第3章

敏捷软件开发

扫码观看视频

本章首先介绍敏捷软件开发方法，包括敏捷软件开发宣言中的 4 个价值观和 12 条原则；接着具体讲述流行的敏捷软件开发方法 Scrum、Kanban 和 XP；之后再讲述 CI/CD 以及 DevOps 等相关方面的内容。

本章目标

- 了解“敏捷软件开发宣言”的 4 个价值观和 12 条原则
- 理解重要的敏捷软件开发实践，例如 Sprint、用户故事、Backlog
- 理解面向敏捷软件开发的 Scrum、Kanban、XP 方法
- 理解敏捷软件开发和 DevOps 是一种理念，而 CI/CD 是实现这两种理念的一种方法

敏捷软件开发是软件开发行业的一个大流行语，它是管理软件开发项目的一种不同方式。它不是一种特定的软件开发方法，而是一组基于敏捷软件开发宣言中表达的价值和原则的方法和实践的总称。

3.1 敏捷软件开发方法

随着计算机技术的迅猛发展和全球化进程的加快，软件需求常常发生变化，强烈的市场竞争要求更快速地开发软件，同时软件也能够以更快的速度更新。传统的方法在开发时效上时常面临挑战，因此，强调快捷、小文档、轻量级的敏捷软件开发方法开始流行。如今，“敏捷”已经成为一个非常时尚的词语。敏捷软件开发方法是一种轻量级的软件工程方法，相对于传统的软件工程方法，它更强调软件开发过程中各种变化的必然性，主张通过团队成员之间充分的交流与沟通以及合理的机制来有效地响应变化。

敏捷软件开发开始于“敏捷软件开发宣言”。在 2001 年 2 月，17 位软件开发方法学家在美国犹他州召开了长达两天的会议，制定并签署了“敏捷软件开发宣言”，该宣言给出了

4个价值观。

(1) 个体与交互高于过程和工具。

这并不是否定过程与工具的重要性,而是更加强调人与人的沟通在软件开发中的作用。因为软件开发过程最终还是要人来实施的,只有涉及软件开发过程的各方面人员(需求人员、设计师、程序员、测试人员、客户和项目经理等)充分地沟通和交流,才能保证最终的软件产品符合客户的需求。如果只是具有良好的开发过程和先进的过程工具,而开发人员本身技能很差,又不能很好地沟通,那么软件产品最终一样会失败。

(2) 可运行软件高于详尽的文档。

对用户来说,更多地通过直接运行程序而不是阅读大量的使用文档来了解软件功能。因此,敏捷软件开发强调不断地、快速地向用户提交可运行程序(不一定是完整程序),来让用户了解软件以及得到用户的认可。重要文档仍然是不可缺少的,能帮助用户更精准、全面地了解软件的功能,但软件开发的主要目标是开发出可执行的软件;敏捷软件开发并没有消除文档的使用,而是通过为开发人员提供完成工作所需的信息,例如用户故事(user stories),简化了文档的使用,即敏捷软件开发宣言仍看重文档记录的过程,但更重视可工作软件。

(3) 与客户协作高于合同(契约)谈判。

大量实践表明,在软件开发的前期,很少有客户能够精确完整地表达他们的需求,即便是那些已经确定下来的需求,也常常会在开发过程中改变。因此,靠合同谈判的方式将需求确定下来非常困难。对开发人员来说,客户的部分需求变更甚至会导致软件大范围的重构,而通过深入分析客户需求之后,有时还会发现通过适当调整需求就可以避免做出重大调整。而对于前者的情况,开发团队往往通过和客户谈判,撰写精确的需求合同来限制需求变更。但这会导致最终的软件产品功能与客户需求之间存在难以避免的差异,也会导致客户的满意度降低。因此,敏捷软件开发强调与客户的协作,通过密切的沟通合作而不是合同契约来确定客户的需求。

(4) 对变更及时响应高于遵循计划。

任何的软件开发都需要制订一个详细的开发计划,确定各项任务活动的先后顺序以及大致日期。然而,随着项目的进展,需求、业务环境、技术、团队等都有可能发生变化,任务的优先顺序和时间有时会面临必须的调整,所以,必须保证项目计划能够很好地适应这种难以预料的变化,并能够根据变化修订计划。例如,软件开发的后期,如果团队人员流失,那么如果时间允许,适当延后计划比补充新的开发人员进入项目要风险更小。

发表“敏捷软件开发宣言”的17位软件开发人员组成了敏捷软件开发联盟(agile software development alliance),简称“敏捷联盟”。他们当中有极限编程的发明者 Kent Beck、Scrum 的发明者 Jeff Sutherland 和 Crystal 的发明者 Alistair Cockburn。“敏捷联盟”为了帮助希望使用敏捷方法来进行软件开发的人们,定义了12条原则。

(1) 我们首先要做的是通过尽早和持续交付有价值的软件来让客户满意。

(2) 需求变更可以发生在整个软件的开发过程中,即使在开发后期,我们也欢迎客户对于需求的变更。敏捷过程利用变更为客户创造竞争优势。

(3) 经常交付可工作的软件。交付的时间间隔越短越好,最好2～3周一次。

(4) 在整个的软件开发周期中,业务人员和开发人员应该天天在一起工作。

(5) 围绕受激励的个人构建项目,给他们提供所需的环境和支持,并且信任他们能够完成工作。

(6) 在团队的内部,最有效果和效率的信息传递方法是面对面交谈。

(7) 可工作的软件是进度的首要度量标准。

(8) 敏捷过程提倡可持续的开发速度。责任人、开发人员和用户应该能够保持一种长期稳定的开发速度。

(9) 不断地关注优秀的技能和良好的设计会增强敏捷能力。

(10) 尽量使工作简单化。

(11) 好的架构、需求和设计来源于自身组织团队。

(12) 每隔一定时间,团队应该反省如何才能有效地工作,并相应地调整自己的行为。

最广泛使用的敏捷方法包括:

- Scrum;
- 精益软件开发;
- 极限编程(XP);
- 水晶(Crystal);
- 看板(Kanban);
- 动态系统开发方法;
- 特征驱动开发。

下面主要讲述 Scrum、看板和极限编程。

3.2 Scrum

3.2.1 概述

Scrum 作为敏捷软件开发的落地方法之一,用不断迭代的框架方法来管理复杂产品的开发,成为当前十分流行的敏捷管理方法。项目成员会以 1～2 周的迭代周期(称为 Sprint)不断地产出新版本软件,而在每次迭代完成后,项目成员和利益相关方再次碰头确认下次迭代的方向和目标。

Scrum 有一套独特且固定的管理方式,从角色、工件和不同形式的会议三个维度出发,来保证执行过程更高效。例如在每次 Sprint 开始前会确立整个过程:迭代规划、每日站会、迭代演示和回顾,并在 Sprint 期间用可视化工件确认进度和收集客户反馈。

1. Scrum 中的 3 种角色

(1) 产品经理:产品经理负责规划产品,并将研发这种产品的愿景传达给团队;需要整理产品需求清单(Backlog),关注市场需求的变化来调整产品需求优先级,确认下次迭代需要交付的功能;与团队、客户、利益相关方持续保持沟通和反馈,保证每位项目成员了解项目的意义和愿景。

(2) 敏捷专家(Scrum Master):敏捷专家帮助团队尽其所能地完成工作。例如,组织会议,处理遇到的障碍和挑战,与产品经理合作,在下次迭代前准备好 Backlog,确保团队遵循 Scrum 流程。敏捷专家对团队成员在做的事情没有指挥权,但对这一过程拥有指挥权。

例如，敏捷专家不能告诉某人该做什么，但可以提出新的 Sprint。

（3）Scrum 团队：Scrum 团队由 5～7 名成员组成。与传统的开发团队不同，成员们没有固定角色，例如，可能会由测试人员来做研发。团队成员之间相互帮助、共享成果，旨在完成全部的工作。Scrum 团队需要做好整体规划，并为每次迭代划分合适的工作量。

2. Scrum 会议

Scrum 会议包括整理产品需求清单、确定迭代规划、梳理产品需求清单等步骤，如图 3-1 所示。各步骤详细解释如下。

（1）整理产品需求清单：产品经理和 Scrum 团队进行碰头，基于用户故事和需求反馈来确定产品需求的优先级。Backlog 并不是待办事项列表，而是产品的所有功能列表。然后研发团队在每次迭代阶段去完成清单中一部分，最终完成整个项目。

（2）确定迭代规划：在每次迭代开始之前，产品经理会在迭代规划会议上和团队讨论优先级高的功能需求。然后确认有哪些功能将会在下次迭代时完成，并将这些功能从产品需求清单中移至迭代任务清单中。

（3）梳理产品需求清单：结束迭代后，产品经理需要和团队碰头来确认下次迭代的任务清单。团队可以利用这个阶段剔除相关度低的用户故事，提出新的用户故事，再重新评估故事的优先级，或将用户故事分成更小的任务。这次梳理会议的目的是确保产品需求清单里的内容足够详细，并且和项目目标保持一致。

（4）每日站会：每天花 15 分钟左右开一次站会，期间团队的每个成员都会讨论当前的进度和出现的问题。这个过程有助于团队保持日常联系。

（5）迭代演示：在每次迭代结束时，团队需要向产品经理报告已完成的工作，并做产品现场演示。

（6）迭代回顾：在每次迭代结束后，团队需要开例会总结使用 Scrum 进行研发带来的影响，并探讨在下次迭代中是否有能做得更好的地方。

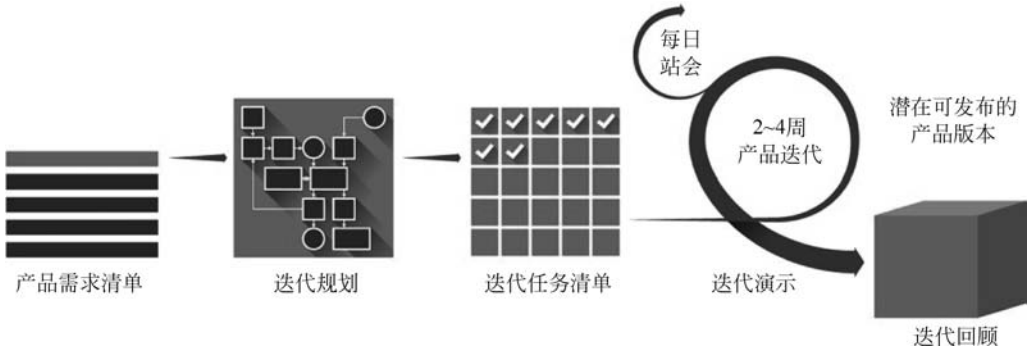

图 3-1　Scrum 会议

3. Scrum 项目所需的常用工件

（1）Scrum 任务板：用户可以用 Scrum 任务板使 Sprint 任务清单形象化。任务板可以用不同的形式来呈现，比较传统的做法有索引卡、便利贴或白板。Scrum 任务板通常分为三列：待办事项、在办事项和完成事项。团队需要在整个 Sprint 过程中不断更新。例如，如果某人想出新任务，他会写一张新卡并将其放入合适的位置。

(2) 用户故事：用户故事是从用户角度对软件提出功能的描述。它包括用户类型细分，用户想要什么以及为什么需要它。它们遵循相似的结构："作为＜用户类型＞，我希望＜执行某项任务＞以便我能＜实现某个目标＞"。团队根据这些用户故事进行研发来满足用户需求。

(3) 燃尽图：纵轴表示任务总量估计，横轴表示迭代时间。剩余工作量可以通过不同的点位或其他指标来表示。当事情没有按照计划进行并且影响后续决策时，燃尽图可以在这时给团队提个醒。燃尽图如图 3-2 所示。

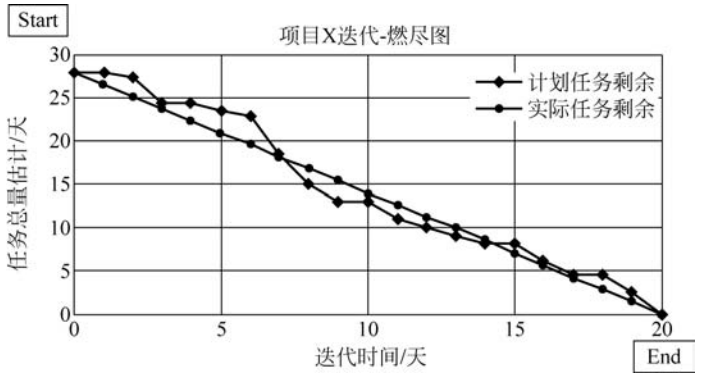

图 3-2 燃尽图

3.2.2 Sprint

Sprint(冲刺)是 Scrum 团队一起完成增量工作的实际时间段。对于一个 Sprint 来说，通常需要两周的时间，尽管有些团队认为一周的时间更容易确定，而一个月的时间更容易实现有价值的增长。敏捷专家 Dave West 建议，工作越复杂，未知数越多，Sprint 的时间应该越短。但这实际上取决于具体的团队，如果无法正常工作，可以进行更改。在此期间，如有必要，可以在产品经理和 Scrum 团队之间重新进行协商。

所有的事件——从计划到回顾——都发生在 Sprint 阶段。一旦一个 Sprint 的时长被确定，它就必须在整个开发期间保持一致。这有助于团队从过去的经验中学习，并将这种经验应用到未来的 Sprint 中。

3.2.3 每日站会

这是一种每天在同一时间(通常是上午)和地点举行的超短会议，以保持会议的简单性。许多团队试图在 15 分钟内完成会议，但这只是一个指导原则。每日 Scrum 的目标是让团队中的每个成员都保持同步，与 Sprint 目标保持一致，并为接下来的 24 小时制订计划。

一种常见的开站会的方法是让每个团队成员回答如下三个关于实现 Sprint 目标的问题：

- 我昨天做了什么？
- 我今天计划做什么？
- 有什么问题吗？

然而，我们可能会看到站会很快变成大家读自己昨天和今天的日历上的安排。站会的目的是，每天开会时，不会因为谈话让人分心，这样团队就可以一整天把注意力集中在工作上。所以，如果它变成了每天读日历安排，就要试图做出改变，以获得创造性。

3.2.4　用户故事

实际开发流程中，最重要的是做好用户故事的划分。用户故事是从用户的角度来描述用户渴望得到的功能。

1. 用户的 3 个要素

(1) 角色：谁使用这个功能。

(2) 活动：需要完成什么样的功能。

(3) 商业价值：为什么需要这个功能，这个功能能带来什么样的价值。

需要注意的是用户故事不能只使用技术语言来描述，要使用用户可以理解的业务语言来描述。

2. 3C 原则

用户故事的描述信息以传统的手写方式写在纸质卡片上，所以 Ron Jeffries 提出了如下 3C 原则。

(1) 卡片(Card)：用户故事一般写在小的记事卡片上。卡片上可能会写上对故事的简短描述，工作量估算等。

(2) 交谈(Conversation)：用户故事背后的细节来源于和客户或者产品经理的交流沟通。

(3) 确认(Confirmation)：通过验收测试确认用户故事被正确地完成。

3. INVEST 原则

好的用户故事应该遵循如下的 INVEST 原则。

(1) 独立性(Independent)：要尽可能地让一个用户故事独立于其他的用户故事。用户故事之间的依赖使得制订计划、确定优先级、工作量估算都变得很困难。通常可以通过组合用户故事和分解用户故事来减少依赖性。

(2) 可协商性(Negotiable)：用户故事的内容是可以协商的，用户故事不是合同。用户故事卡片上只是对用户故事的一个简短描述，不包括太多细节。具体细节在沟通阶段给出。如果用户故事卡片上带有了太多细节，实际上就限制了和用户的沟通。

(3) 有价值(Valuable)：每个故事必须对客户具有价值(无论是用户还是购买方)。让用户故事具有价值的比较好的方法是让客户来写下它们。一旦客户意识到这是用户故事，并不是契约而且可以进行协商时，他们将非常乐意写下用户故事。

(4) 可估算性(Estimable)：开发团队需要去估算用户故事以便确定其优先级和工作量，并安排计划。但是开发者难以估算故事，这是因为他们对于领域知识的缺乏(这种情况下需要更多的沟通)，或者是因为故事太大了(这时需要把故事切分成更小)。

(5) 短小(Small)：好的用户故事在工作量上要尽量少，最好不要超过 10 个理想人天的工作量，至少要确保能够在迭代或 Sprint 中完成。用户故事越大，在安排计划和工作量估算等方面的风险就会越大。

(6) 可测试性(Testable)：用户故事是可以测试的，以便于确认它是可以完成的。如果用户故事不能被测试，那么就无法知道它什么时候可以完成。

实际开发流程中，最为重要的是做好用户故事的划分。用户故事是从用户的角度来描述用户渴望得到的功能。

下面分析某团队的实际开发记录(项目“图书影视交流平台”)。在对需求进行初步分析后，首先需要条目化用户故事，并按优先级进行排序，之后按照用户故事为管理粒度进行开发、测试以及交付活动。团队划分了如表 3-1 的用户故事，包括其内容及优先级。

表 3-1　用户故事划分

用户	用 户 故 事	优先级
游客	• 展示搜索内容 • 搜索小组、帖子 • 注册	中
用户	• 撰写评价 • 评分 • 回复帖子 • 登录	高
	• 创建小组 • 点赞、评价 • 查看详细信息 • 修改个人简历 • 发表帖子 • 申请成为管理员	中
	• 添加图书、影音文件 • 退出小组 • 密码修改 • 加入小组 • 头像修改 • 置顶、精华帖子	低
管理员	• 管理用户 • 处理申请 • 处理举报	高

选取项目的部分用户故事描述，在此以表格的形式进行展示。

（1）创建小组如表 3-2 所示。

表 3-2　创建小组

用户故事标题	创 建 小 组
描述信息	作为用户，我想要创建小组，以便于小组之间进行影视的交流
优先级	中
重要程度	一般
预计工时	10 人时|1.25 人天

（2）展示搜索内容如表 3-3 所示。

表 3-3　展示搜索内容

用户故事标题	展示搜索内容
描述信息	作为用户，我想要获取搜索结果，以便于更好地发现内容
优先级	中
重要程度	一般
预计工时	4 人时|0.5 人天

3.2.5　Backlog

Backlog 是 Scrum 中经过优先级排序的动态刷新的产品需求清单，用来制订发布计划和迭代计划。使用 Backlog 可以通过需求的动态管理应对变化，避免浪费，并且易于优先交付对用户价值高的需求。

Backlog 的关键要点如下所述。

（1）清楚地表述列表中每个需求任务给用户带来的价值，作为优先级排序的重要参考。

（2）动态的需求管理而非“冻结”方式，产品经理持续地管理和及时地刷新需求清单，在每轮迭代前，都要重新筛选出高优先级需求进入本轮迭代。

（3）需求分析的过程是可迭代的，而非一次性分析清楚所有需求（只对近期迭代要做的需求进行详细分析，其他需求停留在粗粒度）。

3.2.6　结对编程

结对编程，即两个程序员肩并肩地坐在同一台计算机前合作编程，在一个程序员编程的同时，另一个负责检查代码的正确性和可读性。结对的程序员之间可以是动态调整的，但是结对必须经过缜密的思考和计划，因为多数程序员习惯了独自编码。通过结对，程序员通常可以更快地解决问题；由于两个程序员具有相同缺点和盲点的可能性要小得多，因此可出现更少的错误，可缩短测试的时间和降低测试的成本；程序员之间的互相激励、帮助和监

督,可降低编程的枯燥性和程序员懒惰的可能性。同时,由于软件中的任何一段代码至少有两位程序员非常熟悉,因此,个别的人员流动对项目进展造成的影响就会相对小。

3.3 看板

3.3.1 概述

看板(Kanban)作为可视化框架可以用于敏捷方法,能够清晰地向团队成员展示整个项目进度(要做什么、什么时候做、做多少)。当需要对系统进行小幅度改动的时候,团队成员可以采用看板方法来轻量化解决这个问题,因为看板本身并不需要额外去制订流程。

看板的灵感来源于丰田生产系统和精益生产。在20世纪40年代,丰田工程师Taiichi Ohno从超市库存管理的动态平衡中受到启迪,并借此建立对应的模型来改进其工程的工作流程。当货架空了,仓库管理员就会第一时间去补货和进货,这样能够随时满足客户的需求,同时不至于有过多的库存积压。因为始终保持供需平衡,因此也提高了库存管理的效率。

这些想法时至今日依然适用于软件团队和IT项目。在这种情况下,在制品(WIP)代替库存,只有看板上空了以后才能加入新工作。看板很好地将WIP数量与团队能力结合,从而达到生产过程中的动态平衡,提高了工作的灵活性、透明度和产出质量。

看板图是项目中实施看板的常见工具,如图3-3所示。依照传统,开发团队用一块白板和看板卡(便利贴或者白纸+磁铁)就能当看板图用了,便利贴代表着不同的工作。当然近年来项目管理软件工具已经能够在线创建看板了。

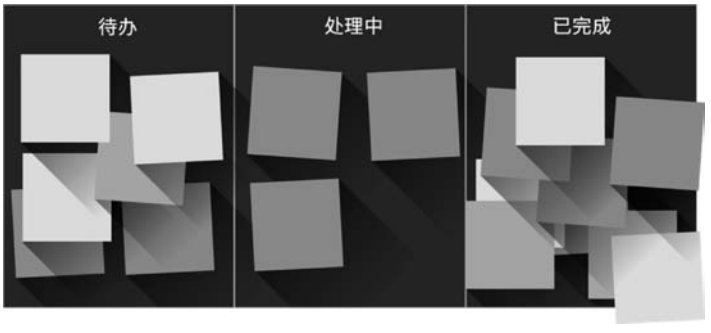

图3-3 看板图

无论用哪种形式来创建看板图,看板都有一个原则:划分为不同列来代表其工作状态。如最为常见的,一般分三列:待办、处理中、已完成。软件开发项目的分列可能包括待办、准备阶段、研发、测试、审批和已完成。

看板项目包括如下5条核心原则。

(1) 可视化工作流程:流程可视化可以帮助管理者清晰地了解整体情况和各项进度,尽早地发现其中的问题并及时进行改进。

(2) 限制WIP:WIP将确定看板图上每列的最大、最小工作量。通过对WIP进行限

制，开发者能够根据自己的意愿来调整速度、灵活度，提升解决高优先级需求的效率。

（3）管理和改进流程：团队需要对看板图上的流程进行定期监控和总结改进。

（4）制订明确的执行策略：为了防止在进行看板时发生协作变化，因此团队需要有明确的执行策略。每位成员都需要了解如何完成任务和“完成”的真正含义。

（5）持续改进：看板方法鼓励持续性的小幅度改进。一旦看板系统到位，该团队将能够识别和理解问题并提出改进建议。团队通过回顾总结工作流程和测量周期时间来评估其有效性，提高产出质量。

3.3.2　看板与 Scrum 的区别

看板与 Scrum 有所不同，看板对团队的个人能力要求较高，更灵活，适合新开发的产品，而 Scrum 适合成熟一些的产品和团队。其差异细节如表 3-4 所示。

表 3-4　看板与 Scrum 的区别

看　　板	Scrum
没指定定时迭代，可以分计划、发布、过程改进，可以事件驱动而不是设定时限	要求定时迭代
承诺不是必须的	团队在每个迭代承诺一定数量的工作
使用开发周期作为计划和过程改进的度量数据	以速度（velocity）作为计划和过程改进的度量数据
没有指定跨功能团队，也容许专门的团队	指定跨功能团队
没有指定工作任务大小	工作任务细分，可于一个迭代中完成
没有指定任何图表	指定使用燃尽图
设定 WIP 的限制（每个工作流程状态）	间接限制 WIP（每个迭代）
没有指定任何估算方式	规定估算过程
只要生产力容许，可以随时加工作任务	在迭代中不能加入新工作任务
多个团队和团员分享看板	由单一团队负责任务列表（Sprint Backlog）
没有指定任何团队角色	指定 3 个角色（产品经理、Scrum 专家、Scrum 团队）
看板反映持久开发情况	Scrum Board 在每个迭代后重设
优先级是非必须的	规定优先化的 Product Backlog

注：Scrum Board 是 Scrum 团队常使用的一个工具，它可将工作流程可视化，将工作流程分解为可管理的块——“故事”；Product Backlog 指一系列计划实现的产品功能/特性，或项目的交付物清单。

更具体来说，如果团队需要在某特定的时间发布或推广产品，以达到一定的市场预期的话，则团队一般会将需求进行拆分和细化，拆分为较小的需求后，团队可以通过检查每个 Sprint 的进度并进行调整，从而预测交付时间，进而确保整个项目成功交付，这时 Scrum 是首选的方式。其次，由于 Scrum 承诺在每个 Sprint 内不对计划做修改，如果团队经常会应对紧急情况或者修改任务的优先级，那么看板方法因其灵活的工作流程可以更好地适应。再者，在 Scrum 中每个 Sprint 的时间长度是固定的（2～4 周），并且每个 Sprint 结束后会交付潜在可交付产品的增量，如果项目需要有固定的交付时间（2～4 周），那么 Scrum 是比较好的选择。最后如果团队不足 5 人，在人员方面可能无法发挥 Scrum 的最大功效或存在一

定的浪费,那么建议使用看板方法。

在实际的小团队项目敏捷开发中,一般认为 Scrum 和看板都是不错的选择,且可视具体情况,灵活地调整迭代的周期,在两种模式上进行自定义的微调。

以开发某博客网站为例,网站的基本功能包括注册、登录,与博客内容相关的编辑、发布、收藏、评论、推荐、搜索以及与博客作者相关的查看、跟踪、搜索等。

若采用 Scrum 开发,团队首先与用户一起整理产品需求清单,接着将用户故事按照优先级排列。在第一次迭代前确定迭代规划,如第一次迭代计划实现注册、登录和博客的编辑、发布、收藏、评论功能,并且把每个用户故事划分为更小的用户故事,如将登录拆分为前端与后端的设计。然后实施迭代,在迭代的每天进行站会,沟通团队进度。迭代后进行迭代演示,并且梳理产品需求清单,确保产品需求清单里的内容足够详细。然后进行迭代回顾,总结有待改进的地方。然后重复前面的步骤,开始第二次迭代。

若采用看板开发,团队首先与用户确定用户故事,并且根据团队实际情况设置 WIP。若将开发过程分列为待办、准备、研发、测试、审批和已完成,在每轮迭代开始前,团队可以选取本轮迭代计划开发的用户故事,并可进一步将其划分为更小的用户故事,并严格设定需求优先级。如在第一次迭代实现博客内容的相关功能,编辑和发布的优先级最高,收藏与评论次之。然后,至少每天检查一次看板,将看板上的工作项在满足 WIP 的情况下在不同开发过程中调整顺序,若遇到 WIP 限制的情况,人员可灵活调整,如研发的人员可去帮忙测试,保证顺利开发,之后的迭代重复相同的步骤。看板与 Scrum 不同的是,在一次迭代中可以灵活添加用户故事,快速地响应需求变化。

尽管在前面提到了很多差异,团队同时采用 Scrum 和看板来帮助他们更有效率地工作的现象也是很普遍的。

3.4 极限编程

极限编程(exstream programing,XP)是一种实践性较强的规范化的软件开发方法,它强调用户需求和团队工作。利用 XP 方法进行软件开发实践的工程师,即使在开发周期的末期,也可以很快地响应用户需求。在团队工作中,项目经理、用户以及开发人员都有责任为提高软件产品的质量而努力。XP 特别适用于软件需求模糊且容易改变、开发团队人数少于 10 人、开发地点集中(例如一个办公室)的场合。

XP 包含了一组相互作用和相互影响的规则和实践。在项目计划阶段,需要建立合理和简洁的用户故事。在设计系统的体系架构时,可以采用 CRC(class,responsibility,collaboration)卡促使团队成员共同努力。代码的质量在 XP 项目中非常重要。为了保证代码的质量,可以采用结对编程以及在编码之前构造测试用例等措施。在测试方面,开发人员有责任向用户证明代码的正确性,而不是由用户来查找代码的缺陷。合理的测试用例及较高的测试覆盖率是 XP 项目测试所追求的目标。图 3-4 更加详细地描述了 XP 所推崇的规则和实践方法。

图 3-4　XP 所推崇的规则和实践方法

3.4.1　XP 的 4 个价值观

XP 的 4 个价值观介绍如下。

1. 交流

由于 XP 方法使用较少的文档，因此 XP 方法非常强调项目人员之间的沟通交流，尤其是直接面对面的交流。交流不仅能使相关人员更为精确地理解需求，还能够尽可能地避免因为需求变更导致的不一致。事实上，一些项目的失败就是项目相关人员沟通不到位导致

的,例如客户的需求变更没有准确及时地传递给涉及的开发人员,导致系统的不一致和集成的困难,而如果这一问题在软件交付时才发现,那么修正的代价可能惊人的高。因此,项目相关人员之间的充分沟通交流是极度重要的,尤其是对 XP 这种只有核心文档的项目来说。

2. 简单

简单是 XP 推崇的理念,一切都使用最简单、最小代价的方式达到目的,以及用最简洁的设计达到客户的要求。例如简单的过程(根据项目特点,对过程模型剪裁)、简单的模型(可使用任意模型,只要能达到目的即可)、简单的文档(只编写核心的、必需的文档)、简洁的代码设计实现等。该价值观体现了软件开发的"刚好够用"的思想,避免冗余繁杂。

3. 反馈

及时高效的反馈能够确保开发工作的正确性,并能够在发生错误时更及时地纠正偏差。例如,团队工作时,相关开发人员在一起工作,这样每个人的意见都能够在几分钟甚至几秒钟内得到反馈。而通过非正式的评审(如走查)也可在几分钟的沟通中得到反馈,相比正式的评审会议,这种方式显然省时得多,而且也更及时。

4. 勇气

Scott W. Ambler 在《敏捷建模:极限编程和统一过程的有效实践》一书中指出,"敏捷方法要求与其他人密切合作,充分信任他人,也信任自己,这需要勇气。XP 和敏捷建模(Agile Modeling,AM)等方法要求做能做到的最简单的事,相信明天能解决明天的问题,这需要勇气。AM 要求只有在绝对需要的情况下才创建文档,而不是只要觉得舒适就去创建,这需要勇气。XP 和 AM 要求让业务人员制订业务决策,如排定需求的优先级,而让技术人员制订技术决策,如软件如何去满足需求,这需要勇气。AM 要求用尽可能简单的工具,如白板和纸,除非复杂的建模工具能够提供更高价值时才去使用它们,这需要勇气。AM 要求不要为了推迟困难任务(如需要使用代码来验证模型)而把大量时间浪费在图的加工上,这需要勇气。AM 要求信任同事,相信程序员能制订设计决策,因此不需要给他们提供过多的细节,这需要勇气。AM 要求必胜的信心,去结束 IT 产业中接近灾难和彻底失败的循环,这需要勇气"。同时,"需要勇气来承认自己是会犯错误的,需要勇气来相信自己明天能克服明天出现的问题"。

3.4.2 XP 的 12 个核心实践

XP 的 12 个核心实践介绍如下。

1. 完整的团队

使用 XP 方法时,项目组的所有成员最好在同一个场所工作,以便及时沟通和解决问题。同时,项目组中要求有一个现场用户,由其提出需求并制订需求优先级,编写验收测试用例。

2. 计划对策

有两个计划是必需的:发布计划和迭代计划。计划是根据业务需求的优先级和技术评估来制订的,优先实现高优先级和技术难度低的需求,而低优先级和高技术难度的需求则可以根据情况调整到后续计划中实现,这样可以尽可能地保证项目顺利进行以及给有难度的

技术争取更多的时间。制订的计划常常是可调整的，因为随着项目的进行，难免会出现一些前期无法预料的事情，这时就要根据具体情况适当修正计划。

3. 系统隐喻

系统隐喻是对待开发软件系统的一种形象化隐喻，这种隐喻描述了开发人员将来如何构建系统，起到概念性框架的作用。这种隐喻必须是团队成员所共同熟悉的。

4. 小型发布

需要经常、不间断地发布可运行的、具有商业价值的小软件版本，以供用户使用、评估和及时反馈。

5. 测试驱动

XP 方法推荐在编写代码之前优先编写测试代码，这样开发人员可以在开发中快速地检验自己的代码是否正确实现了功能。

6. 简单设计

系统的设计应该尽可能简洁，刚刚好满足当前定义的功能最好，简单、易懂、无冗余、能通过所有的测试，没有重复混乱的逻辑，正确完整地实现了开发人员的意图，同时尽可能少地使用类和方法。设计还应符合系统隐喻，以便于以后对系统重构。

7. 结对编程

XP 方法强烈推荐的一个核心实践是结对编程，可参看 3.2.6 节。

8. 设计改进

在整个开发过程中，需要对程序的结构和设计不断地评估和改善。在不改变外部可见功能的情况下，按照高内聚、低耦合的原则对程序内部进行改进，力求代码简洁、无冗余。

9. 持续集成

持续集成是指完成一个模块的开发和单元测试之后，立即将其组装到系统中进行集成测试。而且必须完成本次集成才能继续下一次集成。这样虽然集成的次数会增加很多，但保证了每一个完成的模块始终是组装完毕、经过测试和可执行的。

10. 集体所有权

集体所有权是指团队中的任何人都可以在任何时候修改系统任何位置上的代码。这是建立在小系统开发的前提下，由于团队的成员都可以参与模型的开发，又有系统隐喻，因此，基本上每一个成员都对系统有一定程度的了解。而且，结对编程、编码标准、持续集成等方法都为集体所有权提供了支持，能及时尽早地发现代码中的缺陷和错误，避免在后期集中爆发。

11. 编码标准

很多软件开发模型都强调编码标准，XP 也不例外。为编码制定统一的标准，包括代码、注释、命名等，使得代码在整体上呈现一致性，为日后维护提供极大的便利。

12. 工作安排

XP 方法要求团队中的每位成员都时刻保持充沛的精力投入到项目中，长时间、超负荷的工作会使工作效率极大地下降，因此，XP 方法建议采用每周 40 小时工作制，如果加班，也不得连续超过 2 周。

3.5 CI/CD

3.5.1 CI/CD 概述

CI/CD 是一套使软件开发的构建、测试和部署阶段自动化的方法。自动化可缩短交付时间,并提高整个开发生命周期的可靠性。

CI/CD 中的 CI(Continuous Integration)代表持续集成。CD(Continuous Delivery 或 Continuous Deplayment)是指持续交付和持续部署,具体取决于团队选择如何促使代码更改以进行生产。

持续集成和持续交付是 CI/CD 中的两个不同流程,具有不同的用途。

(1) CI 完成自动构建和测试步骤,以确保代码更改能够可靠地合并到代码仓库中。

(2) CD 提供了向最终用户交付代码的快速无缝方法。

CI/CD 的目标是帮助开发人员以速度和效率交付软件。团队不断将代码交付到生产中,运行新功能和错误修复的持续流程。CI/CD 概览如图 3-5 所示。

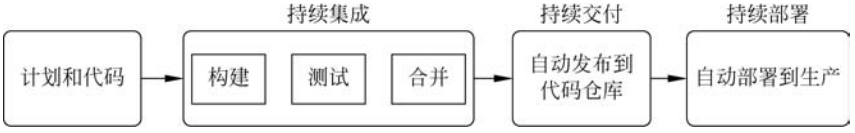

图 3-5 CI/CD 概览

1. 持续集成(CI)

持续集成是不断地将更新集成到代码仓库的方法,即团队成员经常集成他们的工作。CI 提供一个一致的自动化流程,包括构建、测试和合并新软件。

通过 CI,开发人员可将代码进行更改(无论是校正还是更新)提交到代码仓库中。更改总是很小,因此很容易跟踪。每个新集成都会触发自动生成和测试序列。此过程可向开发人员提供快速反馈,并通知每一个错误。

理想的情况下,CI 反馈回路不应超过 10 分钟。构建和合并应尽可能频繁地进行,最好是每天多次。CI 是拥有众多独立从事复杂应用程序工作的开发人员的大项目的理想之选。自动构建可使团队避免合并冲突、错误和重复工作。

2. 持续交付(CD)

持续交付是指持续地将各类更改(包括新功能、缺陷修复、配置变化等)安全、快速、高质量地落实到生产环境或用户手中的能力。

持续交付从持续集成结束的地方开始。CD 使开发人员能够随时向不同的环境和最终用户部署常规软件的更改(新功能、改进、错误修复)。

所有进入 CD 过程的代码必须首先通过 CI。

较小的、较频繁的软件发布,破坏性较小,在出问题时更容易排除故障或回滚(指的是程序或数据处理错误,将程序对数据恢复到上一次正确状态的行为)。开发团队还能够快速提供新功能,帮助公司更好地满足客户需求。

3. 持续测试

持续测试作为软件持续集成中的重要组成部分,为软件项目的成功提供了保证软件质

量持续改进的重要手段。持续测试是运行自动化测试的方法，而代码更改则通过 CI 和 CD 进行。单个 CI/CD 过程可以具有如下的多种类型的测试：

- 单元测试（确保单个功能在构建过程中正确执行的 CI 测试）；
- 集成测试（检查组件和服务是否能协同工作）；
- 功能测试（确保功能按团队预期执行）；
- 验收测试（性能、可扩展性、应力、容量等）；
- 静态代码分析（检查语法问题和漏洞）；
- 自动测试（如 API 测试和安全测试）。

并非每个 CI/CD 过程都需要有所有这些测试，但持续测试的目标始终相同。持续测试会快速发现错误，并在错误导致生产问题之前提醒团队。

3.5.2　CI/CD 管道

CI/CD 管道是所有软件在其开发生命周期中遵循的可运行、秩序渐进的路径。典型的管道可构建代码、运行测试并安全地部署新版本的应用程序。CI/CD 管道如图 3-6 所示。

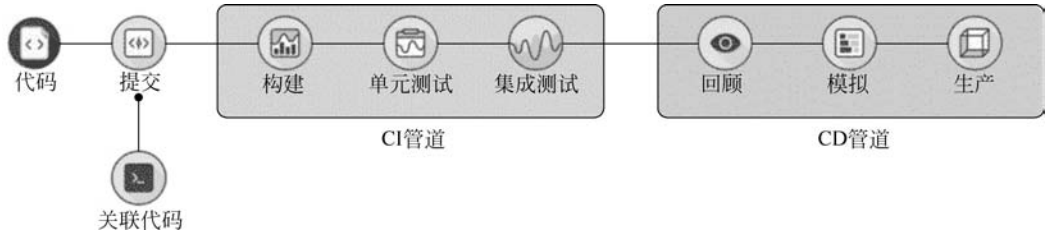

图 3-6　CI/CD 管道

自动化管道为团队提供了多项如下的优势：

- 快速部署新软件更新；
- 可靠的构建和测试流程；
- 最终在生产中产生更少的错误；
- 所有代码的更改、测试和部署的透明日志。

CI/CD 管道不会改变开发人员创建软件的方式。如果没有自动管道，开发人员仍需要手动完成相同的步骤。不过，手动方法的效率较低，因为团队必须更加专注于重复任务和修复错误，而不是编写软件。CI/CD 管道的一个例子如图 3-7 所示。

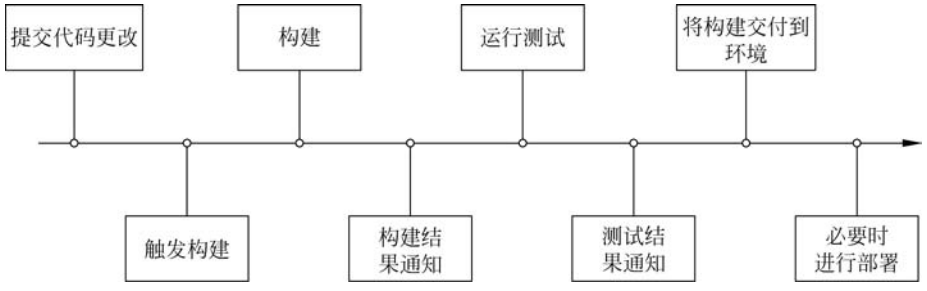

图 3-7　CI/CD 管道的一个例子

典型的 CI/CD 管道分为 4 个主要阶段：提交、构建、测试和部署，如表 3-5 所示。

表 3-5 典型的 CI/CD 管道的 4 个主要阶段

提交阶段	构建阶段	测试阶段	部署阶段
代码出库在提交后触发管道	第二阶段在初始测试结束后开始	此阶段是一个安全网，可防止异常错误到达最终用户	如果代码更改通过测试阶段，管道将启动最终部署阶段
将新功能和更新与代码库合并	如有必要，该过程将编译程序（典型的 Java、C/C++和 Go 代码），管道将构建容器	自动化测试验证代码的正确性	应用程序上线
开发人员会获得有关新代码质量的反馈	管道将代码和依赖关系相结合，以创建软件的可运行实例	自动化测试检查产品的行为	除了为最终用户保留的生产环境外，通常还有多个部署环境
CI/CD 工具运行单元测试和集成测试，以检查潜在的问题	如果此步骤失败，则代码（或其依赖关系）存在问题，开发人员必须在继续之前解决问题	管道向开发人员提供反馈，并报告新代码更改的状态	实时监控可确保新功能按预测执行

一个常见的做法是在每个阶段的末尾设置通知。关键时刻的警告和报告可使团队成员保持同步。

测试阶段往往是不同管道之间变化最大的。根据项目大小，测试阶段可以持续数秒到数小时。一些大型项目甚至分多个阶段运行测试。在该设置中，测试通常并行运行以节省时间。

较复杂的管道除了 4 个核心管道外，还有其他步骤，如数据同步、应用程序和库修补或存档信息资源等。

3.5.3 CI/CD 的优势

1. 更快、更可靠的版本发布

CI/CD 加快了开发过程，使产品能够快速到达用户手中。管道还可以降低编写代码时的风险，使开发人员能够专注于编码而不是修复错误。

2. 更高的可见性

CI/CD 管道允许团队详细分析构建和测试的结果。更高的透明度揭示了开发流程中潜在的可以改进的地方。开发人员可以轻松跨越各个阶段，发现效率低下的地方，并优化流程以提高生产力。

3. 早期错误检测

多种类型的自动化测试使得可在早期阶段发现大多数错误问题。在自动化测试和手动质量保证（QA）检查之间，采用 CI/CD 很少会在最后时刻检查出新的错误。

4. 快速反馈循环

快速更新可以让用户不断地反馈。可利用 A/B 测试（一种将网页或应用程序的两个版

本相互比较以确定哪个版本的性能更好的方法)中的用户输入测试功能,并与用户一起试验产品的早期版本。

5. 更快乐的开发和运维团队

CI/CD 允许开发人员以更少的手动任务和更少的纠错来促使代码更改。标准化的环境、交付过程中的测试、单独的环境变量和自动回滚,可使得运维团队享有稳定性。

许多软件工具都支持实现 CI/CD 方法。这些工具包括仓库管理工具,例如 Github 和 Bitbucket,用于自动化构建的 Jenkins,以及用于自动化测试的 Selenium。

3.6　DevOps

根据定义,DevOps 概述了一种在软件开发过程和组织文化的转变,通过自动化与整合开发团队和 IT 运作团队的尝试,加快交付高质量的软件——这两个团队在传统上是相互分离的或者是独立的。在实践中,DevOps 流程和文化超越了开发和运维,将所有应用程序的利益相关者的投入——包括平台和基础设施工程、安全性、遵从性、治理、风险管理、业务线、最终用户和客户——纳入软件开发生命周期。

DevOps 代表了过去 20 多年来软件交付周期的发展现状,从每隔几个月甚至几年发布一次庞大的应用程序范围内的代码,到每天甚至每天发布几次迭代的更小的产品特性或功能更新。

最终,DevOps 是为了满足软件用户对创新的功能,不间断的性能和可用性不断增长的需求。

直到 2000 年之前,大多数软件都是使用瀑布方法开发和更新的,瀑布方法是大规模开发项目的一种线性方法。软件开发团队将花费数月时间开发影响大部分或全部应用程序的大量新代码。由于更改非常广泛,他们往往要花几个月的时间将新代码集成到代码仓库中。

接下来,质量保证(QA)、安全和运维团队将花费几个月的时间来测试代码。结果与软件发布之间的间隔是数月甚至数年,并且通常在发布之间也会有几个重要的补丁或错误要进行修复。这种“大爆炸”式的功能交付方式的特点是复杂而有风险的部署计划,与上游和下游系统难以安排的联动,以及 IT 运维“非常希望”业务需求在生产“上线”前的几个月没有发生巨大的变化。

为了加速开发和提高质量,开发团队开始采用敏捷软件开发方法,这种方法是迭代的,而不是线性的,并且专注于对应用程序代码库进行更小、更频繁的更新。这些方法中最主要的是持续集成和持续交付(CI/CD)。在 CI/CD 中,较小块的新代码每一两周内就会合并到代码仓库中,然后自动集成、测试并准备部署到生产环境中。敏捷将“大爆炸”方法演变成一系列“小冲击”,这些“小冲击”也分担了风险。

这些敏捷开发方法越有效地加速软件开发和交付,它们就越容易将仍然孤立的 IT 运维(系统供应、配置、验收测试、管理、监控)暴露为软件交付生命周期中的下一个瓶颈。

所以 DevOps 是从敏捷发展起来的。它添加了新的过程和工具,这些新的过程和工具将 CI/CD 的持续迭代和自动化扩展到软件交付生命周期的其余部分。在流程的每一个步骤中,它实现了开发和运维之间的密切协作。

3.6.1 DevOps 生命周期

DevOps 生命周期如图 3-8 所示。

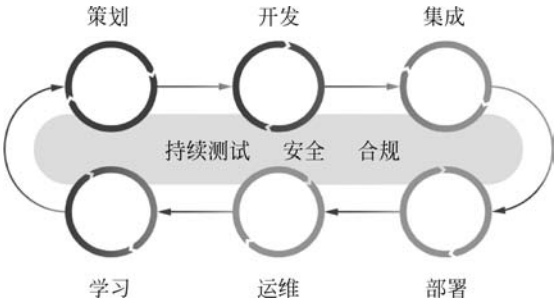

图 3-8 DevOps 生命周期

DevOps 生命周期(以线性方式描述,有时称为持续交付管道)是一系列迭代的、自动化的开发过程或工作流程,在一个更大的、自动化的、迭代的开发生命周期中执行,旨在优化高质量软件的快速交付。工作流程的名称和数量可以根据所询问的对象而有所不同,但通常可以归结为以下六种。

(1) 策划(或理念)。在这个工作流程中,团队从优先处理的最终用户反馈和案例研究,以及所有内部的利益相关者的意见,确定下一个版本中的新特性和功能。策划阶段的目标是通过生成功能储备来最大化产品的业务价值,这些功能在交付时将产生具有价值的期望结果。

(2) 开发。这是编程步骤,开发人员在基于待办事项列表中的用户故事等,测试、编写代码,并构建新的增强特性。方法的组合非常普遍,如测试驱动开发(TDD)、结对编程和同行代码审查,以及其他一些是常见的。开发人员经常使用他们的本地工作站来执行编写和测试代码的"内部循环",然后再将代码发送到持续交付的管道中。

(3) 集成(或构建,或持续集成和持续交付)。如上所述,在此工作流程中,新代码被集成到现有代码仓库中,然后测试并打包到可执行的部署中。常见的自动化活动包括将代码更改合并为"主"副本,从代码仓库检出该代码,以及自动编译、单元测试和打包到可执行的文件中。最佳方法是将 CI 阶段的输出存储在二进制代码仓库中,以备下一个阶段使用。

(4) 部署(通常称为持续部署)。在这里,运行时构建输出(来自集成)被部署到一个运行时环境中——通常是一个开发环境,在那里执行运行时测试,以保证质量、合规和安全。如果发现了错误或缺陷,开发人员就有机会在任一最终用户看到它们之前将其纠正和弥补。通常有用于开发、测试和生产的环境,每个环境都需要逐步"更严格"地质量检验。部署到生产环境的一个好方法通常是,首先部署到最终用户的子集,然后在确定稳定后最终部署到所有用户。

(5) 运维。如果说将功能交付到生产环境是"第一天",那么一旦功能在生产中运行,就会发生"第二天"的运维。监控功能的性能、行为和可用性,应确保功能能够为终端用户提供增值服务。运维部门应确保功能顺利运行,并确保服务没有中断——通过确保网络、存储、平台、计算和安全状况都是健康的。如果出了问题,运维部门应确保事件被识别,适当的人

员被提醒，问题被确定，并实施修复。

（6）学习（有时也称为持续反馈）。学习是指收集来自最终用户和客户关于特性、功能、性能和业务价值的反馈，以便用于策划下一个版本的增强功能和特性。这也将包括从运维活动中获得的任何学习和待办事项，使开发人员能够主动避免任何过去已经发生并且可能在未来再次发生的事件。

如下的另外三个重要的连续工作流程发生在这些工作流之间。

（1）持续测试：经典的 DevOps 生命周期包括一个离散的“测试”阶段，它发生在集成和部署之间。然而，DevOps 已经进步了，某些测试元素可以发生在策划（行为驱动开发）、开发（单元测试、合约测试）、集成（静态代码扫描、通用漏洞披露（CVE）扫描、验证代码质量）、部署（冒烟测试、渗透测试、配置测试）、运维（混沌测试、合规测试）和学习（A/B 测试）。测试是识别风险和漏洞的一种强大方式，并为 IT 运维提供了接受、减轻或补救风险的机会。

（2）安全：虽然瀑布模型和敏捷软件开发在交付或部署后就开始关注安全工作流程，但 DevOps 从一开始（策划阶段）就努力整合安全——当安全问题最容易解决且成本最低时——并持续将之贯穿于开发周期的其余部分。这种安全方法被称为向左移动（看一下图 3-8 就会更容易理解）。

（3）合规：合规（治理和风险）也要在开发生命周期的早期和整个过程中得到最好的处理。受监管的行业通常被要求提供某种程度的可观察性、可跟踪性和对功能在运行时运维环境中如何交付和管理的通道。这需要在持续交付管道和运行时环境中策划、开发、测试和实施策略。审核合规措施对于向第三方审核员证明合规是极其重要的。

3.6.2　DevOps 文化

人们普遍认为，DevOps 方法如果没有对 DevOps 文化的承诺就不起作用，DevOps 文化可以概括为软件开发的一种特别的组织和技术方法。

在组织层面上，DevOps 要求所有软件交付利益相关者——软件开发团队和 IT 运维团队，以及安全、合规、治理、风险和业务线团队——之间持续沟通、协作和分担责任，以便快速和持续地进行创新，并从一开始就将质量目标纳入软件考虑的范围。

在大多数情况下，实现这一目标的最佳方式是打破这些孤岛，并将其重组为跨职能的、自主的 DevOps 团队，这些团队可以从头到尾从事代码项目，从策划到反馈，而不需要向其他团队交接或等待批准。当放在敏捷软件开发的背景下，共同的责任和协作是拥有一个共同的着眼于有价值结果的产品的基石。

在技术层面上，DevOps 要求致力于自动化，使项目在工作流程内和工作流程之间不断前进，并致力于反馈和度量，使团队能够不断加快周期，提高软件质量和性能。

3.6.3　DevOps 工具

DevOps 和 DevOps 文化对支持异步协作、无缝集成 DevOps 工作流程和尽可能地自动化整个 DevOps 生命周期的工具提出了要求。DevOps 工具的类别包括以下 7 种。

（1）项目管理工具：使团队能够建立形成编码项目的用户故事（需求）的待办事项列

表,将其分解为更小的任务,并跟踪任务的完成情况。许多工具支持敏捷项目管理实践,如Scrum、精益软件开发和看板,这些都是开发人员带到 DevOps 中的。流行的开源选项包括 GitHub Issues 和 Jira。

(2) 协作式代码仓库:版本控制的编码环境,允许多个开发人员在同一代码仓库中工作。代码仓库应该与 CI/CD、测试和安全工具集成,这样当代码被提交到库中时,它可以自动进入下一个步骤。开源代码仓库包括 GiHub 和 GitLab。

(3) CI/CD 管道:实现代码检查、构建、测试和部署自动化的工具。Jenkins 是这个类别中较流行的开源工具;许多以前的开源替代品,如 CircleCI,现在只提供商业版本。说到持续部署工具,Spinnaker 是一个开源的、多云的连续交付平台,可帮助使用者快速而自信地发布软件更改。Argo CD 是另一个流行的 Kubernetes 原生 CI/CD 的开源选择。

(4) 测试自动化框架:包括软件工具、库和最佳实践,用于自动化的单元、合约、功能、性能、可用性、渗透和安全测试。这些工具支持多种语言;有些使用人工智能(AI)来自动重新配置测试,以响应代码的更改。测试工具和框架的范围很广,流行的开源测试自动化框架包括 Selenium、Appium、Katalon、Robot Framework 和 Serenity(以前被称为 Thucydides)。

(5) 配置管理(基础设施即代码)工具:这些工具使 DevOps 工程师能够通过执行脚本来配置和提供完全版本化和完全文档化的基础设施。开源选项包括 Ansible(Red Hat)、Chef、Puppet 和 Terraform。Kubernetes 为容器化的应用程序执行配置管理的功能。

(6) 监控工具:这些工具能够帮助 DevOps 团队识别和解决系统问题;它们还能实时收集和分析数据,以揭示代码更改如何影响应用程序性能。开源监控工具包括 Datadog、Nagios、Prometheus 和 Splunk。

(7) 持续反馈工具:收集用户反馈的工具,可以通过热图(记录用户在屏幕上的操作)、调查或自助式问题单来进行。

3.7 敏捷软件开发、CI/CD 和 DevOps

敏捷软件开发和 DevOps 的理念相似,都是为了更好、更快地发布产品,但它们的理念又不完全相同,而 CI/CD 是实现这两种理念的一种方法。

在传统开发中要做好一个产品,大部分精力都要花在前期,如做更多调研,收集更多信息,撰写更多文档;但这种做法的缺点是,始终走在市场后面,无法紧跟潮流,做出的产品容易被淘汰。

敏捷开发的核心是,拥抱变化与快速迭代。

持续集成、持续交付和持续部署提供了一个优秀的 DevOps 环境,对于整个团队来说,好处与挑战并存。无论如何,频繁部署、快速交付以及开发测试流程自动化都将成为未来软件工程的重要组成部分。

DevOps 之所以逐渐流行起来,是因为以下几个因素。

(1) 容器技术开始成熟,特别是 Docker、Kubernetes 等技术的大行其道。

(2) 微服务架构技术的广泛使用。

微服务是支撑 DevOps 方法的手段,传统开发是在一个服务器里面,把各种元素装在一起组合成一个程序,但微服务是每一个服务是一个单独的单元,可以部署在不同的服务器上,通

过面向服务架构(service-oriented architecture,SOA)的方法把它们连接起来,再提供整个功能。

微服务是由一个个团队组成,每个团队都有自己的服务,做好后,可以独立地进行测试、开发、部署,然后将整个应用组合到一起。

(3) 敏捷软件开发流程的深入人心。

诸如 Scrum,、Kanban 和 XP 等敏捷方法被团队广泛使用,TDD、BDD、DDD 这些测试驱动设计、行为驱动设计、域驱动设计等设计方式的采纳,CI 和 CD 这些持续集成和持续部署等方式的实施,这些都是对 DevOps 的强烈需求。

(4) DevOps 带来的变革。

① 角色分工:打破传统团队隔阂,让开发、运维紧密结合,高效协作。

② 研发:专注研发、高度敏捷、持续集成。

③ 产品交付:高质量、快速、频繁、自动化、持续交付。

3.8　敏捷软件开发实例

【例 3-1】 青年租房管理系统。

青年租房管理系统为方便广大青年住房需求而设计,用户可以使用该系统浏览房屋信息并与租主取得联系,最终完成租房流程。系统至少包括如下功能。

(1) 租客注册:绑定手机(或邮箱)并设置个人资料。

(2) 顾客租房:租房包括短租与长租,房间类型有单人间、双人间与四人间。

(3) 租房流程为:租客选择长租或短租(短租按天租,长租按月租),查询并选择可租房间,提交租房申请后,客服人员对顾客资料进行审核并决定是否通过,通过后,短租需租客立刻提交租金,长租则系统自动生成合同,租客导出打印后双方线下签订,顾客每月最后一星期提交租金,系统自动以邮件的方式提醒。

(4) 租客可查询与管理订单,并可进行长租续约。

(5) 租客报修与投诉:租客通过上传图片和文字描述的形式进行报修与投诉。

(6) 客服管理租客:客服可查询租客并修改租客资料。

(7) 客服管理合同:客服可查询与管理合同。

(8) 客服房间管理:除了线上预约租房外,租客还可当面与前台租房,客服查询可用房间并帮助租房,若顾客未注册须先在平台注册,客服还可添置新房源与暂停出租问题房。

(9) 客服管理工单:客服对报修建立工单并安排师傅,再对投诉进行回复。

(10) 师傅处理工单:师傅由客服人员线下审核并导入,师傅接到工单后进行线下处理,处理完成后回复工单,租客可对师傅进行评价。

【解析】

1. 需求分析

系统的用户包括租客、客服、师傅,其中租客需求优先级最高,客服需求优先级次之,师傅需求优先级最低。在租客需求中,租房与订单查询优先级最高,而注册、报修等优先级次之。两次迭代中,可在第一次迭代中优先实现租客需求,在第二次迭代实现客服和师傅的需求。由于租客租房和订单查询涉及客服管理合同以及房间管理,而租客报修又涉及客服管理工单和师傅处理工单,实际上可在两次迭代先后实现租房相关需求和报修等其他需求。

以上为笔者个人分析,实际开发应由团队讨论并根据客观条件共同决定。

2. 编写用户故事和制订迭代计划

1) 编写用户故事

下面分析某个团队的实际开发记录。首先,团队划分了如表 3-6 所示的用户故事、内容及优先级。其中,斜体加粗为该团队自增额外需求,加删除线并加粗为最后未实现需求。

表 3-6 用户故事、内容及优先级

用 户	用户故事	优先级
租客	注册 登录 查询房源 租房 ***地图选房***	高
	VR 看房 查询订单 管理订单	中
	报修投诉 评价师傅 **用户论坛**	低
客服	审核租房申请 管理租客 管理合同	高
	查询可用房源 修改房源 **辞退师傅**	中
	审核并导入师傅 回复投诉 管理工单	低
师傅	查看工单 回复工单	低

具体的部分用户故事内容,如下所述。

(1) 注册,如表 3-7 所示。

(2) 登录,如表 3-8 所示。

表 3-7 注册

用户故事标题	注 册
描述信息	作为未注册的租客 我想要使用邮箱注册账号 以便使用租客客户端完整功能
优先级	高
重要程度	关键
预计工时	9 人时|1.13 人天

表 3-8 登录

用户故事标题	登 录
描述信息	作为已经注册的用户 我想要使用账号进入系统 以便正常使用租客系统功能
优先级	高
重要程度	关键
预计工时	2 人时|0.25 人天

（3）查询房源，如表 3-9 所示。

（4）租房，如表 3-10 所示。

表 3-9　查询房源

用户故事标题	查 询 房 源
描述信息	作为已经注册的租客 我想要通过相关的地理位置和价格筛选 以便找到合适房源
优先级	高
重要程度	关键
预计工时	4 人时|0.5 人天

表 3-10　租房

用户故事标题	租　　房
描述信息	作为已经登录的租客 我想要选择租用的房间 以便租房
优先级	高
重要程度	重要
预计工时	4 人时|0.5 人天

（5）查询订单，如表 3-11 所示。

（6）管理订单，如表 3-12 所示。

表 3-11　查询订单

用户故事标题	查 询 订 单
描述信息	作为已经登录的租客 我想要检索自己租房订单 以便于查询订单
优先级	中
重要程度	重要
预计工时	2 人时|0.25 人天

表 3-12　管理订单

用户故事标题	管 理 订 单
描述信息	作为已经登录的租客 我想要删除租房订单 以便对个人隐私进行保护
优先级	中
重要程度	重要
预计工时	2 人时|0.25 人天

（7）报修投诉，如表 3-13 所示。

（8）评价师傅，如表 3-14 所示。

表 3-13　报修投诉

用户故事标题	报 修 投 诉
描述信息	作为已经登录的租客 我想要申请房间报修以及投诉 以便及时处理问题并反映问题
优先级	低
重要程度	重要
预计工时	4 人时|0.5 人天

表 3-14　评价师傅

用户故事标题	评 价 师 傅
描述信息	作为已经登录的租客 我想要评价师傅 以便让其他租客了解师傅的工作质量
优先级	低
重要程度	重要
预计工时	2 人时|0.25 人天

(9) 审核租房申请,如表 3-15 所示。

(10) 管理租客,如表 3-16 所示。

表 3-15 审核租房申请

用户故事标题	审核租房申请
描述信息	作为客服 我想要审核所有的租房申请 以便具有复核资格的租客可以成功申请
优先级	高
重要程度	关键
预计工时	4 人时|0.5 人天

表 3-16 管理租客

用户故事标题	管 理 租 客
描述信息	作为客服 我想要查询并修改所有租客信息以便管理租客
优先级	高
重要程度	关键
预计工时	4 人时|0.5 人天

(11) 管理合同,如表 3-17 所示。

(12) 查询可用房源,如表 3-18 所示。

表 3-17 管理合同

用户故事标题	管 理 合 同
描述信息	作为客服 我想要按关键词查询合同 以便获取相关合同的信息
优先级	高
重要程度	关键
预计工时	3 人时|0.38 人天

表 3-18 查询可用房源

用户故事标题	查询可用房源
描述信息	作为客服 我想要查询所有可用房源 以便帮助租客租房
优先级	中
重要程度	关键
预计工时	5 人时|0.63 人天

(13) 修改房源,如表 3-19 所示。

(14) 审核并导入师傅,如表 3-20 所示。

表 3-19 修改房源

用户故事标题	修 改 房 源
描述信息	作为客服 我想要能够增加或暂停房源 以便实时更新房源信息
优先级	中
重要程度	关键
预计工时	2 人时|0.25 人天

表 3-20 审核并导入师傅

用户故事标题	审核并导入师傅
描述信息	作为客服 我想要审核并导入相关维修师傅信息 以便更好地管理相关维修状态,提供更好的维修服务
优先级	低
重要程度	关键
预计工时	4 人时|0.5 人天

（15）回复投诉，如表 3-21 所示。

（16）管理工单，如表 3-22 所示。

表 3-21　回复投诉

用户故事标题	回 复 投 诉
描述信息	作为客服 我想要回复相关投诉 以便解决用户对相关问题的投诉
优先级	低
重要程度	关键
预计工时	4 人时|0.5 人天

表 3-22　管理工单

用户故事标题	管 理 工 单
描述信息	作为客服 我想要根据用户报修信息安排师傅上门服务 以便合理地解决用户问题
优先级	低
重要程度	重要
预计工时	2 人时|0.25 人天

（17）查看工单，如表 3-23 所示。

（18）回复工单，如表 3-24 所示。

表 3-23　查看工单

用户故事标题	查 看 工 单
描述信息	作为师傅 我想要查看自己的所有工单 以便知道哪些用户需要服务
优先级	低
重要程度	重要
预计工时	2 人时|0.25 人天

表 3-24　回复工单

用户故事标题	回 复 工 单
描述信息	作为师傅 我想要回复相关工单 以便让租客了解报修的进行情况
优先级	低
重要程度	一般
预计工时	4 人时|0.5 人天

2）制订迭代计划

在进行迭代前，首先需要制订迭代计划，在规划好用户故事后，需要进一步细化，从而明确所有细节。团队经过讨论分析，计划进行两轮迭代完成项目开发。在第一轮迭代中，团队计划优先实现租房相关需求。首先实现注册、登录、查询房源的基础功能，然后实现租房的整个基本流程，接着实现对合同和租客的管理功能，最后实现合同的审核功能以及订单的查询和管理功能。在第二轮迭代中，团队计划完成报修相关需求，先实现工单的查看和管理功能，然后是对师傅的评价和导入功能，接着是工单和投诉的回复功能，最后是对特色功能 VR 看房等的开发。

习题

1. 选择题

（1）Scrum 有一套独特且固定的管理方式，从（　　）几个维度出发，来保证执行过程更高效。

A. 角色、工件　　B. 角色、不同形式的会议

C. 角色、工件、不同形式的会议　　D. 角色、工件、客户需求

（2）下列选项中不属于看板项目的 5 条核心原则的是（　　）。

A. 可视化工作流程　　B. 限制 WIP

C. 制订明确的执行策略　　D. 额外制订流程应对变化

(3) XP 中,在设计系统的体系架构时,可以采用(　　)促使团队成员共同努力。

A. WIP　　B. CRC 卡　　C. 燃尽图　　D. Scrum 任务板

(4) XP 的 4 个价值观包括(　　)。

A. 交流、简单、反馈、勇气　　B. 沟通、合作、自信、挑战

C. 交流、简单、合作、信任　　D. 沟通、坚持、合作、挑战

(5) 下列选项中不属于 CI 管道的是(　　)。

A. 构建　　B. 模拟　　C. 单元测试　　D. 集成测试

(6) 下列选项中不属于典型的 CI/CD 管道的阶段是(　　)。

A. 提交　　B. 构建　　C. 模拟　　D. 测试和部署

(7) 下列选项中不属于 DevOps 生命周期的是(　　)。

A. 持续反馈　　B. 集成或构建　　C. 持续部署　　D. 调查分析

(8) 敏捷开发和 DevOps 的理念相似,都是为了更好、更快地发布产品,但又不完全相同,而(　　)是实现这两种理念的一种方法。

A. XP　　B. Scrum　　C. CI/CD　　D. 看板

(9) XP 所推崇的规则和实践方法中,所有代码都必须通过(　　)。

A. 单元测试　　B. 集成测试　　C. 功能测试　　D. 验收测试

(10) CI/CD 实践过程中不包括(　　)。

A. 持续集成　　B. 持续改进　　C. 持续交付　　D. 持续测试

2. 判断题

(1) Scrum Master 帮助团队尽其所能地完成工作。例如组织会议,处理遇到的障碍和挑战等,同时 Scrum Master 应该为团队成员指派对应任务,告诉团队成员应该做什么。(　　)

(2) Backlog 的关键要点在于清楚表述列表中每个需求任务能对用户带来的价值,进行动态的需求管理以及迭代的需求分析过程。(　　)

(3) 无论用哪种形式来创建看板图,看板都会有一个原则:提高工作的灵活性、透明度和产出质量。(　　)

(4) 使用 XP 方法时,项目组的所有成员最好在同一个场所工作,以便及时地沟通和解决问题。同时,要求有一个现场用户,由其提出需求并制订需求优先级,编写验收测试用例。(　　)

(5) CI/CD 的目标是帮助开发人员以速度和效率交付软件。团队不断将代码交付到生产中,运行新功能和错误修复的持续流程。(　　)

(6) 所有进入 CD 过程的代码必须首先通过 CI。(　　)

(7) XP 中通过结对编程,程序员通常可以更快地解决问题,出现更少的错误,缩短测试的时间和降低测试的成本。但相应地,个别的人员流动对项目进展造成的影响就会很大。(　　)

(8) 每个 CI/CD 过程都需要有如单元测试、集成测试等所有测试,且持续测试的目标始终相同。(　　)

(9) CI/CD 管道还可以降低编写代码时的风险,使开发人员能够专注于修复代码错误。(　　)

(10) 在技术层面上,DevOps 要求致力于自动化,使项目在工作流程内和工作流程之间不断前进,并致力于反馈和测量,使团队能够不断加快周期,提高软件质量和性能。(　　)

3. 简答题

(1) 什么是敏捷软件开发?

(2) 请简述 Scrum 会议的步骤并加以说明。

(3) 请简述用户故事中的 INVEST 原则。

(4) 请简述 Scrum 与看板的区别。

(5) 请简述 XP 的定义和特点。

(6) 请简述 CI/CD 及其用途。

(7) 请简述 CI/CD 所具有的优势。

(8) 请简述 DevOps 工具的类别。

(9) 请简述 DevOps 流行起来的因素。

(10) 请简述 DevOps 带来的变革。