本立而道生，追溯敏捷开发最佳实践之源

撰文 / 莫映

敏捷开发的悄然兴起

就在几年前，敏捷开发作为一种软件开发的方法学，自国外悄然传入了国内。也许现在已经很难再精确找寻最初的时间起点了，不过在笔者的记忆当中，**2002** 年在《程序员》杂志上刊载的一篇由林星撰写的介绍敏捷方法的文章，以及包括**"**新方法学**"**在内的由 **Martin Fowler** 先生所撰写的一系列文章的中译版本在国内网络媒体上的出现，大概可以算作是敏捷开发在国内籍由传统媒体与现代媒体为普通大众所了解和认识的一个开端。

敏捷开发以轻量和反正统的姿态显身，刮起了一股清新的微风。它拒绝繁文缛节，强调对变化的快速适应能力，强调人在开发过程中所起的重要作用。套用一下 **Gabriel Tarde** 的创新扩散理论[1]，这样的观点很快就赢得了不少早期适用者（**early adopter**），获得了他们坚定不移的支持和拥护。想必读者很容易就能猜到，这群早期适用者的主体，便是一直工作于国内开发一线的程序员们。

就如同早先敏捷开发在西方的传播方式一样，凭借着一群早期适用者们自发的言传身教，诸多所谓的敏捷最佳实践，得以为更多的人所听闻。尽管这种传布方式带有明显的经验主义意味，但是作为一项实践活动，软件开发原本就不是形而上的理论空谈，因此从现实的角度出发，这群实践者们对敏捷开发在国内的传播所起的作用，是功不可没的。

日历翻到了 **2007** 年，我们可喜的发现，敏捷开发已经不再是阳春白雪，少数人的游戏了。不少软件开发团体，无论是大公司还是小公司，都在或多或少的进行着各种各样的敏捷实践。他们将敏捷的最佳实践应用于软件开发活动，并从中获得了益处，敏捷开发还在影响着更多的 **IT** 从业人员。或许可以毫不夸张的说，国内的敏捷开发已经步入了一个新的发展阶段。

质疑的声音

任何事物的发展都不是一帆风顺的，敏捷开发也不例外。一方面，常识和惯性思维会对人们接受新事物构成障碍，因此我们时常听到一些对敏捷的质疑声音，也就不足为奇了；另一方面，作为早期适用者的部分程序员们，在传播敏捷的过程中，也有其自身的局限性，从而导致对敏捷的片面认识与夸大，这也更加促成了质疑的合理性。

一些人认为，以极限编程为代表的敏捷方法对人有很高的要求，开发者需要具备相当的素质，才能驾驭极限编程中像测试驱动开发、重构这样的高阶敏捷实践，否则，开发效率只会更低，项目风险只会更大。另一些人认为，敏捷方法过于强调对变化的适应，它采用一个一个零散的用户故事来捕获用户的需求，致使对需求的把控无法做到全面而系统，最后只能是来一个需求实现一个需求，太过盲目。还有一些人认为，极限编程中所倡导的通过一步一步重构的方式来演化最终的系统架构的方法是不现实的，因为好的架构是没有办法重构出来的，而是必须要经过先期的系统

[1]这一理论解释了创新事物在不同时间段为不同类型的人群所接受的过程，见：
http://en.wikipedia.org/wiki/Diffusion_of_innovations

分析与建模设计，才能保证有一个稳定的架构基础，否则，频繁的重构反而会导致架构的频繁变化，甚至是推倒重来，这会造成软件开发的浪费。凡此种种，不一而足。

寻找敏捷开发的源头

要想以一篇文章的篇幅来逐一应对如上的每一条质疑，恐怕是不够的，这也并非本文所要表达的主旨。以笔者的愚见，如果要想一劳永逸的释疑，就不能纠缠于细枝末节，而是必须要追本溯源，搞清楚敏捷开发到底是为了什么。只有明晰了敏捷开发的目的，才能更好的理解那些所谓的最佳实践，明白它们如何能够保证项目的成功，进而更好的加以有效运用，而不是误用。那么，敏捷开发中的诸多最佳实践到底是为了提高开发效率？加强团队沟通？提升代码质量？还是别的什么目的呢？

对于这一问题，或许我们可以从 **Martin Fowler** 先生的**"**新方法学**"**[2]一文中获得启示。在文中，**Martin Fowler** 对众多敏捷方法做了概括，提出了所有敏捷方法共有的显著特征，其中之一便是适应性（**Adaptive**）。所谓适应性，就是不求在很长的时间跨度内做出详细的计划，然后一成不变的遵照执行，而是通过迭代的过程逐步获得反馈，以适应不断变化的需求。如今，国内的 **IT** 业已经从朝阳产业逐步走向成熟，一方面是客户对软件系统要求的逐渐提升，另一方面则是来自同行激烈的市场竞争，所有这些都对软件开发提出了更高的要求。软件团队必须以更短的时间将系统迅速推向市场，并以更灵活的手段与更低廉的成本来应对来自客户源源不断的需求挑战。在这样的背景之下，以往以瀑布模型为主的开发方法就会很难适应。因为，瀑布模型所强调的，是软件开发周期中界限分明的各个阶段，以及作为每个阶段验收成果的大量文档，需求是在前期即被明确界定的，变化的需求很难在后期被引入（或者说引入的代价非常高昂），人们需要投入大量的精力来编制一系列很容易过时的文档。为了解决这一矛盾，将原本固定不变的软件开发全周期划分为一系列小的迭代周期似乎是一个很自然的推理。如此一来，一方面软件团队可以在很短时间内将软件系统的部分功能呈现给最终用户，另一方面，通过来自客户的即时反馈，团队也能在每个迭代周期内审时度势的规划本次迭代的开发内容，这便是以迭代反馈为特征的敏捷方法在**"**适应性**"**方面的优势体现。一言以蔽之，快速多变的业务与市场向软件开发提出了适应性要求，而敏捷方法则以迭代反馈的手段来满足适应性要求**--**敏捷开发的最终驱动力来自于外部环境，它是在用技术的手段来应对业务的不断变化。这便是敏捷开发的源头所在。

此外，作为对上述观点的补充说明，敏捷方法的另一条重要原则不得不被提及，那就是：将决策推迟到职责要求的最后一刻。有关这一观点的论述，可以在 **Mary Poppendieck** 和 **Tom Poppendieck** 所著的《精益软件开发》一书中找到，有意思的是，在汽车制造行业里，有一个活生生的例子印证了这一原则的正确性，那就是以大野内一先生为首的丰田公司所创立的精益制造理论。在丰田公司的汽车生产线上，一直要等到客户买下一辆车之后才开始这辆汽车的生产，这可以说是推迟决策的典型范例。当我们不具备足够的能力对眼前的状况做出决策时，不妨就将决策向后推迟，直到有足够的信息能够让我们做出判断。这一方面可以降低当前工作的难度和成本，另一方面也可以消除软件开发中的资源浪费现象。要知道，做出一个超前的决策是有着相当难度的，这往往对决策者有着更高的要求。多年以前，我曾经亲眼见识过日本同行们带给我们的**"**视觉冲击**"**，厚厚一大摞需求文档，洋洋数百页分析设计。乍一看这似乎是让人觉得不可思议的事情，可是仔细想来却也不乏合理性：这需要文档的编写者熟谙业务领域的知识，准确把握需求的脉搏，进而有着深厚的设计功底。如果从这一角度出发，敏捷方法所提倡的推迟决策原则，到是可以让你快速入手，从而降低了技术的门槛，增加了成功的机会，这尤其适合当今快速多变的市场，以及现代人所习惯的快餐式文化。有趣的是，从某种意义上而言，这看起来似乎与前述有

[2]参见 http://www.martinfowler.com/articles/newMethodology.html

关"敏捷方法对人的要求很高"的质疑是相悖的。另一方面，软件开发中的资源浪费也是一个普遍存在的现象，人们常常在讨论过度的设计，开发无用的功能，编写过时的文档，这些都是资源浪费的具体体现。笔者也曾编写过从来都不曾被使用过的代码，也许作为技术人员而言，全凭一时喜好而迫不及待的去实现一个不曾被市场充分证明的功能是可以理解的，那么作为团队的管理者，漠视软件成本的无端浪费，则是一件很遗憾的事情。让软件资源在当前阶段发挥出最大的价值应该是有经营头脑的管理者们所希望的，而将决策推迟，则有助于减少盲目性，对资源的分配做出正确的判断，从而让价值最大化。

推迟决策与迭代反馈有着密切的关联性。需求的变化与市场的竞争，要求软件团队节约成本，合理分配资源，以达到价值的最大化，消除不必要的浪费。而迭代反馈则恰好是实现这一目的的有效手段，从一点开始快速入手，逐步迭代求精，每次迭代都将获得针对当前工作的反馈信息，团队根据这些反馈信息来制订下一步该做什么的决策，于是，更多深思熟虑的决策被推迟到后面，它们的依据便是每次迭代的实际反馈--这便是同一枚硬币的正反两面。

最佳实践并非无源之水

推迟决策与迭代反馈，作为"硬币"的两面，贯穿于敏捷开发的始终。而敏捷方法中所给出的众多最佳实践，便是保证我们能够有效达成这两个方面的具体措施。这便是最佳实践存在的根本价值。如果我们仔细审视一下这些最佳实践，便会发现，很多实践都或多或少的体现着推迟决策与迭代反馈的思想。理解了这一点，最佳实践就不会成为无源之水，就能避免生搬硬套，做到灵活运用。

测试驱动开发

测试驱动开发是备受众多技术人员所推崇的一项敏捷实践。"测试、编码、重构"的不断循环，贯穿于开发过程的始终。从迭代反馈的角度而言，测试驱动开发构成了敏捷开发中最为基本的负反馈环，每一次从测试到重构的循环即是一个小的迭代。测试代码在第一时间为保证功能代码的正确性提供了反馈信息，而重构只有在以充分测试作为反馈的前提之下才能有效的进行。以测试驱动的方式逐步构建而成的完整测试用例集，就是一整套能够精确而及时的反映系统当前健康状况的负反馈系统。此外，测试驱动开发的特别之处还在于，它作为一种软件设计的手段，免去了对早期设计的承诺，而是将设计融入到了开发过程之中，或者以推迟决策的观点来表达，就是将针对设计的决策推迟到了编码开始之后。开发者可以通过在测试代码编写与令测试通过的功能代码的编写过程中获得足够的反馈，以做出正确的设计决策。

用户故事

用户故事是敏捷方法中最为常用的需求捕获手段与表达方式。对于用户故事的编写，尽管有所谓的**"INVEST"**原则作为指导[3]，但总体而言，用户故事的叙述并不要求十分精确，书写也较为随意，它强调的是以简单的表达形式在第一时间捕获到客户所关心的商业价值，因此它被形象的称为用户需求的占位符（**placeholder**）。用户故事在形式上的随意性，与一般意义上的用例有着显著的区别。不过，这恰好就是敏捷方法在需求捕获过程中推迟决策的一种体现：避免早期对需求的盲目承诺，而是将需求的详细分析推迟到开始实现之前，包括整个迭代开发期间，甚至是后期，这样可以有效的避免因为对需求的错误认识而导致的不必要的资源浪费。此外，用户故事作为一种可以被量化跟踪的需求点，也是迭代开发中天然的开发单元。一个迭代周期往往被描述为

[3]由 William C. Wake 提出，参见：http://xp123.com/xplor/xp0308/index.shtml

完成若干用户故事的过程。对用户故事的状态跟踪（这些状态包括：待开发，开发中，待测试，测试中，已完成等等），可以为当前的开发状况提供有效的反馈信息。这便是用户故事在迭代反馈方面所体现的价值。

结对编程

以往，人们对结对编程是否会导致开发效率低下有所疑虑，以笔者的愚见，一部分原因恐怕在于对如何有效的进行结对编程缺乏了解。有效的结对编程需要结对双方共同的努力。在结对过程中，操纵键盘和鼠标的开发者好比是把握汽车方向盘的驾驶员（**driver**），而与之结对的开发者则是引导驾驶员正确行驶的领航员（**navigator**），两个角色交替进行，便构成了结对编程。来自结对者的即时反馈，可以在第一时间一定程度的避免将错误引入当前的代码之中。与此同时，结对的过程也让知识获得了共享。而适时的在团队内进行结对轮换，则可以在更大范围内获得反馈，修正错误并共享经验，其结果是令多数人都对系统有了深入的了解。在以往，这一结果是需要通过专门的 **code review** 环节才能达成的。在推迟决策方面，让团队成员尽可能多的全面了解系统，将有助于推迟由谁来完成特定任务的决定。当因工作的需要或是其他原因而在团队中出现了人员流动的情况时，知识共享将有助于避免项目陷入被动的境地。

其他实践

敏捷开发的最佳实践还有许多，为人们所熟知的还包括：持续集成、每日站立式会议等等。与前述的实践类似，这些实践也或多或少的体现了迭代反馈与推迟决策的思想。持续集成是多数敏捷团队采用的一项实践，无论是每日构建，还是更短周期的构建，其目的都是要在第一时间为团队提供系统在集成方面的状况反馈，将集成所遇到的问题尽早暴露。还有每日的站立式会议，通过每一位团队成员讲述自己昨天与今天的工作，以及所遇到的问题，同样为整个团队提供了项目当前状况的真实反馈。这些反馈都将有助于团队做出正确的决策。

结论

敏捷开发的根本目的在于，以更灵活的手段和更低廉的成本，快速响应多变的客户需求，积极应对激烈的市场竞争。而这其中，迭代反馈与推迟决策作为隐含于敏捷方法背后的原则，始终贯穿了敏捷开发的全过程，指导着众多敏捷实践。由于有了这些原则，那些所谓的敏捷最佳实践便成了有源之水，有据可依。掌握了敏捷方法背后的原则，将有助于我们更好的理解敏捷开发中的诸项实践，避免囫囵吞枣，生搬硬套。并且，更重要的是，敏捷原则可以引导我们脱离书本，并创造更多的最佳实践，进而将我们的视野拓宽至更加广阔的领域。