第 3 章

流畅高质量交付用户价值


6min

敏捷的关键是什么？本质上来讲就是流畅高质量地交付用户价值。

这里我们抓住 3 个关键字：第一，流畅，流畅英文翻译为 Flow，就是流动；第二，最终交付的是用户价值(Value)，所以一定要清楚地知道什么是用户价值；第三内建质量(Built In Quality，BIQ)。首先希望它流动起来，流动起来之后内建质量，最终是希望交付价值，本质上以下三点就是要去做的事情。

1. 明确用户价值

我们如何明确用户价值？其实，现在做软件交付的瓶颈从某些角度来讲并不在技术端，研发领域上没有什么技术瓶颈，例如 DevOps 在绝大多数公司已经做得很好很快，唯一的问题是拿不准用户价值。拿不准用户价值就不能使用以前的瀑布模式来做，因为瀑布模式只能针对明确的目标去做。只有在需求不明确的情况下，才会换个提议，要不试试这个，要不试试那个，这时就有了快速试错的概念。

传统的瀑布模式希望一次性完整地大批量交付用户价值，在当前 VUCA 时代很难做到，难度在于我们很难一次就把用户价值看得那么准，用户价值会随着时间推移发生变化，所以用户价值决定了当前的研发模式是使用敏捷还是瀑布模式。如果能渐进式地明确用户价值或者局部地明确用户价值，找到最小可交付产品，则使用敏捷迭代开发模式是比较推荐的一种方法。

2. 提升流动速度

用户希望看到价值，所以应尽可能快地交付，解决当前阶段的痛点。那么必须提高流动速度，分批交付的时间越短，节约的时间就越多。提升流动速度除了技术和框架的改造，还有工具的支持，常见的持续交付体系都是以自动化为基础的。

3. 构建高速交付下的质量保证体系

如果用户价值明确，研发团队能保证实现这些价值，业务分析人员(BA)能帮助研发人

员完成质量验证，则高速交付下保证质量是没问题的，但现在的情况是业务分析人员拿不准用户的痛点，研发人员做出来的解决方案也不知道对不对，而此时又需要快速交付给用户，就会产生质量的问题。大家想想，如果很明确用户需要解决什么问题，研发人员直接做出来给用户验收就行了，其实都不需要测试，所以云层一直觉得测试不是绝对必须的，但客观事实上团队又需要测试，因为我们不能让用户直接去试，这样的代价太大了。

在高速交付的背景下很难拿准用户价值，需要进行测试，用一个低成本的方式来代替高成本的做法，这才是核心问题。质量对于不同的用户价值是不同的，如果大家最近看过《我们的乐队》，会发现普通的大众听音乐和专业人员听音乐是完全不一样的。例如周杰伦的《告白气球》很好听，其实这首歌的编曲没有什么特别；但是当听黑暗三部曲《夜的第七章》《以父之名》和《夜曲》的时候，觉得编曲的水平就非常顶尖了。所以才有外行看热闹、内行看门道的说法。对于普通用户来讲价值明确、好用就行了，其实不需要专业的测试。

只有在非常专业的领域才需要高质量保证，但我们做的软件绝大多数情况是给普通用户使用的，所以小型互联网公司没有测试是很正常的，因为你没有办法从专业的角度去保证质量，毕竟成本太高。例如我买个肉包子，我还专门检验一下里面的肉是不是纯的猪肉，其实是没必要的，性价比很重要。

敏捷测试的核心是保障最后交付的用户价值是客户所期望的用户价值，并且是在高速交付的情况下保证质量。这对我们来讲是个伪命题，是要求我们做得又快又好，表面上看起来没办法，但办法总比困难多。

后面有专门的章节讲用户价值，这里先部分展开讲解为什么用户价值如此重要。用户往往只给我们一部分信息，这部分信息是我们能看得见的，传统说法叫作显式需求。显式需求就是用户明确告诉你需要的内容，但在显式需求后面往往是有隐式需求的。用户价值里面真正难的，并不是用户所讲的表面的内容，而是背后的深层次内容。

目标与关键成果法（Objectives and Key Results，OKR）同样是在讲价值这件事情，做事情有多种方法可以落地，以前做事情解决问题就行，例如胃疼了喝热水就好，但现在我们需要知道如何去预防胃疼才是最重要的目标，而不是简单地喝热水去解决胃疼的标，治本才是最后需要的关键价值目标。

最近后浪很流行，梦婧老师写了篇文章说后浪这件事情，里面就讲到世界的对等性，云层的课程还是很有前瞻性的，大概几个月前我就讲过这个问题了，没想到这次又红了，讲的是95后的员工不能被骂的原因，其实不知道大家有没有想过，95后的员工好不好管理？很多人会说95后不好管理，现在的年轻人很自由很奔放，要他干什么他都不愿意，例如我家孩子同样是这样的，我跟他说今天下午要好好学习，他说不，他要看动画片。我拿他一点办法也没有，又不能打。换以前我们小时候，给我印象最深刻的是老师非常厉害，只要上课不听

话，“啪”一个黑板擦或一个粉笔就扔过来了，然后去罚站，我们那个年代就是这么过来的，现在孩子不会有这样的经历。其实对于 95 后的员工来讲，我们发现不好管理，在于不同的管理者看法是不一样的，就像上过架构师和敏捷课程，你就会发觉，越是年轻的员工越好管理。因为对于他们来讲做这件事情的价值目标是非常明确的，95 后员工有唯一并且清晰的目标。就像大家说的只要钱给够就行了，难度在于如果钱给不够怎么办。

对于像云层这种 80 年代甚至于 80 年代之前的人就不一样了，不是简简单单一个钱的事情，公司不但要给我钱而且还要给我长期的发展空间，能够动态地调整并且工作时的心情还要好，最重要的是不加班，要经常能请假陪孩子等。所以其实是有区别的，你说 95 后不能骂，是真的不能骂吗？或者我们会说 95 后或 00 后不能吃苦等，但当我们真正去看会发觉，其实我们只看到了一些 95 后或者 00 后不好的一面，但是没有看到他们更好的一面，于是大家会发觉价值的变化出现了。95 后愿意去做自己感兴趣的事情，而且能够专注地将这件事情做专、做精，并且没有后顾之忧地去完成这件事，因为工作不是为了生存。

我们现在所有人都从生存开始走上有尊严地活着，你骂我是没有用的，你骂得对我认，你骂得不对我走，这就是现在的情况。因为已经没有生存的压力，开始走上了有尊严地活着或者逐步去实现过上自己想过的生活。这个时候面临一个问题，到了三十岁左右的时候，你可能过上了自己想要过的生活，独立自由，但是也面临了很多的责任，上有老下有小、房贷、车贷等，你是愿意为了别人回到生存，还是独自过自己想过的生活呢？这是当前最难的抉择，如图 3-1 所示。

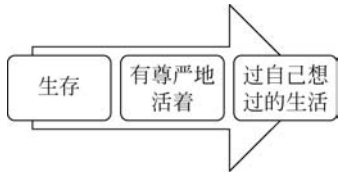

图 3-1　用户价值的改变

在人生轨迹中解决问题的方式开始变化了。从开始为了生存而拼命工作，到有了较好的生活条件，但是各种欠债很多，要及时偿还，再到高收入带来的高生活品质，这时候觉得天天加班“996”也愿意，甚至“007”也可以，有钱真香。再之后就会发觉我做这件事情比别人做得好，愿意沉迷于工作，日渐肥胖无法自拔，很多时候生活都不需要了，先把钱赚到，因为有了钱之后你会发觉很多事情都能解决了，如图 3-2 所示。

在这个过程中按照标准说法就是时代变化，从“要我做”变成“我要做”，这其实就是文化的变革。我们会很奇怪，为什么别人的科技发展比我们快？以前是我逼着你去做创造性工作，例如写代码，逼着你天天写 100 行代码，能写出来但质量肯定不高，因为你不是发自内心去做这件事情。现在我们按工作成果给你工资，你自己决定写多少行代码，当真正放开让你去做的时候，你反而会觉得如果做得好回报是很好的，其实这就是所谓的自由工作制，特别适合 IT 行业的创造性工作。

为什么要谈创造性这个词呢？绝大多数工作是没有创造性的，在这种情况下不需要

图 3-2　工作的进步

“我要做”这件事情，只需“要你做”就行了，所以在大多数情况下工作仍然不需要主动性，只需被动地推，导致存在大量等待或者拖延的情况。大家在疫情的这段时间会发现，很少有人可以做到逼迫自己进步的，你还是被动的。包括云层也是靠被动方法来做的，云层是怎么提高自己的工作主动性的呢？其实是云层自己立个 Flag，我们下周开始上两天课，说完之后云层可能马上就后悔了，但是说都说出来了，那就逼自己去做吧！所以云层是通过给自己立 Flag 去做创造性压力的，这是云层解决问题的方式。

时代的变化会让我们在做某件事情时，带来成长的变化，会让我们在看待问题时方式和层面不同了，以前看问题会非常表面，都是显性需求，而现在看问题会从最终的价值出发。

最终的价值是什么？云层认为对于一个年轻人来讲最有效的方法，就是在接受寂寞、独立的过程中努力熬出头，熬出头后会发现很多事情都简单了。但如果你不愿意接受这个过程，去谈世界上为什么没有真爱这种事情，那还是简单点，有钱就好。想要有钱，需要花时间去创造真正的价值，有钱往往是最有安全感的东西，因为大部分东西可以用钱买到，大部分问题可以用钱解决。

所以找到最终的价值，是我们成长过程中所看到的价值变化。

3.1　加速交付

敏捷最终实现的是要交付价值，交付价值到底是什么？我们不仅要看显式需求，还要

能真正找到用户的最终目标，这个最终目标就是我们需要的价值目标。当价值目标确定后，剩下的过程就是如何让目标快速实现了。

3.1.1　如何加速小批量交付

想要加速小批量交付，首先需要分析交付过程并找到瓶颈，绝大多数软件研发遵守以下流程，如图 3-3 所示。

图 3-3　软件交付流程

交付流程包括创建分支、更新代码、合并分支并发包、发包后测试、测试通过后发布生产等，整个过程其实是很长的。这其中需求以周为级别、研发以月为级别、打包以天为级别、测试以周为级别等、发布以天为级别来更新统计。所以基本上提出需求要一周、研发完成要两个月、打包发布要一两天、测试完成要两周左右、发布上线要两三天，往往交付用户一个价值的时候所需的时间是三四个月一个版本。

例如每个月的第一周固定发布升级版本，在这个过程中我们要把瀑布模式优化。现在的瓶颈很明显，就在月级别的研发过程，因为它占比是最大的。如果我们希望小批量交付，则要减少每次提交需求的量，减少研发的任务，这样交付周期才能从月级别压缩成周级别甚至天级别，而测试也要从周级别压缩成日级别或小时级别，这是小批量研发过程中要去推动流动速度的关键。云层建议大家去玩两遍敏捷中的 Coin Game（硬币游戏），玩过后会对整个流动过程的理解有很大帮助，并且会帮助大家理解看板。看板中会讲到当有了过程，如何去跟踪每个过程的执行状态，从而会特别容易看到瓶颈在哪。

3.1.2　可以多快

通过小批量交付，迭代交付的速度会比以前快，快到什么程度呢？以前瀑布模式以月级别交付，现在敏捷模式通过团队（业务部门、开发部门、测试部门）在一起规划每次迭代的内容，完成之后提交、发布、生产，从而实现天级别发布。而 DevOps 彻底把运维上线过程也整合进来，从而实现小时级上线，如图 3-4 所示。

软件交付理想中最好的做法是像瀑布一样，规定下个星期食堂做什么菜，提前买好这

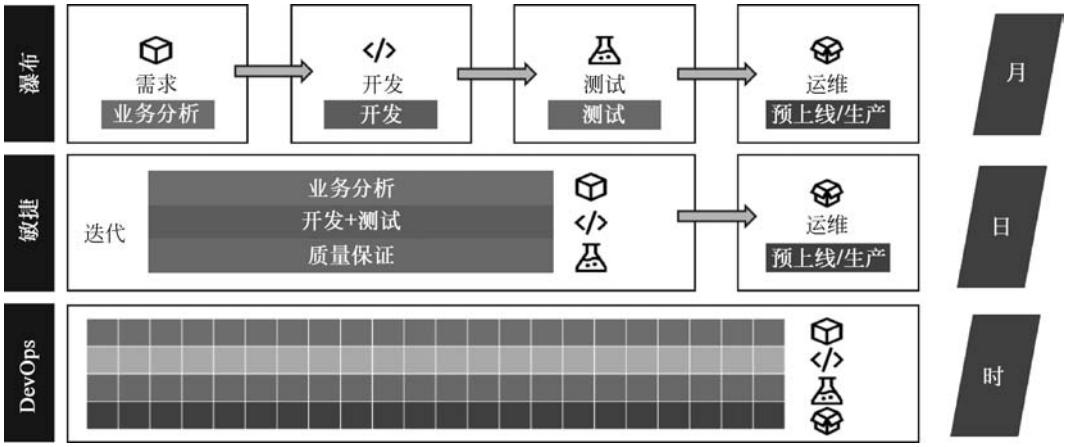

图 3-4　软件交付可以多快

些菜，缺点是不一定能满足用户变化的需求，所以大家会发觉食堂吃久了还是会觉得出去点菜吃好，因为这是自己内心想要的，特别是在食堂没有自己想吃的菜的情况下。

现在行业的做法基本是 DevOps，核心是把运维整合进来，没有发到生产上的东西都称为在制品（Work In Process，WIP）。整个 DevOps 里强调的是消除浪费，就像肯德基、麦当劳这类快餐，你要的食物是可以随时快速交付的，而不像传统中餐要吃个煲仔饭大概要等 30 分钟，因为煲仔饭不能提前做好，而快餐基本不超过 5 分钟就可以做出来，作为需要果腹的人来讲快速吃饱的价值实现了。

DevOps 的一个精髓就是通过整合交付环节的各个角色，以最短的时间交付用户价值。其实这也是生活中特别明显的支付升级，方便、快捷、安全。以前我们买东西需要先规划行程，坐一两小时的车到商店，然后去选，还要对比价格合不合适或者去试穿，看是否好看再下单。现在变敏捷了，我直接在网上买东西，连门都不需要出去了，看到合适的东西买下来寄过来，现在甚至可以直接在体验店试或下单寄回家，不用大包小包地逛商场了。消费的便利及支付的快捷都客观地提高了消费频次或者消费周期，特别是 100 元以内的免密支付，让你觉得买回来这点东西是没有“感觉”的，其实这就是加速支付，那么如何去实现这个加速过程呢？

3.1.3　如何加速

做加速一定要求自动化，研发自动化、测试自动化及发布自动化。注意云层没有写需求自动化，其实是有需求自动化这个概念的，但现在已经有需求自动化技术，只要按照模板去填写需求，它会自动帮你做需求分析、自动研发、自动测试的过程。但云层并不推荐现在

做这件事情，因为原则上现在所有的东西本质上还是被隔离在业务域外的，现在所围绕的范围并不包含业务域，只能说希望提供给我们的价值尽量小并且尽量准确，但是并没有到如何让这个业务域更快，还是围绕着研发域来谈的，研发域并不是一个业务域的部分内容，所以这就是为什么大家会说像 PO、PMO、BA 工作好像很好做，做产品经理很简单，随便写一写就可以了。其实现在看起来简单，大概半年一年后就很难了，因为只要后端的速度运行起来，对前端的要求就会越来越高，而且非常好量化。在 BAT 公司对于整个业务域要求是非常高的，高在如何花最少的钱做出用户最想要的内容，就像巧妇难为无米之炊。

交付加速有两种方式：过程自动化及减少过程，如图 3-5 所示。

如何过程自动化？例如希望快点把菜做好怎么办，放进微波炉或者蒸箱 30 分钟就好，这就是自动化的做法。减少人工在里面的影响，并且让它更快一点，就是温度高一点、功率大一点。

图 3-5　如何加速

还有一种方法就是减少过程，核心是减少沟通的过程和内容。敏捷宣言里有一条“可工作的软件优于冗余的文档”，其实敏捷宣言并没有说不要文档，说的是冗余的文档有用，但是不如做可用的软件，即减少对于内容的说明，文档太多其实是没有用的。少沟通是通过同理心主动的方式来开展的，看板讲的就是通过可视化来减少沟通，把以前的 get 变成 push，如同在性能课中讲从点播变成广播的优化过程，让过程减少。想想看，如果每个人都在问你明天能不能发布新版本，不如挂块牌子标注明天一定发布新版本或者明天能发布新版本但还差哪一个部门没有做好，大家一看就知道解决那个“吊车尾”部门就行了，这也是加速的方法。

3.2　过程自动化

Everything As Code 是《持续交付（发布可靠软件的系统方法）》书上提到的概念，也是 DevOps Master 考试必考的一点。所谓 Everything As Code，就是所有的东西一定要代码化，这也是为什么大家会看到最近几年测试开发比较主流，原因是对于绝大多数测试人员来讲缺乏写代码的能力。

测试开发指的是具备70%开发能力，20%的测试工具使用能力，再加10%的测试技能思想。在提高交付速度的初期，提高自动化比例是基础建设，这是当下比较有价值的工作，等到自动化达到一定比例，测试开发工作饱和时再开始谈测试设计，所以怎样才能成为一个真正的测试人员或者当下所需的测试人员，这是不同时代下知识体系结构的问题。

3.2.1 项目化管理体系

过程完全自动化后要回到项目化管理体系，如图3-6所示。做高流动虽然谈的是研发域的事情，但是仍然需要有一定的需求管理，需求端怎么去做一些高速流动内容，本质上还是任务管理体系和看板体系的内容。

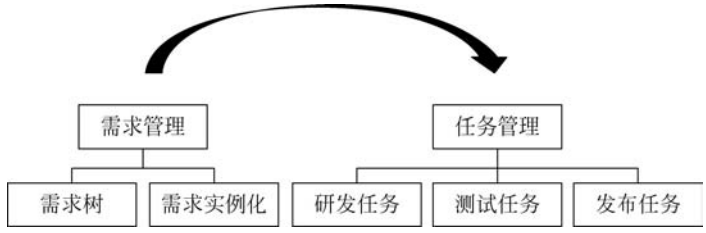

图3-6 项目化管理体系

项目化管理体系要求两件事，第一是传统需求管理的需求树，敏捷中有实例化的要求，因此需要有树形化的需求管理，否则不知道怎么交付需求，需求管理最后会以用户故事地图的形式来做，但仍然涉及条目化的问题；第二是任务管理，实现对应的需求应该包含哪些任务，这里包含研发任务、测试任务和发布任务对应的内容，让我们能够看到一条需求是什么时候进入研发状态的，研发围绕它做了哪些任务拆分，并且每条任务的研发任务对应哪些测试任务，最后发布的时候把研发任务和测试任务都验证了再发布出去，这时候就需要有一个任务管理体系。

任务管理体系当前的名词叫作流水线(Pipeline)，就像制作一个零件要经历几个工序，把确定好的内容从这个设备的左边推进去，右边出来交付结果，如果质量不合格，就把交付结果退回去。软件可以一对一或一对N生产，最后的结果是可以回溯的，整个项目化管理体系要可回溯、可跟踪且可量化。

3.2.2 自动化依赖于规范

当前的行为驱动开发(Behavior Driven Development，BDD)、验收测试驱动开发(Acceptance Test Driven Development，ATDD)自动化理想体系，有强烈的规范要求，因为我们希望能够遵守规范去做。当下垃圾分类的要求就比较规范，但其实也有一些模糊的情况，例如盒装

酸奶怎么规范，是干垃圾还是湿垃圾？牛奶盒可以非常简单地还原成干垃圾，但是盒装酸奶很难简单地还原成所谓规范的干垃圾，原因是盒装酸奶倒不干净，倒不干净就面临直接就丢到干垃圾里面，拆开来还是大量湿的，但是又不能丢到湿垃圾里面。如果做得规范，就要把酸奶盒拆开洗干净再晒干，然后放入干垃圾，但对我们来讲这很难做到。

所以本质还是在规范要求做到哪个级别上，以前云层上课问过一个等价类边界值的问题，问足球有一部分压在球门线上到底算不算进门，其实这就是等价类边界值中的边界值问题，所以很多场景需要有绝对的判断机制，不能仅是所谓的概念。例如纸盒到底含多少水分是干垃圾，含多少水分是湿垃圾，其实我们是没有这个规范的。干、湿垃圾分类能自动化吗？其实是不能自动化的，因为没有明确规范。因为做不到规范就需人工干预，人工干预就是因为有些边界需要人工去判断和容错。为了解决这些没有明确规范的被测对象，往往通过测试代码的一些容错模式来兼容，但这个做法并不好。

3.2.3　让研发自动化

研发的核心过程就是代码管理，需要有个类似于 Git 的管理工具去做代码管理。如果想有效地让多个团队协作，需要使用类似 GitFlow 的管理模式去做代码多分支管理，如图 3-7 所示。

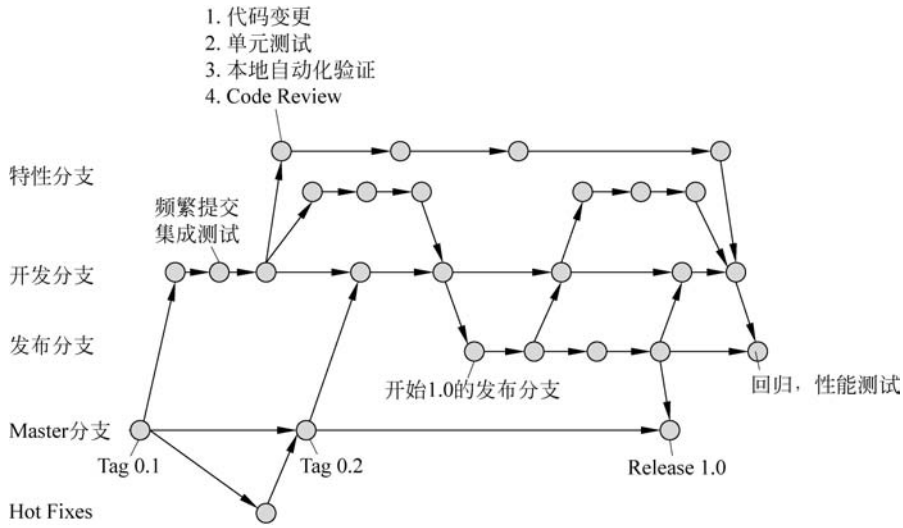

图 3-7　GitFlow 代码管理模式

GitFlow 包含 Master 分支、发布分支、开发分支和特性分支，然后我们会给每个开发任务新建一个特性分支，特性分支完成后回到开发分支，从开发分支提交到发布分支，围绕发布分支和开发分支进行测试，最后将发布上线成功的分支合并回主干。在整个过程中自动

化怎么做呢？

手工过程要先去想如何在 Git 上打命令，从开发分支去创建一个分支，接着需要把代码更新到自己的 IDE 里，然后编写代码，写完代码之后要先在本地 Git 上提交，再上传到服务器上，上传完成后要把当前分支代码完整更新到本机上，然后编译打包，编译打包若有错则需再合并，合并完成后再发布。那么如何将这个过程自动化，构建为完全自动化的分支管理体系呢？

自动化管理分支体系是指在平台上根本就不用关心整个代码管理体系是怎么回事，只需看到分支线选择分支使用。测试人员所要的功能是可以快速看到不同的分支状态，并且选择打包部署。开发人员需要方便地新建、选择分支，同步开发环境，并且在提交更新后，自动判断代码是否冲突及是否存在明显的 Bug，如果不合格就退回。

很少有公司能够做到研发体系的基本自动化，大多数公司是开发人员自己输入 Git 命令从服务器上创建新的分支，写完代码后上传，再手工输入 Git 命令做代码合并。人多了之后很容易导致混乱，然后就开始有各种匪夷所思的命名规则，分支多且混乱，导致发布的时候合并困难。这个时候就需要一套管理体系去解决这些问题，帮助大家可视化，甚至收拢权限去管理分支，分支越多合并就越难，最后发布很容易出问题，就是因为要合并好多个分支，各种乱七八糟的分支，谁都能建分支。

解决这个问题可以参考 2019 年在上海举办的 DevOps Days 上一个分享中讲到的 Facebook 公司只走主干开发，所有开发都可以从主干上拉版本出来，然后合并回去。只有一个要求就是谁最后提交谁负责解决前面的问题。这个模式的好处是在这个过程中后面提交的需解决前面的问题，强行逼迫大家不要在主干上长期占有修改权。

以前的问题是拉个分支就用一个星期，写了一个星期的代码后发觉合并不回去了，现在的要求是就给你几分钟时间，想好要写什么马上把代码拉出来改，改完后合并回去只需大概十几分钟。改一点东西就写一点东西，好处是变更越小整个代码的合并代价就越小，两个人寸步不离在一起出现问题了特别容易解决，但如果两个人隔了很远再去看经常就合并不起来了，因为认知差距出现了，所以需要有个非常好的自动化管理分支体系。

3.2.4 代码质量保证

代码质量保证是构造质量（Build In Quality，BIQ）的基本要求，通常通过 Sonar Qube 的代码扫描实现单元测试和度量统计。大多数公司有，但是做好的难度在于，第一，如何推动公司去做单元测试，一般开发人员是不愿意的，要证明自己写的代码是对的很麻烦，而且还有很多业务需要完成，时间上也不允许；第二，需要一个可视化的质量报告来了解业务实现的情况，但可视化质量报告需要业务部门及测试部门辅助才能完成。

解决了前两点问题后，剩下的难题就在于如何推动检查规则的优化，为什么要优化检查规则？因为检查规则如果过于严苛，则无法达到要求。在这个公司由于业务特点和重视程度可能做得比较好，但带到另一个公司去做就未必适应了，这时候就要做一些优化，所以难点在于可视化报告体系。现在一般的做法是从 Sonar 的数据库里抽出来质量报告，再去写一套自己的中台大屏。

3.2.5　测试质量保证

测试过程包含获取测试包、构建测试环境、部署测试环境和执行测试，难点在于环境的申请、部署和测试执行，其中测试数据和部署 Mock 隔离是最难的。

环境如何一键申请、一键部署，如何调度自动化的执行，自动化执行中如何提供测试数据，如何回收测试报告，如何构建特殊测试环境隔离某些模块，这是质量保证所需要解决的问题。它围绕的内容是持续测试，质量保证过程中最后一步是打包发布的规范。

整个测试质量保证过程中对被测对象的规范是有要求的，如果没有规范测试很难做到高度自动化及全流程自动化。

3.2.6　发布流程

自动化的最后一步是发布打包规则，如何确保在不同平台可以打不同的包，并且确保相关配置信息同步正确。

第一，要有明确的构建脚本，如基于 Maven 打包规则，发布基本上以 Ansible 或 K8s、Docker 容器等工具为主。在打包时需要考虑代码与配置信息的分离，因为生产包、测试包和预生产包的配置是不一样的。在 Spring Boot 中通常需写 3 个 properties 文件对应 Dev、Test 和 Pre 3 套配置信息，并且还要考虑自动化测试环境和性能测试环境等额外环境。

第二，如何发布到生产，使用容器、Jar 还是 War 包发布，因为发布速度的要求不一样，还涉及如何做灰度测试、生产测试等，以及如何确定上线成功之后合并回主干的过程。

以上所有过程都涉及自动化，在这个过程中很重要的一件事情是，要理清楚公司每个环节的每步是怎么做的，接着将所有过程变成自动化流程。发布流程很难，如果不能自动化，那么要清楚了解是什么规则影响它不能自动化，最起码要做到关键路径的自动化。

3.2.7　常见的持续交付流水线

在梳理了整个研发流程后，接着来讲解常见的持续交付流水线产品的研发流程，Coding 平台的流程图如图 3-8 所示。

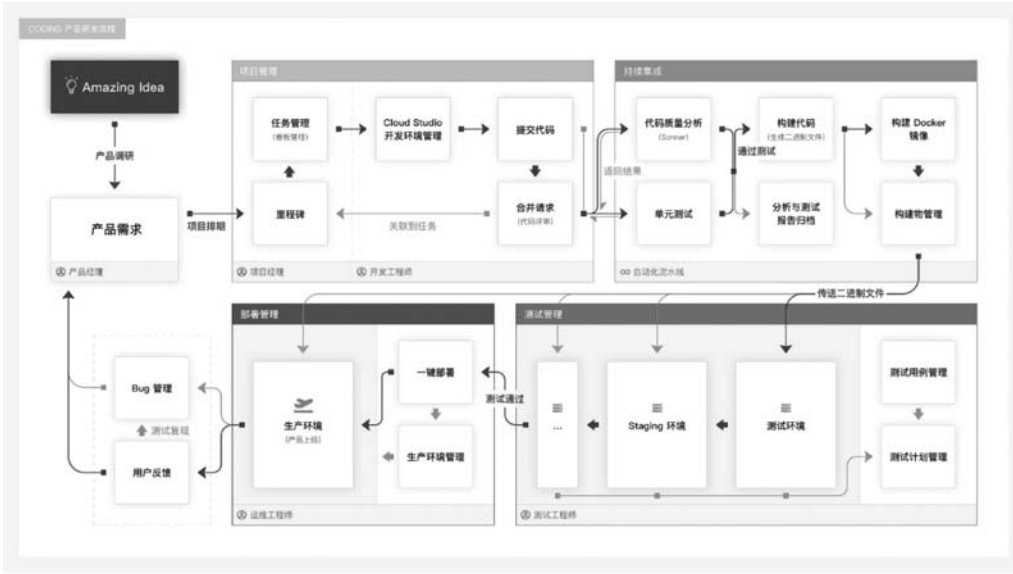

图 3-8　Coding 产品研发流程

产品经理做需求排期存在难度，难在如何构建一个产品的交付排期。在任务管理看板中有个问题叫作资源永远是不够的。如果给你无限的资源，则你会无限地放大产品需求，将没用的需求也加进去，前面章节讲过只有限制交付规模才能实现快速交付，如何限制交付规模？和我们挑选婚纱照一样，所有照片给你看，最后选出几张，不要的全部删除，只有这个过程才会选出你想要的照片，否则你会觉得不如全部打包好了。

在这个过程中要做一个管理，管理什么？一方面，合并请求要跟任务形成对应，一个里程碑对应多个任务，这些任务里包含哪些代码；另一方面，合并请求的代码需要做持续集成，进行代码分析扫描、单元测试，通过之后构建二方库或者发布 Docker 镜像。镜像可以在 Harbor 里管理，二方库可以使用 Nexus 做私有化管理。测试管理中心需要有多套测试环境，如果只有一套测试环境，则所有的测试用例都要在这套环境上运行，会有互相影响。所以原则上至少需要两套测试环境，手工或自动化需要有账号隔离，除了账号隔离之外可能还要动态生成一些数据，所以需要有数据底层。这时候你就会发现构建一套有效的测试环境需要很长时间。在云层看来整个自动化测试里面最大的瓶颈就是环境和数据，而不是测试执行过程，难点在于如何在生成 Docker 镜像时就把需要的数据带进去，甚至数据版本化。

测试完多个版本之后接着做什么？将这个验证好的 Docker 镜像直接一键部署到生产环境，生产环境做生产上的灰度隔离，接着做 Bug 管理和用户反馈，最后回到需求这里就是一个比较标准的持续交付流水线。

对于绝大公司来讲，当需要做一个 DevOps 流程时，本质上就是做持续交付流水线。这就涉及上面提到的各种工具，要么走开源体系把它们串在一起；要么做类似于像这样的产品体系，把所有数据收集在一个平台上，能随时看到一个产品需求现在在什么阶段，什么时候能够上线，以及现在的测试情况。因为对于整个项目管理人员或者高层领导来讲，更需要的是更加全面地去看，价值离交付之间有多远的距离，而不是简单地知道正在做，这需要有量化的数据及可视化的过程。

这是持续交付流水线再上一层要做的事情，流水线工具可以通过各种工具串起来，但是更需要的是在串起来的基础上再去做一个自己的平台。所以大家可以看到华为的 DevCloud 平台、阿里的云效平台和腾讯的 Coding 平台，其实它们做的就是帮大家把开源的东西管理在一起，在自己公司里最后也是要做这件事情。

做持续交付流水线平台首先要理清楚流程，然后自己编写代码及平台，整合流程的各个工具，云层比较推荐找一个开发人员帮你去做这件事情，因为自己去写代码所需要的技能太广了。写一个测试端的工具框架就已经很难了，而 DevOps 级别的东西是很大的，大到更需要一个开发人员甚至几个开发人员来做这件事情。

3.2.8　常见的持续交付工具

当前主流工具中 JIRA 在需求管理上用得比较多，如图 3-9 所示。Git、GitHub 和 GitLab 是常见的代码管理平台，Jenkins 是任务管理平台，Maven 是打包构建框架，Docker 是容器化平台，Gradle 因为做手机端开发要用，所以也要了解。Selenium 和 Junit 用来做 GUI 自动化，做前端 JavaScript 测试需要用到 Qunit，Cucumber 了解一下就行了，不用深入了解。Jmeter、Newman 和 Postman 需要懂一点，因为做接口测试需要，Ansible 作为多平台运维工具要懂，Chef 和 SaltStack 最好也了解一下，DockerHub 和 Harbor 是容器化管理平台，NPM 和 Pytest 作为 Python 的相关框架需要掌握，云平台可根据公司情况选择，日志监控方面 ELK、Zabbix、Dynatrace、Grafana 和 Promethus 都需要掌握。

原则上在做整个流水线体系时，掌握前面讲的这些工具就够了，再根据公司情况可能涉及一些商业工具的整合。如果拥有 DevOps 相关认证能够梳理清楚整个理论体系，在公司中具体落地过流水线工具，则你可达到一个 DevOps 高级工程师的水平。如果有过这类平台的开发架构经验，那么年薪百万的架构师职位也唾手可得。

做过相关的内容就会发现其实并没有想象中那么难，不停地跨栈并熟悉不同领域的工具可能反而是比较困难的。它需要你的知识面很全面，因为只有这样你才能做到端到端的质量保证，其实 DevOps 里面谈到端到端也是在讲这个内容。

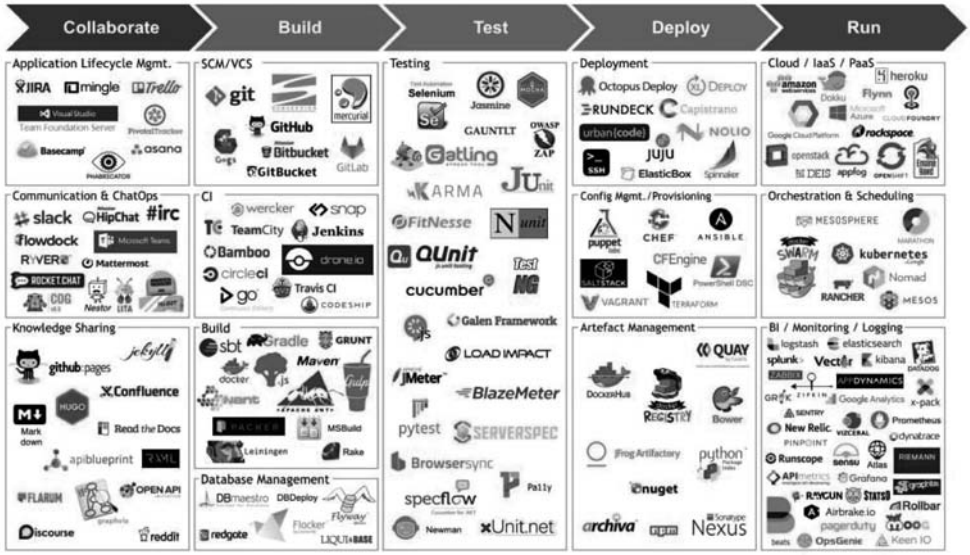

图 3-9　流水线工具集合

3.2.9　构建软件研发效能体系

IT 行业中的软件研发效能体系到底是什么，工程效能？软件研发效能体系就是要让整个软件研发的效能提升，简单来讲就是速度快、质量好、成本低。

在云层看来研发效能体系是分成 4 个维度去做的。第一，管理效能，需求域如何确认交付范围的有效性，需求拆分质量；第二，研发效能，如何让开发更快地写出代码，框架化、工具、版本管理体系能不能更好；第三，运维效能，如何快速发布到生产上，使用工具和框架体系发布；第四，质量效能，如何低成本地保证质量。

我们需要把管理效能、研发效能、运维效能和质量效能整体提高，才能达到最后的目的。在落地 DevOps 时，往往围绕后 3 点的内容，准确来讲不能称为 DevOps，只能称为持续交付的部分。仅围绕技术级别优化已经不够了，如何管理需求、持续为研发域提供待实现价值是要进一步解决的问题。

我们需要在完整的价值流（Value Stream）中找到各个环节的时间和成本，不断消除浪费、提高效能，这个时候需要从 Ops 往前到 Test，再到 Dev 甚至 Biz，所以如果要用一个完整的单词体现可能是 BizDevTestOps 吧。

3.3　减少过程

前面讲了提高交付速度的第一种方法是让过程自动化，接下来讲第二种方法，即减少过程。

3.3.1　构建交付迭代

资源不足不是无法完成交付用户价值的原因。面对有限的资源，最重要的是做最有价值的事情。

首先要做的是需求排期，用户永远会说所有的东西都很重要，都需要放入交付目标中，甚至会动用很多种方法让你认为它是很重要的。如果有能力，你可以尝试一次性全部满足，但是在能力及资源不足的情况下就要考虑完成最重要的部分。所以说资源不足不是无法完成交付问题的关键，关键在于你怎么去帮助用户梳理更关键的内容。

构建 MVP 的多次迭代交付是一直在强调的内容，通过用户故事地图的迭代计划，在有限的资源下逐步交付用户最有价值的内容，并且在每次交付时根据用户的情况动态地调整下一次的交付目标，从而帮助用户最大化实现价值。

3.3.2　可视化过程

在减少过程方法中寻找合适的路径也是一种有效的方法。当我们使用打车软件时，会在不同成本和不同价值目标的维度选择最合适的内容。例如有些事情是很紧急的、不计成本的，我们这时候会选择打快车或者打专车。前段时间看到一篇文章是一个老板叫一个员工坐高铁去北京给某个老板送身份证，这代表什么？寄个快递才几十元钱，为什么要你亲自去送呢，这件事情肯定很紧急，花 1000 元的车票也是值得的，因为寄快递要第二天才能收到。

消除浪费的方法是如何在有限的资源下最有性价比地做这件事情。例如云层下个月要从上海去一次石家庄，就看了一下去石家庄怎么走。无疑就是坐火车或飞机，坐飞机大概两小时，价格也就四百多元钱，算算来回的时间大概要 5 个多小时。我又看了有直接到石家庄的高铁，大概要 6 小时，价格也是四百元左右，坐高铁是可以考虑的，还有别的选择吗？原来还有直达夜车，大概晚上八九点出发，第二天早上到石家庄，11 小时左右。面对 3 种选择，要根据实际情况选择以当前价值最合适的方式。成本最低的做法是坐夜车去石家庄，

然后晚上坐飞机或者夜车返回，不在石家庄过夜，但有些人在火车上就是睡不着觉，所以无论如何坐高铁还是坐飞机都要回来，甚至可以在石家庄再多待一天，下午上完课后住一天，第二天再坐高铁回来也可以，还可以顺便在石家庄逛一下。通过过程可视化分析，找到最适合当前公司交付的速度与成本的平衡点，从而提升价值交付的能力。

3.3.3 价值管理

软件交付可以分为 3 个阶段，待交付、在制品和交付品，如图 3-10 所示。

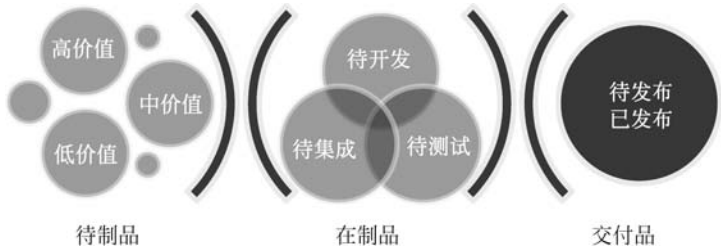

图 3-10　价值过程管理

待交付是指与用户确认过的所有要交付的内容，将这些内容进行价值排序得到高价值、中价值和低价值。推荐在项目开发初期选择高价值、高风险的问题解决方案，因为项目初期高价值、低风险问题是可控的，所以不急着做，应该先把高价值、高风险的问题解决。

首选永远是解决高价值的问题，然后在时间允许的情况下先解决高风险的问题。因为高风险问题没解决还可以解决低风险的问题。如果把低风险的问题先解决了，剩下时间再解决高风险的问题，会出现高风险问题无法解决的情况，因为高风险问题是不可确定的问题，没有办法在时间上非常好地规划出来。

你可以认为高价值、低风险问题是瀑布模式可以解决的，所以定好时间去做就行了，但是在价值管理上应该先做高价值、高风险的问题。在做的过程中就会有待开发、待测试和待集成的在制品，所以管理 WIP 在制品是最重要的事情。如果什么事情都在做，则结果是什么事情都交付不了。

例如云层要考试，会优先学特别难的内容，因为我知道剩下的比较简单的问题肯定能在几小时内解决，哪怕不专门学也能解决。但是我绝不会先把已经掌握得很熟的内容做得更熟，然后剩下一点时间再去做不熟的内容，且在过程中一定要控制在制品的数量，不能把所有的高价值、高风险，高价值、低风险的问题都拿到一起做，因为太多的在制品一起交付，只要某一个在制品不达标，那么整个交付可能就会失败，这也是互联网大小周开发的原因之一，两周一个大版本加一点功能，一周一个小版本改一些 Bug。

一般建议在制品控制在并行小于或等于 4，限制了在制品的数量后，每个在制品的交付

时间就会缩短，而整个在制品的流速就会提高。同样，我们在做的事情越少，每件事情完成的周期越短，流动的速度就上升了，所以价值管理的核心其实就在于控制在制品、提高流速。控制在制品很重要，这里云层推荐大家看《凤凰项目：一个 IT 运维的传奇故事》这本书。

3.3.4　从批量生产到单件流

做了价值管理后，接着我们会思考如何把生产过程再做改变及优化。以前传统的生产过程是批量生产，就是 A 生产一批零件，然后给 B，B 生产一批零件，然后给 C，C 再生产一批零件，然后把 A、B、C 组合在一起。这时会出现问题，如果 A 里面有一小批零件是错误的，等到 C 做完我们再去集成的时候发现问题就太晚了。所以需要去改变整个交付过程，传统批量交付过程的缺点就是批量生产等待的过程太长，而如果变为单件流及时生产（Just In Time，JIT）的管理体系，即 A 生产了一个零件就给 B，B 接着生产再给 C，这个过程意味着最终用户等待单个交付的过程就很短了，如图 3-11 所示。

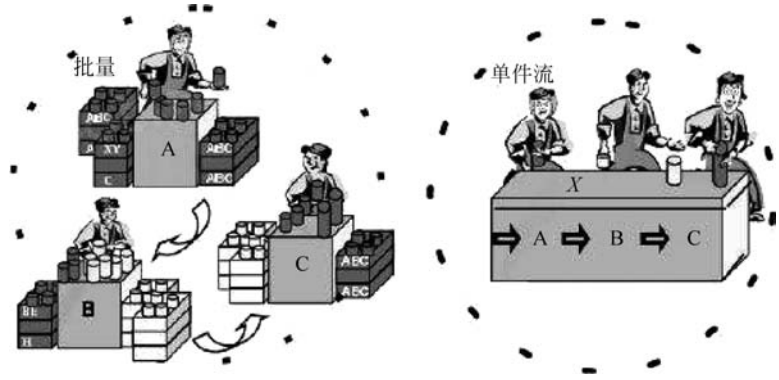

图 3-11　从批量交付到单件流

以前等待成批采购的商品要一个月，当转换成单件流时等待时间变短，在这个过程中单件流的核心是通过减少当前节点工作量来提高流动速度。在提高流动速度做完后还可以想一想做得好不好，如果不好，则可以在下个版本做得更好，这样就可以很快地改进产品了。

当然这里会面临成本控制的问题，因为批量生产相对成本较低。例如云层给几十个人上课就是一个批量的做法，好处就是成本比较低，坏处是无法为每个人的每个问题提供定制化解决方案。但是如果走单件流，给 A 同学一口气讲完这门课，大概 3 天就能讲完，并且还能针对 A 同学的情况做动态调整，下次再讲时这个课程就升级了。其实走单件流对于课程的提升、学习、用户交付是比较好的，因为你花 3 天时间就能学完，而现在走批量交付大概要学 3 个月，大家肯定说 3 天能学完为什么要等 3 个月？因为做单件流第一对个人要求很高；第二成本很高。

3.3.5 4个流动层次

除了单件流之外还有更高级的做法来提升流动速度，精益管理中的4个流动层次如图3-12所示。

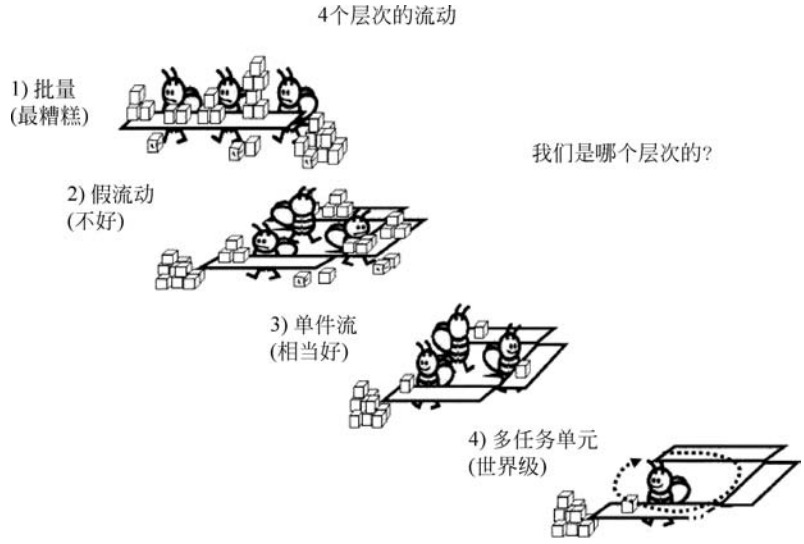

图3-12 4个层次的流动

最糟糕的是批量开发完后交给测试，测试完后交给运维去发布。其次是假流动，开发写几个模块后交给测试，测试测几个模块后交给运维，看起来好像是批量，但其实不够好，首先，流动的内容不是一个整体；其次，每个人负责的部分仍然是单工种的。比较好的情况是单件流，也就是一个需求一个功能点，一个代码块一个测试，最后只发这个功能。

微服务(Micro Service)的做法和单件流很类似，要求函数级别都模块化，这时只上一个小模块，互相影响的范围很少，但现实情况是大多数公司很少能实现单件流，因为对整个团队的能力要求太高，更不要说最后一个流动层次(多任务单元)了，也就是全栈。

云层觉得在当下全栈似乎越来越可能了，以前大家认为全栈是不可能做到的。例如云层想拍一个Vlog谈一下这个课程，以前要专门的摄影、专门的打光、专门的后期、专门写稿子的人，自己还要会表演、懂专业技术，要求很高，可能需要十几个人去做这件事情。但是现在我一个人就能做到，虽然不会化妆，但可以开美颜，虽然不会拍视频，但现在所有的手机应用都可以自己剪辑、拍摄，然后只要随便买个灯光打给自己就行了，所以现在的情况是工具智能化了，所以做到全栈是比较容易的，测试开发人员要做的事情就是多任务单元级别，即一个人负责整个流程，难点在于有没有合适的工具帮助你做到多专多能。

DevOps要求人人皆需求、开发、测试、运维，其实是可以做到的，因为工具可以逐步成

为服务单元为大家提供专业支撑。以前发布要考虑怎么布线、装服务器、重启服务器等，现在不需要了，买个亚马逊云（Amazon Web Services，AWS）或阿里云，然后发布直接打包上传就可以了，工具支撑你做完所有事情，所以现在可以做到多任务单元级别的内容，也就是全栈。其实全栈不是真正的全栈，而是你能够把所有资源整合在一起，管理起来。

现在的服务平台越来越多，大家在小团队做的事情也越来越多，不像以前做一件事情推动不起来，因为没有这些资源，但现在非常容易推动，所以现在都是小团队化。

大家看李佳琦这样的直播团队，一般也就 20 人以内。其实在我看来五六个人就够了，自己带货、一个配词、一个秘书化妆、一个懂点技术的人，进货、打单和财务都可以远程托管。大家会发现现在所谓的多任务单元级别在变，未来也许大家的工作都是在家接“云”活。

你的工作不会像以前似的在公司里只做一件事情，对公司来讲希望你做单件流的一部分或者多任务单元的一部分，也就是我把事情交给你，你能马上帮我解决。把多个外包公司串在一起去做这件事情，也就是说个人工作室或者多人工作室的形式会越来越多，垂直领域的专业团队会成为稀缺资源。

3.3.6　可视化价值

为了帮助管理整个流动的过程，需要有价值的可视化过程，可视化的最佳方案就是看板。通过看板将需求变成开发任务，再从开发任务展开到具体细节的子任务，从实现研发转到测试，从测试转到发布运维，这里需要有跟踪过程。

可视化端到端的价值交付从而实现前后职能拉通，能够实现以用户价值为驱动中心，做左右模块对齐，如图 3-13 所示。Scrum 体系中的每日站会也非常依赖于看板提供的整体信息，在后面的章节将做详细的介绍。

图 3-13　可视化价值

3.4 顺畅高质量交付有用价值的困难

促进价值流畅流动这个过程是通过消除浪费实现的。前文讲解了，第一，什么是价值；第二，如何做端到端的自动化；第三，如何去做过程减法。本节来讲一下常常做不到顺畅高质量交付的原因。

3.4.1 Why Not

很多公司遇到的第一个困难就是目标不一致，往往用户目标和交付目标是不一样的。例如云层想给大家讲的内容和大家想要听的内容不一定是完全对称的，很多人会说云层老师讲得太难了，或者想云层讲得接地气点，但在云层的角度来看是没有必要去接地气的，这就是需求和目标的不一致问题。你可能会发觉云层写的文章太深奥了或者不哗众取宠，不标新立异，看得人很少，导致的结果是一般懂的人都懂，不懂的人也不去看了，核心问题是目标不一致。

其实大家在工作中也会有这个问题，因为在公司里作为交付目标来讲，并没有以用户第一。什么叫没有以用户第一，如果为了用户云层就专门去讲测试开发了，但云层还是想讲点未来的东西，因为云层的目标在星辰大海而不在眼前一时，这就是价值目标不一样的地方。另外一个问题是考核的方法不一样，客户考核永远是以价值为目标，围绕 OKR 来做，但是公司考核一般围绕 KPI，也就是规定你做事情做了没有，如果你做了并且做得好我给你 KPI 考核。虽然现在主流的考核在往 OKR 转，但 OKR 也是把双刃剑，你要去考虑如何平衡是做短期价值交付还是做长期价值交付的内容。

最后一点，也是更关键的一点，即现在做不到关键目标一致，也就是能力不一致问题。客户由于不具备敏捷的认知，无法与交付团队形成共赢心态，导致与你的交付能力对应不上。用户总是希望大而全地交付，不接受小批量交付，客户不愿意跟你反复沟通，最后就会导致客户能力与你的交付能力不一致了。例如云层问大家下个星期想听什么课程，有些同学能提出自己要知道什么，也有同学做不到，因为还不知道自己该会什么，甚至还没有独立解决部分问题的能力。这时候会希望系统地听听老师讲的内容，然后构建个人的能力，进一步问老师该如何解决眼前的问题。

其实现在做敏捷及 DevOps 转型时这类能力不一致的情况也很多，客户有很多历史债，想解决这些问题，但是又不能接受或者推翻以前的架构、流程和人。就像现在孩子读书一样，老师要求的很多任务是要家长参与及协作的，你会觉得这不应该是学校的工作么，但其实你没有具备和老师相同的匹配能力，孩子的教育首先需要家长介入。

3.4.2　研发效能度量

有了以上知识，本节讲解研发效能。阿里的研发效能度量体系如图 3-14 所示。基本上一线互联网公司都有类似的定义，围绕持续发布能力、需求响应周期、交付吞吐率和交付过程质量及交付质量这 5 个点出发，后续章节会专门进行介绍。

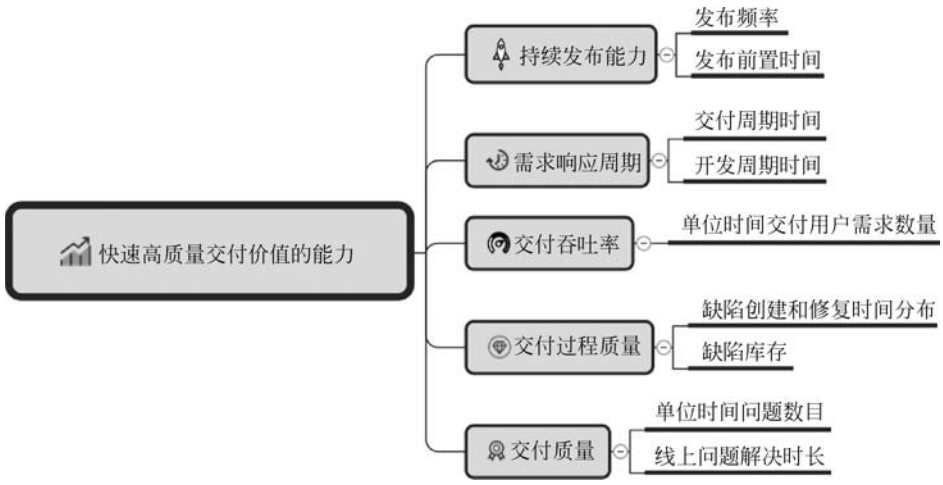

图 3-14　阿里研发效能度量体系

3.5　小结

本章主要围绕着以下 4 点来介绍：第一，价值一定要确定好，且要小批量交付价值；第二，加速流动效率，一切自动化；第三，过程可视化，优化过程；第四，构建整个研发效能的度量与团队统一能力。

3.6 本章问题

（1）当前 CI&CD 流程中哪些已经实现了自动化，哪些没有？

（2）当前团队的交付过程有哪些是可以优化的？

（3）如何帮助客户统一目标、价值和能力？