敏捷的软件开发在软件工程当中的应用

随着软件开发方法的不断演进，混合的开发方法在各软件企业和团队中应用越来越广泛。每一种开发方法都有其优点，如传统的瀑布式方要求有详细的项目计划和文档，部署、QA 测试和交付过程严谨。而敏捷方法的优点则体现在能够快速迭代，更多的强调人员在整个开发过程中所发挥的作用。

有研究机构数据显示，越来越多的开发团队开始采用混合的开发方法。其中，有的团队同时采用 XP、SCRUM 等多种敏捷方法，也有同时采用敏捷和传统相结合的方法，而只采用一种敏捷方法的团队或企业的比例还不足三分之一。

而如果换一个角度，站在客户的立场上呢？也许付钱购买软件的客户会有一些不同的想法。项目 B 从开始之后一个月便交付了第一个可工作的版本，从那时起客户就开始使用这个软件的部分功能，并且不断地把自己使用的感受反馈给开发团队。在真实的业务运营过程中，客户甚至发现了一种新的盈利模式，并进行了一次大规模的业务调整，这次调整的结果也直观地体现在软件项目中。虽然项目 B 的整体交付速率低于项目 A，但它提供的所有功能都是客户真正需要的，它们为客户提供实实在在的价值——更不用说，客户提前好几个月就开始使用这个软件。

实际上，这是一个关于软件价值的讨论。和“成功项目”一样，对于“软件的价值”，不同的人也会有不同的定义。不过作为付钱购买软件的客户，他对于软件价值的定义是一目了然的：他能够从使用软件中创造多少价值，软件能够为他的业务提供多少价值，这就是软件的价值。或者说得更简明一点：软件价值源自使用。

这正是为什么很多客户青睐“项目 B”的原因——这里并不是肯定所有的客户都有同样的观点，稍后笔者会举出反例，但至少支持这一观点的客户不在少数。因为他们处在一个残酷而快速变化的商业环境中：他们的供应商在变化，他们的客户在变化，他们所处的经济环境和政策环境也在变化。这一切的变化迫使他们的业务也要随之变化。今天这个经济全球化的时代是一个“快鱼吃慢鱼”的时代，客户迫切希望新的软件系统为他们带来竞争优势——哪怕这个软件系统尚未完成，只要能够投入使用。最后，客户对于新的软件系统究竟应该是什么样子并没有百分之百的把握，他们的想法往往要在真正使用软件之后才会浮现成型。几方面的因素加在一起，使得这些客户更愿意尽快开始使用软件、提出反馈、并不断完善软件，而不是提出一组需求、然后坐等几个月之后原封不动地拿到这些功能。

想想这个客户会怎么定义一个“成功的软件项目”？也许这个项目超过了预期的时间，也许投入了更多的人力，但这些并不意味着“项目失败”——只是付出更高的成本。关键在于，他投

入的这些成本能够带来多大的收益，他的投资回报率是否划算。对于这个客户而言，如果项目能够随时给他提供可用的、能够创造最大价值的软件，能够随时让这种有价值的想法得以实现，这就是一个成功的项目。

这里所说明的就是一种为客户创造最大化价值的软件开发方法。这样的方法有很多种，但它们有一个共同的特点：尽快、尽可能频繁地交付可以工作的软件，让客户尽快开始使用软件，从使用中创造价值、厘清思路、提出反馈。仍然以 ThoughtWorks 的项目为例，这些项目通常在启动开发阶段之后一个月内就会发布第一个版本，随后每一周或每两周发布一个新版本——每个版本都是一个可以工作的软件，每个版本都比前一个版本具有更丰富的功能，并且每个版本都包含客户认为迄今为止最有价值的那些功能。 “开发下一个版本”的过程叫做“迭代”，这些开发方法最大的共同点就是“迭代式开发”——不是一股脑地交付全部功能，而是每次增加一点、渐进地交付最有价值的功能。

对于很多软件企业而言，项目 A 是一个“理想的”成功项目。那么，是什么让情况变得不那么理想？答案是一个所有软件开发者耳熟能详的词：需求变更。在真实的项目中，客户通常不会等到最后一天再照单全收整个项目，因为他知道自己的业务正在发生变化。这时需求变更就出现了，伴随着来回的扯皮和讨价还价。更糟的是，大量的需求变更发生在项目晚期——因为直到这时客户才真正看到、使用到这个软件，他的很多想法才真正浮现成型。随着这

种“最后一分钟的需求变更”，项目超期、超出预算也就成了家常便饭。能够像项目 A 这样完工交付的，实在是凤毛麟角的幸运儿。

为了对付需求变更这个噩梦，软件开发者们还发明了另一个词：变更控制。这个有趣的词暗示着：需求变更是一种“不好”的东西，是需要“控制”的东西。然而站在客户的角度上想想，他在亲身使用了软件之后提出的要求，难道不是最有价值的东西吗？把这种真正创造业务价值的要求“控制”起来，难道是合理的吗？

在前面提到过，并非所有的客户都一定青睐迭代式开发。那么，哪些软件项目不一定需要迭代式开发呢？从整篇文章的内容不难看出，如果客户的业务绝对不会变化，如果客户的需求巨细靡遗非常明确，如果客户不需要尽快开始使用软件以便收回成本，那么迭代式开发对他的帮助就会小得多。不过，如果读者认真思考的话，这样的例子也许并不多——也许比你最初认为的要少得多。一个很好的例子是“神州六号”火箭使用的计算机控制系统。还有多少这样的例子？读者不妨试着自己想想。

为了让迭代式开发能够成为现实，为了确保尽快、尽可能频繁地交付，为了确保每次交付的都是最有价值的功能，软件开发者、软件企业和客户——需要很多的改变。这里既有职责与权利的划分，也有开发过程和团队的重组，还有技术层面的实践指导。这些正是敏捷方法学所涵盖的内容。缺少了这些东西，“为客户创造最大价值”就只能成为一句空话。

企业架构(EA)和敏捷方法(AM)拥有共同的目标——交付能够跟业务需要对齐的软件，并响应对这些业务需要无可避免的变更。

报告的标题确实说：“是的，但需要付出努力”，所以仍然还有希望。但需要 EA 组和 AM 项目认识到对方有价值的贡献，并在他们的工作中做出适应性调整。

一个曾经使用过其中一种但因为缺乏对另一个的使用而失败了的项目会最大程度拥有使用两者的经验。例如，一个重要的文档处理系统可以使用最好的 AM 实践开发出来，但不能协调好系统的 EA 需要如跨越需求、接口、和操作性问题等。作为选择，一个采用瀑布方式的项目会准备妥当它的所有的企业架构，但是却不能向及早的向客户展现它的价值，或者不能够通过有意义的迭代来解决风险问题。所以，这些 paper 都是来自于经验的，例如：项目是如何因为忽略了其他可行的规程才陷入这种境地的，有效的处理方式是什么等。

一个意义更加深远的案例可能是在项目启动时均衡 EA 和 AM。 然而，这其实非常难，很少发生，主要是因为组织性问题，以及谁在过程的哪个部分被涉及的角度。你会看到很多的失败，例如架构师跟客户（更惨的是在根本没有客户）但没有开发团队参与的情况下整理需求，然后开发团队脱离开架构师进行接管。

Jim Watson 和 Michael Rosen 告诉我们，关于这个专题的范围，SOA 可以被看作是 EA 的一个实例。因此这里所有相关的问题

和解决方案适用于采用了 SOA 并存在 AM 团队的组织(无需惊讶，这与 InfoQ 上的文章 SOA 和敏捷：是朋友？还是敌人？相吻合)

EA 和 AM 的交互并不依赖于 SOA，但值得注意的是 SOA 提供了相互的兴趣和问题以允许进程一起使用 EA 和 AM。例如，想在一个 SOA 主导的项目定义真正有用的业务级别的服务可能具有难度，一个缺乏 AM 开发实践的由 EA 主导的 SOA 会产生许多的 SOA shelfware，因为它很难实现或者仅仅定义出不是真正需要的接口。

一个推荐的方案是， 对一个 AM 团队而言它被当作架构的一个包含部分，作为每个团队的成员与 EA 组进行联络。当被要求阐明推荐 Architect Reloadus 或是 Architect Oryzus（其定义见 Martin Fowler 的 Who Needs an Architect? ）中的哪种架构类型时，Michael Rosen 建议哪种也不采用。在大的组织中会拥有重要的 EA 组，一个典型的 IT 组可能拥有 2000 个员工，500 个架构性的重大项目，在 EA 组中只有 70 个架构师。没有足够的架构时可供应因此 Architect Oryzus 很难应用。Architect Reloadus 同样不能得到应用，因为它们没有可实施的环境。有效的架构师的使用方式是作为一个单独的 AM 团队的咨询顾问，这样，一个来自 EA 组的架构师就可以发挥效用而不是嵌入到团队中。

所以，拥有 EA 组和 AM 团队的组织不必要互相容忍，虽然他们拥有共同的目标，他们的缺省操作模式是不与其它成对的并且

（成对使用通常会）产生问题。因此这些实践等对达成企业的战略目标和交付战术性的软件项目非常有用。

敏捷联盟创始人之一、咨询师兼图书作者 Mike Cohn 最近根据其自身经验将“如何帮助团队采纳敏捷”总结为三对核心模式，当团队向敏捷过渡时，可以利用这些模式。Mike 建议，团队或者组织在逐步采用敏捷的过程中，应该从每对模式中选出一个最适合他们自身情况的模式。

“小步前进”是指最初在一个试航团队中尝试敏捷的转型，然后逐渐推广到整个组织中的方法。Mike 建议，这种方法在以下几个方面具有优势：最小化因错误而导致的成本、将最初成功的可能性最大化、培养内部的“专家”，以协助后期推广过程的顺利进行。Mike 紧接着提及三个隐患：团队在试验阶段产生的早期的成功，可能会给整个组织带来错误的期望；组织推广所用的时间会更长；一旦失败，怀疑者将把其视为公司无法实现承诺的一种信号。

与其相反，“全面推广”的特征是从一开始就让所有团队进行转型，它可以在以下方面让企业受益：展现管理中的各种承诺，组织会变得更加灵活，避免同时使用两个过程带来的不一致，以及减少总体上的抵触感。Mike 同时也指出了“全面推进”的缺点：高风险，高开销，可能需要机构重组，会遇到来自于组织的很大压力。

“技术实践优先”要求团队接受敏捷是从关注 XP 的诸多实践开始的，比如简单设计、测试驱动开发、结对编程、持续集成以及短迭代周期。它带给团队的好处是：转型的启动非常迅速而且平滑。Mike 指出这种方法的不足在于：通常较难做到，而且会导致开销激增，同时还可能将团队带离以用户为中心的思考，从而失去了敏捷的真正意义。

相反，“迭代优先”方法，它最初只关注“团队以迭代方式工作”，一旦这个目标受到阻碍，才着手改变技术实践。它的优势可能在于：它很容易实现，而且遇到团队成员抵触的可能性很小。但也有另一个风险：团队可能永远也不会采用对于改善敏捷而言最基础的工程实践。

“秘密行动”是指团队在采用敏捷实践过程中积累的大量知识只保留在团队的内部。它允许团队在受到其他人关注之前就能获得成功，这就是它给团队带来的好处；那些关注即来自于希望模仿他们的人，也来自于可能会反对他们的人。其缺点包括：难以获得组织所能提供的必要的支持，同时，即使这个团队成功了，也不容易说服怀疑者们去信服。

“公开推广”是指团队在采用敏捷过程中所做的努力对于团队以外甚至组织以外都是公开的知识。它的优势在于：它会激励团队去坚持采用敏捷之路，帮助团队得到外部的支持，更早地发现怀疑者们的疑虑， 并证明高层管理者支持这种变迁并希望它成功。其可能引起的不良后果是，假如公开宣布开始做某件事，最

终却没有成功，别人会认为这是非常鲁莽的，也就是说，此时反对者的质疑声就彻底抵消了这种方法的优势所在，而这正是“公开推广”的劣势

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。

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。