实现TDD, 结对编程, ...等等”

不难发现，这四个区别显见的是： Scrum非常突出Self-Orgnization, XP注重强有力的工程实践约束，即Scrum偏重于过程，XP则偏重于实践，但是实际中，两者是结合一起应用的。所以建议在管理模式上启用Scrum， 而在实践中，创造一个适合自己项目组的XP。