“ 轻 ”方法与满意质量市场驱动的软件工程实践 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/149/2021_2022__E2_80_9C_E8_BD_BB_E2_80_9D_E6_c41_149676.htm 随着信息技术的迅猛发展，今天的IT从业人员正处于这样一种进退两难的境地：一方面，根据以往的痛苦开发经历，他们知道如果采用杂凑的作坊模式来开发复杂的、高质量的信息系统具有太大的风险；另一方面，他们也同样知道，形式化的、戒律森严的软件工程方法（典型情况下是与ISO9000和SEI-CMM相关的）又常常是官僚主义和耗费时间的，不可能满足目前高度竞争的“ Internet时代 ”环境下对于进度方面不断增长的挑战性要求。显然，如何使得企业在保证软件质量的前提下，同时又能够适应快速变化的市场需求，无疑是业内人士关注的焦点。为此，本文从市场驱动的IT开发特点分析入手，对目前国际上正日趋成熟的“ 轻 ”方法和满意质量的内涵和操作加以讨论，同时以快速应用开发为实例介绍了具体操作及注意事项，为读者进一步深入理解现代软件工程实践工作提供帮助。 一、市场驱动的IT开发工作 当今的IT企业正处于一种前所未有的竞争环境之中，无论是作为独立软件供应商还是增殖服务商，也无论是开发直接面向市场的软件产品如游戏、工具、多媒体等等，还是为用户定制软件如电子商务应用、MIS等等，市场竞争都无处不在，IT开发工作被迫在市场驱动的状态进行，具有以下典型特征： 快速演变升级的基础技术，必须及时跟踪基础技术的革新并调整策略。 持续的竞争，必须不断推陈出新来满足用户需求。 时间就是金钱需要在时间、质量、费用之间达成折衷。 从业人员聪明且教育程度

高，能够在一定范围内进行自我管理。 以往流行于IT行业的设计评审构造模型已过时。IT业最新的技术和工具改变了以前项目运行所必须遵循的逻辑顺序。比如：现在一些工具允许“ 即时 ”创建屏幕，使得冗长的设计评审构造模型不再适用，而要求一种更难于计划的原型审核构造再审核再构造的过程。 角色重叠：现代技术和工具使原来专门的设计、分析、编码人员之间严格的区分界限变的模糊。 以往的IT开发人员相对于当前市场驱动下的开发人员而言，至少拥有以下优势：控制他们的应用开发技术。在传统的企业应用开发中，机器配置和其他系统元素在实现阶段就被指定。现在，这种优势不复存在，很少有哪个系统被实现为在专用机器上的孤立应用。相反，他们经常是企业用户机器上的另一种应用，必须与其他商业应用程序联合使用，这使得IT开发人员不得不为保持与CORBA、Internet、或数据库标准的变化相一致而疲于奔命。同时，持续的竞争、紧迫的进度和严格的预算让IT开发经理不得不经常根据销售人员或客户的要求而调整开发工作。 显然，市场驱动的新环境给IT开发带来新的挑战，如果照搬套用以往的成功经验和开发模式并非明智之举，必须适应新形势的要求，重新思考和定位。 二、“ 轻 ”方法浮出水面 随着软件工业不断发展，各种各样的模型不断涌现或退出历史舞台。早期从不同角度提出的各种设计表示方法（常常以发明者的名字来命名）目前似乎已经聚合成为UML这种被广泛接受的标准，结构化设计方法也正在让位于面向对象设计等更受欢迎的方法学，这种变化在更高层次的全局性开发方法学方面同样进行着。 传统意义上的软件方法学描述通常“ 能够 ”处理任何大小的项目，而实际上真正的困难

就来自于如何对这些方法加以裁剪以适合较小的项目。针对这种理论与实际的脱节现象，国际上一些著名的软件工程专家提出所谓“ 重（heavy）”方法和“ 轻（light）”方法之分，试图为快速发展的软件工业探寻更切合实际的解决方法。所谓“ 重 ”方法，就是指形式化的、戒律森严的软件工程方法学不仅指这些方法所生成纸文档重量，还意指管理资源投入、QA评审的程度和开发人员被要求遵守的严格流程。相对的，诸如快速应用开发（Rapid Application Development，简记为RAD)和原型方法等则可以被称为“ 轻 ”方法不仅是因为这些方法倾向于产生最小数量的纸文档，还因为其将管理资源投入最小化。不幸的是，1990年代的许多RAD项目在方法学上采取了过“ 轻 ”的处理以至于几乎不存在什么方法，这些项目常常退化为杂凑式作坊开发，实质上根本没有任何文档。 显然，需要在两种极端方法之间找到平衡点。 轻方法代表了一种有意识的风险防护方法，依据不同风险在与开发相关的各种活动中投入相应时间、资金和资源。例如，进行多少需求分析工作才算是过多，拟或过少？针对一个几百个需求的开发项目而言，一个需求分析“ 轻 ”方法（requirements-light approach）可能是由将每一个需求通过一个简洁的句子加以文档化的行为所组成，一个需求分析“ 中 ”方法（requirements-medium approach）可能要求对每个需求通过一段描述性文本来文档化，而一个需求分析“ 重 ”方法（requirements-heavy approach）则可能要求详细的UML模型、数据元素定义和每个对象方法的形式化描述。 究竟选择需求分析的“ 轻 ”方法还是“ 重 ”方法很大程度上受到公司产品上市时间或合同期限压力的影响。同时，公司雇员的流

动率也是一种影响因素：作为形式化开发过程的理由之一认为，如果有一份详细的文档来记录需求、设计和编码，那么一旦在项目进行中关键开发成员离职所造成的混乱将会被尽量减小。然而，尽管1970年代和1980年代的系统生命周期预期为十年或二十年，也许Interbet时代的网络公司更愿意正式承诺其电子商务应用仅持续一年，然后被废弃或完全重写。如果正是这种情况，并且如果下一代应用预期与当前应用存在质的差异，那么仅仅为了达到SCM-CMM三级就遵循需求分析“ 重 ”方法还真的有意义吗？ 同样地，针对设计和测试工作采用什么样的形式化和严格程度才是合适的呢？与项目管理有关的时间汇报、进展汇报、状态会议及其他常见活动又如何呢尤其针对那些仅持续一周或两周的项目？这些问题总是相互关联的，但是一些传统上被接受的答案却需要至少每隔几年重新审视一下，因为成本-收益参数正在随着商业环境、技术和软件开发人员的变化而不断变化。 轻方法还重新审视了历史上有关投入资源在需求分析的假定，以及投入资源在过程改进的假定。1981年Barry Boehm在他的经典著作“ 软件工程经济学 ”（Software Engineering Economics）中指出了一项惊人发现，即如果我们在项目的系统分析阶段引入一个缺陷的话，那么在项目的分析阶段发现这个缺陷会比允许这个错误直至进入设计阶段才被发现节省约10倍资源。但是Boehm在此做了一个在新千年的头十年中未必依然正确的基本假定：仅当该缺陷在生命周期某阶段发生时可通过某种方式加以鉴别，那么这种数量级增长关系图才是相关的。在今天的环境中，这个前提假定在许多商业条件下都是不成立的。比如，当Bill Gates阐明对于浏览器IE的需求时，可能他会

说“ 就象Netscape Navigator那样，但要更好 ”，可能Netscape的Marc Andreesen也会这样想：“ 我希望使Navigator就象Mosaic一样，但要更好。 ”但是当Tim BernersLee在构建WWW的初始版本和浏览器的第一个草样时又该如何考虑呢？让他写出详细需求的意义何在呢？ 与此同时，重方法的倡导者争辩说，如果一个缺陷在开发阶段就被发现，那么就不应当责备引入该缺陷的个人，而应重新检查允许该缺陷发生的过程本身。但此处又有一个基本假设，也就是说，我们值得投入资源在鉴别一个过程中潜在缺陷的唯一理由是我们希望再次使用同样的过程因为我们的下一个项目将会与上一个项目足够相似，很自然就应使用同样的过程。但是现在事物变化是如此之快，以至于完全不能保证第N 1个项目会与第N个项目有任何相似之处。因此，昨天的过程可能不得不为了明天的需求而发生实质性变化，换言之，也许只有投资于过程中的重要缺陷才是值得的，因为一些细节仅针对某个特定项目才有意义。 当然，仍然有一些环境需要我们继续依赖于旧的、基本的软件工程原理，在这些环境中重方法被证实依然正确。但是我们应当扪心自问，隐藏在这些原理背后的前提假定是否依然合理。对于许多今天的项目而言，一些根本性的前提假定需要加以改变，而轻方法将是具有最优性能价格比的方法。 可以看出，轻方法的基本思路是试图在项目范围、成本、时间和质量之间达成一种平衡，其关键是在足够的管理可见度、足够的灵活性和足够快的开发速度以完成工作之间找到这种平衡，必须严格审查你想要对项目加入或删除的控制手段的价值何在。尽管我们可以将某个公布于众的开发方法作为基础进行裁剪，但必须深入理解你要执行

的每个步骤的理由，尤其在项目的初始规划阶段，就应明确定义开发方法，确保项目团队成员的参与和认可，并以满足项目的商业需求为目标。