敏捷方法与经典软工的较量

撰文 / 莫映

软件开发的过去与现在：装备与灵活性的钟摆

“在 **IT** 领域，我们正好都在从装备统治一切的时代走出来。现在我们正进入一个唯有灵活性才是至关重要的时代。”**Tom DeMarco** 在为《规划极限编程》一书所做的序中曾将软件开发方法的变迁历史比作人类的军事发展史。在序中，他援引卡尔·冯·克劳塞维茨在《战争论》中的观点，认为军事历史就是一个在装备和灵活性的相对优势之间来回摇摆的钟摆：从古罗马的重装步兵，到中世纪的骑士，再到成吉思汗的轻骑兵，及至一次大战中的坦克与现代战争中的反坦克导弹。每一个新兵种或新武器的诞生，都代表着某种相对性优势：或装备，或灵活性，二者不断的交替轮回。

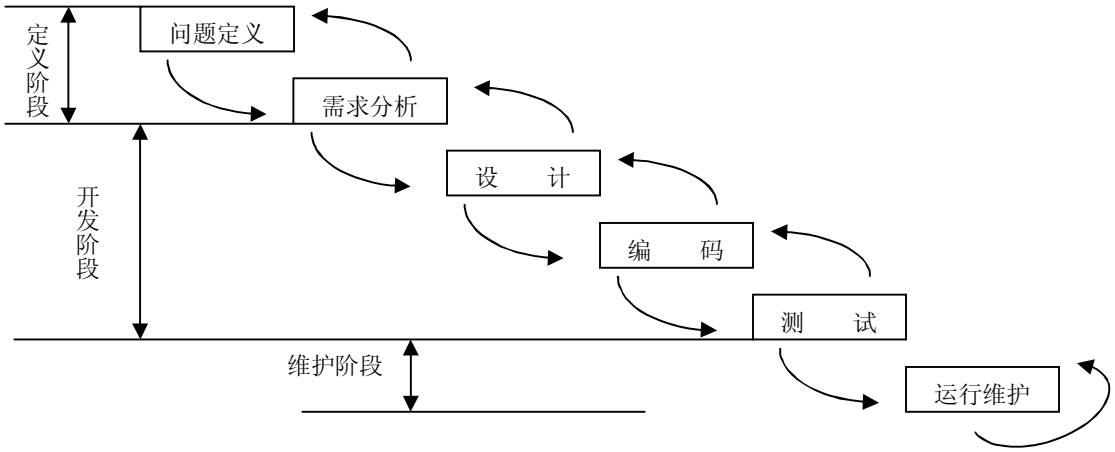
而在仅有短短数十年历史的 **IT** 行业，人们也经历了类似的变迁。早期手工作坊式的软件开发模式铸就了无数个人英雄与他们的传奇故事，此时灵活性的优势彰显无疑。而随着软件复杂度的逐步提升与人们对软件期望值的愈加膨胀，仅凭一己之力的手工作坊已然不再适合。人们迫切需要有方法论的指导，以应对多人协作的开发模式。于是，各种软件开发方法纷至沓来，软件工程作为一门学科，也逐渐进入了平常人的视野，装备的优势初现端倪。然而在经历过从理论到实践的洗礼之后，人们却发现一度被当作济世良药的方法学并未在所有领域都如人所愿：在许多普通的软件企业里，一些在没有方法学指导时就存在的问题，依旧顽固的存在着。为了摆脱困境，人们增加了更多约束行为和思想的条规，并要求提供更多除软件本身以外的东西。这样的做法使软件组织投入了更多的成本，在降低了开发效率的同时，却并未达到预期的效果：进度延期，成本超支，质量低劣，客户关系僵化，团队士气低落……装备的优势地位受到了质疑。在这样一个时代背景之下，强调以灵活性见长的敏捷方法应运而生，其标志便是 **2001** 年敏捷联盟的成立与敏捷宣言的诞生。

敏捷方法的两大焦点：适应性与人

敏捷方法的两大主要特征便是对“适应性”的强调与对“人”的关注。关于这一点，**Martin Fowler** 在他的《新方法学》[1]一文中有过专门的论述。

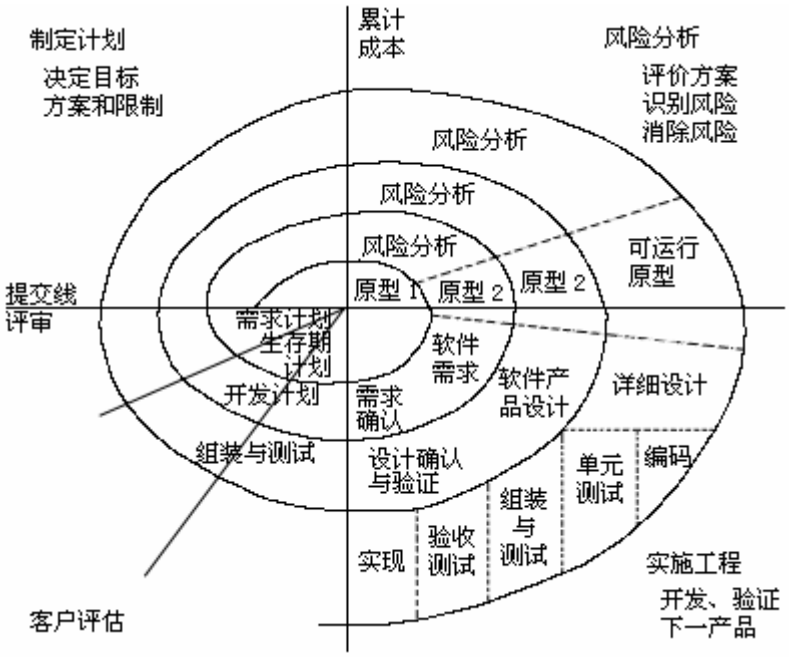
经典的软件工程方法借鉴了建筑工程领域的实践，它强调前期的设计与规划，并尝试在很长时间跨度内为一个软件开发项目制定严格而详尽的计划，然后交由具备普通技能的人群分阶段依序达成目标。这方面的代表就是早在 **1970** 由 **Winston Royce** 提出的“瀑布模型”，直到上个世纪 **80** 年代早期，瀑布模型一直都是唯一被广泛采用的软件开发模型。但是这样的做法在面对变化的需求与外部环境时，却并不能发挥其应有的效力：由于前后相继的线性开发模式使得人们只有在每个阶段的末尾才能见到成熟的制品（包括代码、文档，及其他），因此一旦产生变更，尤其是在软件生命周期的后期，就会导致严重的后果。与此同时，在我们所处的环境中，很多时候需求的变更都属于常态。其中的原因有很多，例如：快速多变的技术潮流与市场环境的作用，亦或是客户自身也未必清楚自己到底需要什么，这是很多软件企业面临的现实问题。

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



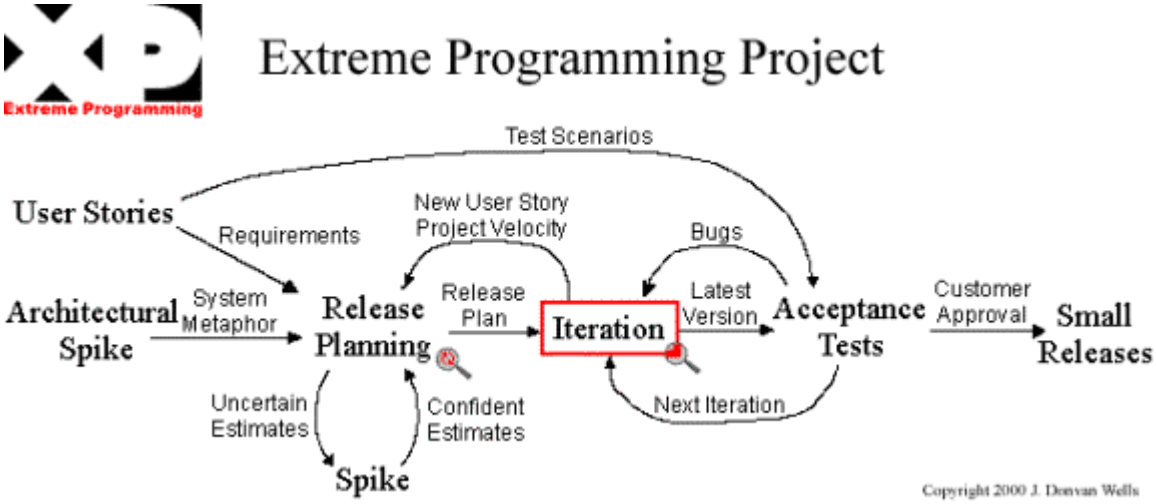
图：经典的瀑布模型

敏捷方法强调对变化的快速响应能力，它通过引入迭代式的开发手段，较好的解决了如何应对变化的问题。这里要指出的是，“迭代”并非是一个新生概念，以迭代为特征的开发方法由来已久，只是名称叫法有所不同。例如：由 **Barry Boehm** 在 **1988** 年提出的“螺旋模型”便是一种具备鲜明的迭代特征的软件开发模式。



图：**Boehm** 的螺旋模型与敏捷方法中的迭代开发很相似

敏捷方法将整个软件生命周期分解为若干个小的迭代周期，通过在每个迭代周期结束时交付阶段性成果来获取切实有效的客户反馈。其目的便是希望通过建立及时的反馈机制，以应对随时可能的需求变更，并做出相应的调整，从而增强我们对软件项目的控制能力。所以，就这一点而言，敏捷方法对变化的环境具有更好的适应能力，相比于经典软件工程方法的“计划性”特征，敏捷方法在“适应性”上具有更大的优势。



图：作为敏捷方法典型代表的极限编程，迭代开发是其核心之一

经典的软件工程方法旨在定义一套完备的过程规范，使软件开发的运作就像是机器设备的运转，人在其中则是可更换的零件，不论是谁参与其中，机器都能运转良好，因此它是面向过程的。这样的做法对于许多软件企业而言，是具有很大吸引力的。这意味着：开发进度的可预见性，流程方法的固化与可复用，人力成本的节省，人员的流动不会对软件开发构成影响。其背后所隐含的观点则是：软件从业者无需是具备非凡智力的高级人才，人是一种可以被任意替代的资源。

早在上个世纪 **70** 年代，**Frederick P. Brooks** 在他的《人月神话》一书中就曾经指出：向进度落后的项目中增加人手，只会使进度更加落后。在衡量项目的进度时，我们无法将人和时间作为两个可以彼此随意互换的因子。其实这也从一个侧面解答了人是否是可替代资源的问题。而在随后出版的《人件》一书中，**Tom Demarco** 则直截了当的指出了：知识型企业的核心就在于人。

在敏捷方法中，人们也非常强调人的作用：没有任何过程方法能够代替开发团队中的成员本身，因为实施过程方法的主体便是人；而过程方法在其中所起的作用，则是对开发团队的工作提供辅助支持。如果得不到团队成员的积极响应，那么不论是什么样的过程方法，都会难于见效。大概许多软件从业者们都愿意相信自己所从事的，是有着高度创造性与专业性的工作，基于这样的前提，他们也就更加愿意接受宣扬“以人为本”的敏捷方法了。

来自制造业的启示：从泰勒主义到精益制造

如前所述，经典的软件工程具备“计划性”与“面向过程”的特点，这在很大程度上受到了泰勒的科学管理理论的影响。这一理论是 **Frederick Taylor** 在上个世纪初为了系统化的提高工厂生产率，并使作业标准化和规范化而提出的。在随后的几十年中，泰勒理论在工业管理领域获得了极大的成功，以至于它对人们的影响也延伸到了其他的新兴领域。

泰勒理论的一个关键理念是：认为干活的人并非是那些知道怎样才能把手里的活干好的人。因此，管理者需要将计划与执行相分离，由具备较高素质的人员来进行先期的设计与规划，然后由普通工人依照固定的计划进行实施，以此来提高生产效率。泰勒主义寓示了一种类型的工作模式与社会结构，也许它在管理一个工厂时是非常有效的，但它却并不适合那些充满变数，而又有着高度创造性和专业性特点的软件开发工作。

不可否认的是，过去和现在，泰勒的理论潜移默化的影响着众多大大小小的软件企业。设计阶段与编码阶段的严格分离，过程中形成大量与程序无关的文档，分属两个对立阵营的架构师与普通

开发者，由独立的质量控制部门对软件制品进行质量检查。无论有意还是无意，这些做法都带上了泰勒主义的烙印。

让我们依然将眼光投向工业领域。事实上，现代制造业也在逐步脱离传统的泰勒模式，一个显著的代表便是由日本丰田汽车公司提出的精益生产管理理论。精益理论的创立者大野耐一曾经说过：最大的浪费就是生产过剩的浪费，制品没有真正产生价值，制造所花的成本就是浪费。精益生产的核心思想就是要消除制造过程中每个环节的不必要浪费，通过降低成本的方法来获得企业更高的经济效益。而当前的敏捷方法则在很大程度上也借鉴了这些思想，例如：在《解析极限编程（第二版）》中涉及极限编程背后的理论支撑时，**Kent Beck** 就曾专门用了一章的篇幅来讨论丰田的生产制度。

精益理论对生产质量是尤为重视的，因为这样可以避免由于不必要的返工而导致的浪费。如果生产线上的制品质量足够好，那就不必求助于独立的质量部门，这也就意味着人人都要为质量负责。它还强调所谓的“零库存”，工人生产的制品会在生产线的下一环节立即投入使用，而不是进入仓库。这可以使制品在立即获得使用价值的同时，也获得了前序工作是否正常的及时反馈，而且还省去了仓储的成本。仔细对照便会发现，所有这些观点都能在敏捷方法中找到影子。

为经典正名：从对立到融合

行文至此，也许读者会觉得本文在很大程度上带有一定的倾向性，不过事情还不是如此绝对。在我们拥抱变化，欢呼敏捷的同时，也应该客观的看到经典软件工程的合理性因素，并从中汲取有益的元素。

首先，在一些具备固定需求的应用领域，经典的软工实践依然可以发挥出很好的价值，这在一些外包项目中较为突出。当然，对任务要求极为苛刻的大型国防项目自不必说了。

其次，经典软工所强调的前期规划与风险控制等诸多思想，其实也影响着敏捷方法的演化，使之形成了不同的流派。例如，一般认为，相比于 **Kent Beck** 的极限编程，**Peter Coad** 的 **FDD**（特性驱动开发）就更加强调前期的领域模型设计，而 **Alistair Cockburn** 所倡导的敏捷方法则非常关注以用例来捕捉需求的技巧。

此外，在一些习惯了经典软工思维的软件开发团体中，敏捷方法的引入也可以采取逐步渗透的方式，例如：**Kent Beck** 就曾提到过在传统的瀑布模型中嵌入迭代式开发的例子。一些大型的软件企业也在引入敏捷方法的部分实践。这些都能给我们以启示：不论是敏捷方法，还是经典软工，亦或是两者的融合，只要在现实中产生实际的价值，就会为人们所接受和认可。