软件工程

Software Engineering

张宏国

哈尔滨理工大学 计算机科学与技术学院

2024年8月

3.1 敏捷软件开发方法

虽然RUP吸取了软件工程最佳实践，但过程过于庞大，属于“重量级过程”，需通过撰写大量文档来尽量避免交流的歧义性，在支持过程和辅助过程工作上耗费了大量资源，想要快速开发软件几乎是不可能的，如何高效地开发软件快速地响应市场需求？

分析

* 前面讲解的过程都是要明确预期目标和需求，并尽量保持不变。
* 通过周密的计划，在阶段、活动和任务的关键点加强评审和辅助性管理，来提高目标系统的可预见性；
* 需通过撰写大量文档来尽量避免交流的歧义性。
* 通过配置管理等辅助过程和支持过程来控制变更。

以上是造成软件不能快速开发的原因。

3.1 敏捷软件开发方法

在20世纪90年代后期，许多软件专家开始提出更强调短期交付，客户/业务专家的紧密参与，更强调对变化的适应性而不是可预见性，更强调为当前的需要而不考虑将来的简化设计，只将最必要的内容文档化，从而使过程“轻量化”，从而提高软件开发的敏捷性。

3.1 敏捷软件开发方法

2001年初许多业界专家聚集一起，共同探讨如何使软件开发团队具有快速工作、适应变化能力，成立敏捷联盟(Agile Alliance)。

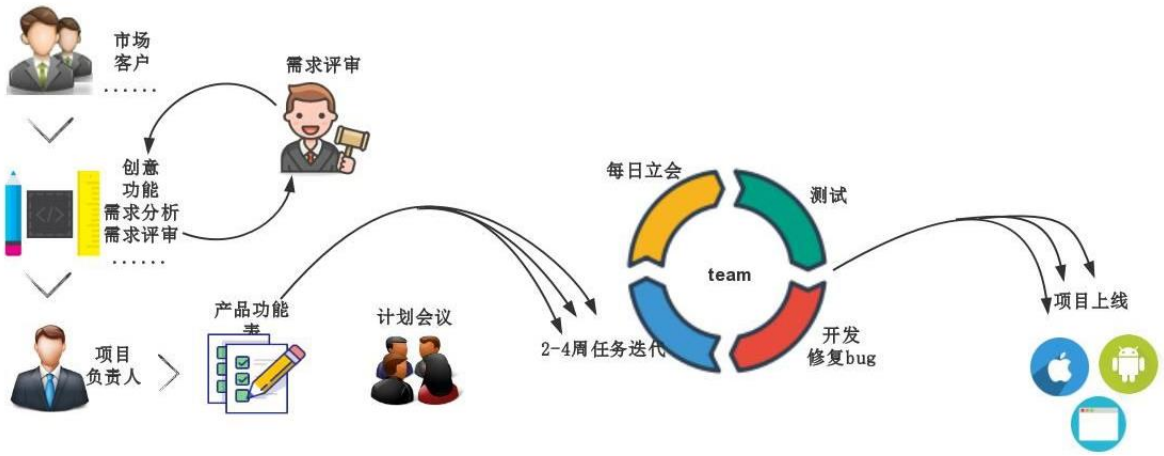
2001年2月敏捷联盟17位著名专家联合起草了敏捷软件开发宣言。宣言包括4个价值观和12个原则。

Agile Alliance®

3.1 敏捷软件开发方法

4个价值观

1. “个体和交互”胜过“过程和工具”
2. “可以使用的软件”胜过“面面俱到的文档”
3. “客户合作”胜过“合同谈判”
4. “响应变化”胜过“遵循计划”



3.1 敏捷软件开发方法

12个原则

1. 最重要的是通过尽早和持续地交付有价值的软件以满足客户需求。
2. 即使在开发后期也欢迎需求的变化。敏捷过程通过驾驭变化带给客户竞争优势。
3. 经常交付可使用的软件，间隔可以从几周到几个月，时间尺度越短越好。
4. 业务人员和开发人员应该在整个项目过程中每天都在一起工作。
5. 使用积极的开发人员构建项目，提供所需环境和支持，并信任他们能够完成。
6. 在开发小组中最有效率和效果的信息传达方式是面对面的交谈。
7. 可以使用的软件是度量进度的主要标准。
8. 敏捷过程提倡持续开发过程。投资人、开发人员和用户应该维持一个稳定步调。
9. 持续地追求卓越的技术与良好的设计会增加敏捷能力。
10. 简单是最重要的,尽量使工作简单化。
11. 最好的架构、需求和设计源于自组织的团队。
12. 团队要定期总结如何提高效率，然后相应地调整自己的行为。

3.1 敏捷软件开发方法

最广泛使用的敏捷方法包括：

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

3.2 Scrum

3.2.1 Scrum概述

Jeff Sutherland 和Ken Schwarber 提出了Scrum开发方法。Scrum有一套其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。

3.2 Scrum

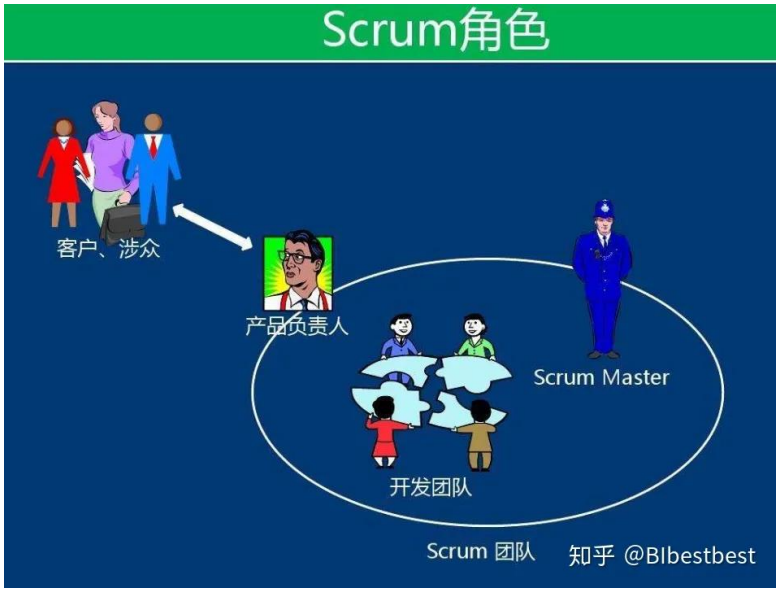
3.2.1 Scrum概述

1. Scrum中的三个角色

（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。

（2）**敏捷主管（Scrum Master）**：帮助团队尽其所能地完成工作，对团队成员在做的事情没有指挥权，而对Scrum这一过程拥有指挥权。

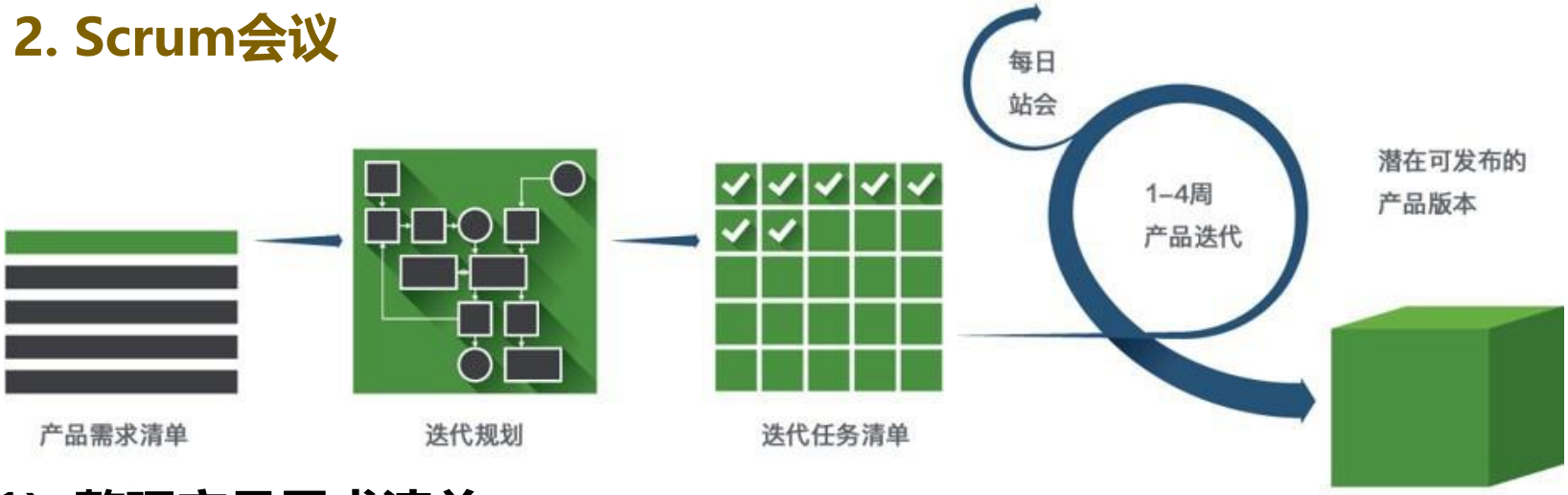
（3）**Scrum团队**：Scrum 团队由5~7名成员组成。与传统开发团队不同，成员们没有固定角色。团队成员间相互帮助、共享成果，旨在完成全部工作。Scrum团队需要做好整体规划，并为每次迭代划分合适的工作量。



3.2 Scrum

3.2.1 Scrum概述

2. Scrum会议



(1) 整理产品需求清单

(2) 确定迭代规划

(3) 梳理迭代任务清单

(4) 每日站会

(5) 迭代演示

(6) 迭代回顾

3.2 Scrum

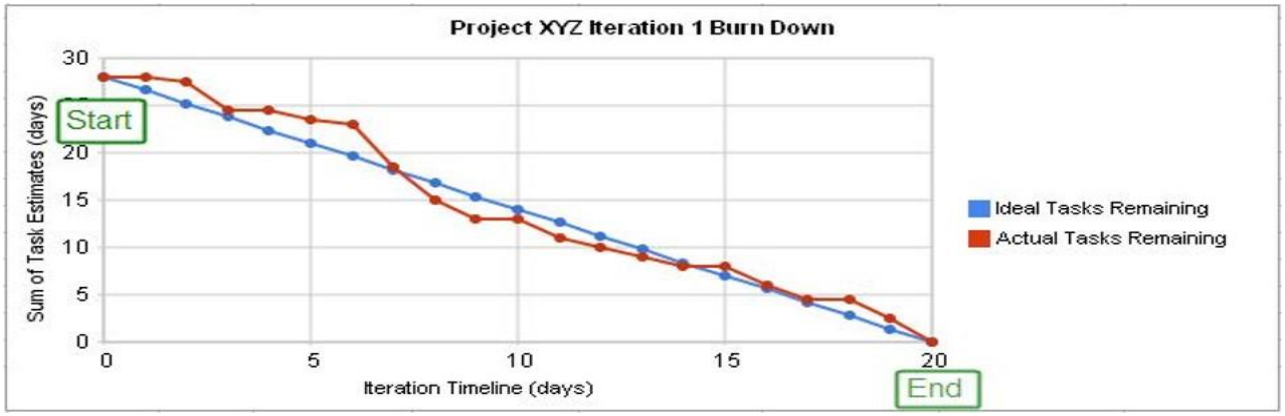
3.2.1 Scrum概述

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，正在进行中和已完成。团队需要在整个Sprint过程中不断更新。

（2）用户故事：用户故事是从客户角度对软件提出功能的描述。它包括用户类型细分，他们想要什么以及他们为什么需要它。

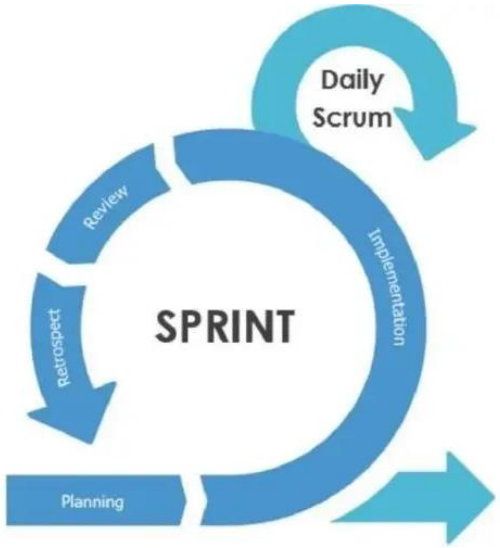
（3）燃尽图：纵轴表示任务总量估计，横轴表示迭代时间。剩下工作可以通过不同的点位或者其他的指标来表示。



3.2 Scrum

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。敏捷专家 Dave West建议，工作越复杂，未知数越多，冲刺应该越短。但这实际上取决于具体的团队情况。



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。

3.2 Scrum

3.2.3 每日站会

（1）这是一个每天在同一时间（通常是上午）和地点举行的超短会议，以保持会议的简单性。每日Scrum的目标是让团队中的每个人都保持同步，与Sprint目标保持一致，并为接下来的24小时制定计划。

（2）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?

3.2 Scrum

3.2.4 用户故事

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。

1. 用户的三要素

* 角色：谁要使用这个功能
* 活动：需要完成什么样的功能
* 商业价值：为什么需要这个功能，这个功能带来什么样的价值

2. 3C 原则

* 卡片（Card）：用户故事一般写在小的记事卡片上。卡片上可能会写上故事的简短描述，工作量估算等。
* 交谈（Conversation）：用户故事背后的细节来源于和客户或者产品负责人的交流沟通。
* 确认（Confirmation）：通过验收测试确认用户故事被正确完成。

3.2 Scrum

3.2.4 用户故事

3. INVEST原则

* 独立性（Independent)
* 可协商性（Negotiable)
* 有价值（Valuable)
* 可以估算性（Estimable)
* 短小（Small)
* 可测试性（Testable)

3.2 Scrum

3.2.4 用户故事

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

举例：图书影视交流平台

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 ● ……	中
用户	● 创建小组 ● ……	中
	● ……	中
	● ……	低
管理员	● 管理用户 ● 办理申请 ● ……	高

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，以便小组之间进行影视交流。
优先级	中
重要程度	一般
预计工时	10人时 | 1.25天

3.2 Scrum

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，用来制定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:

- **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**
- **动态的需求管理而非“冻结”方式**
- **需求分析的过程是可迭代的**

3.2 Scrum

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。

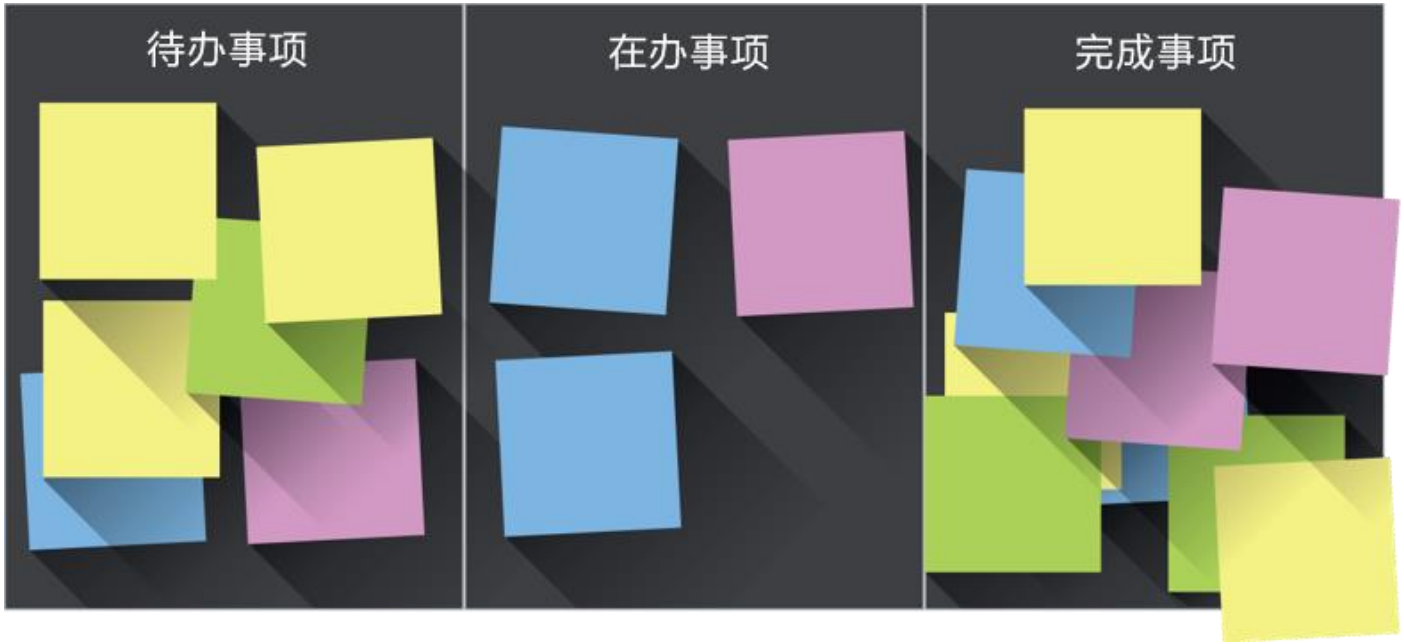
结对编程的优点:

* 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
* 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 看板概述

看板（Kanban）作为可视化框架可以用于敏捷方法，能够清晰地向项目成员展示整个项目进度。当需要对系统进行小幅度改动的时候，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定流程。看板图是项目中实施看板的常见工具。



3.3 看板

3.3.1 看板概述

无论用哪种形式来创建看板图，看板都会有一个原则：划分为不同列来代表其工作状态。

看板项目包括如下5条核心原则：

(1) 可视化工作流程

(2) 限制WIP

(3) 管理和改进流程

(4) 制定明确的执行策略

(5) 持续改进

3.3 看板

3.3.2 看板与Scrum的区别

（1）Scrum与看板有所不同，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适合成熟一点的产品和团队。

（2）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。

（3）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。

3.3 看板

3.3.2 看板与Scrum的区别

（4）在 Scrum 中每个 Sprint 的时间长度是固定的（1~4 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（1~4 周），那么 Scrum 是比较好的选择。

（5）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。

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

3.3 看板

3.3.2 看板与Scrum的区别

Scrum	看板开发方式
要求定时迭代	没指定定时限迭代，可以分开计划、发布、过程改进，可以事件驱动而不是限定时限
团队在每个迭代承诺一定数目的工作	承诺不是必须的
以速度（Velocity）作为计划和过程改进的度量数据	使用开发周期作为计划和过程改进的度量数据
指定跨功能团队	没有指定跨功能团队，也容许专门团队
工作任务细分，可于一个迭代中完成	没有指定工作任务大小
指定使用燃烧图	没有指定任何图表

3.3 看板

3.3.2 看板与Scrum的区别

Scrum	看板开发方式
间接限制开发中工作（每个迭代）	设定开发中工作的限制（每个工作流程状态）
规定估算过程	没有指定任何估算方式
在迭代中不能加入新工作任务	只要生产力容许，可以随时加工作任务
由单一团队负责 Sprint Backlog	多个团队和团员分享看板
指定3个角色（产品经理、Scrum Master、Scrum团队）	没有指定任何团队角色
Scrum board 在每个迭代后重设	看板反映持久开发情况
规定优先化的 product backlog	优先级是非必须的

3.4 极限编程

3.4.1 概述

极限编程（eXtreme Programming，简称XP）是由KentBeck在1996年提出的，是一种软件工程方法学，是敏捷过程中最负盛名的一个。

其名称中“极限”的含义是把好的开发实践运用到极致。

极限编程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。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（比如一个办公室）的场合。

极限编程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。

代码的质量在极限编程项目中非常重要。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。合理的测试用例及较高的测试覆盖率是极限编程项目测试所追求的目标。

3.4 极限编程

3.4.1 概述

XP的常用术语

* **用户故事**：开发人员要求客户把所有的需求写成一个个独立的小故事，每个只需要几天时间就可以完成。开发过程中，客户可以随时提出新故事，或者更改以前的用户故事。

* **发布计划**：为每个系统版本实现哪些用户故事以及这些版本的日期。

* **迭代**：整个开发过程中，开发人员将不断地发布新版本。开发人员和客户一起确定每个发布所包含的用户故事。

* **种子**：一次迭代完成后的工作产物是下一次迭代的种子。

* **项目速度**：小组完成工作的快慢程度。它是相对的。测量项目速度很简单，以一次迭代完成的用户故事的数目或编程任务的数目度量。

3.4 极限编程

3.4.1 概述

XP的常用术语

* **连续整合**：把开发完的用户故事的模块一个个拼装起来，一步步接近乃至最后完成最终的产品。
* **验收测试**：对于每个用户故事，将定义一些测试案例，开发人员将使运行这些测试案例的过程自动化。
* **单元测试**：在开始编写程序前，程序员针对大部分类的方法，先编写出相应的测试程序。
* **重整**：去掉代码的冗余部分，增加代码的可重用性和伸缩性。
* **双人（结对）编程**：两个程序员同时在同一台计算机上共同编写解决同一个问题的程序代码，通常一个人编码，另一个人进行代码审查与测试。

3.4 极限编程

3.4.1 概述

XP的常用术语

* **小（版本）发布：**XP要求完成任何一个改动、整合或者新需求后，就应该立即发布一个新版本。
* **集体拥有代码：**每个人都有权力和义务阅读其他代码，发现和纠正错误，重整代码。
* **隐喻：**为了有助于一致、清楚地理解要完成的客户需求和要开发的系统功能，XP开发小组用想象的比喻来描述系统功能模块是怎样工作的。

XP所推崇的规则和实践方法

计划

- 编写用户故事
- 根据版本发布计划，创建日程安排
- 可以增加发布小版本的频率
- 对项目的进度速度进行度量
- 项目可以被划分为若干次迭代周期
- 每次迭代周期都从相应的迭代计划开始
- 使开发人员尽量熟悉项目的所有方面
- 每天都要召开站会
- 当遇到问题时要及时对过程进行修复

设计

- 力求简洁
- 选择合适的系统命名方式
- 使用CRC卡
- 当遇到棘手的技术问题或设计问题时，可以专门针对此问题提出解决方案，以此来降低软件项目的风险
- 关注目前所需要的功能，忽视将来可能需要的功能
- 及时去除冗余带代码、不使用的功能以及过时的设计。

编码

- 用户参与整个编码过程
- 应该根据共同协商好的标准和协议进行编码
- 在正式编程之前，先构建单元测试方案
- 采用结对编程的方式
- 采用顺序方式而非并行方式进行代码集成
- 经常性地集成代码
- 代码的所有权归集体
- 进行代码优化
- 不安排加班

测试

- 所有代码都必须通过单元测试
- 在新版本发布之前，单元测试必须已完全测试成功
- 发现软件缺陷后，及时创建相应的测试
- 经常性地进行验收测试并公布测试结果

3.4 极限编程

3.4.2 XP的4个价值观

1. 交流

交流不仅能使相关人员更为精确的理解需求，而是能够尽可能避免因为需求变更导致的不一致。

2. 简单

简单是XP推崇的理念，一切都使用最简单的、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。

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。

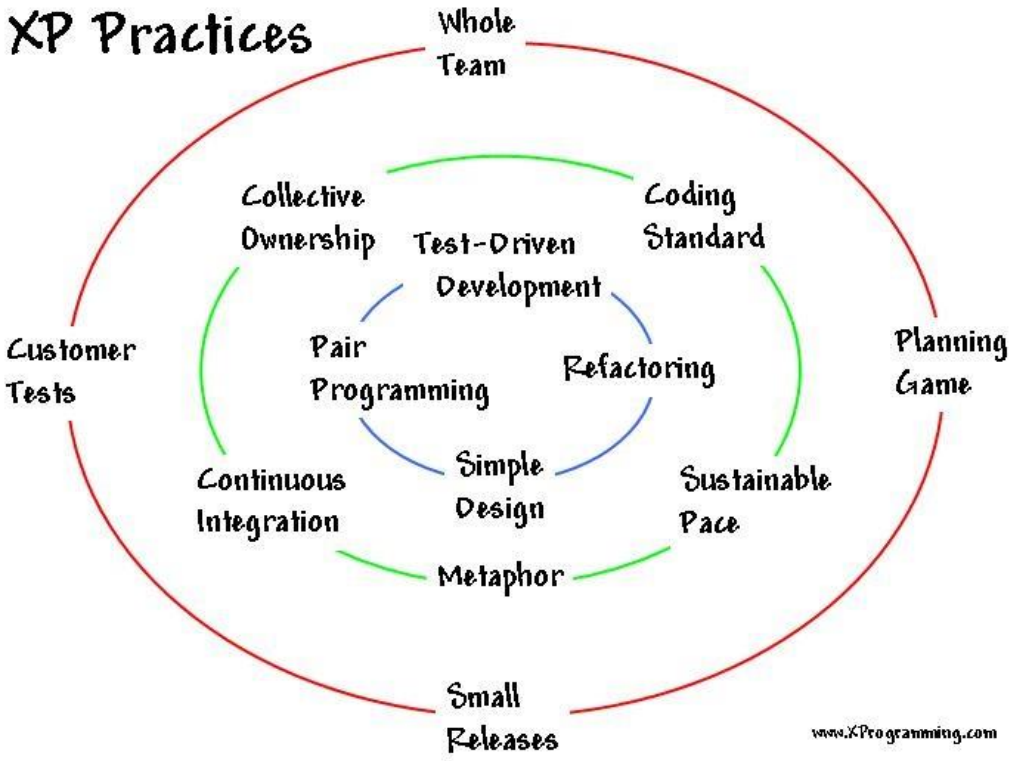
4. 勇气

敏捷方法要求与其他人密切的合作，充分信任他人，也信任自己，这需要勇气。

3.4 极限编程

3.4.3 XP的12个核心实践

1. 完整的团队
2. 计划策略
3. 系统隐喻
4. 小型发布
5. 测试驱动
6. 简单设计
7. 结对编程
8. 设计改进
9. 持续集成
10. 集体所有权
11. 编码标准
12. 工作安排



3.5 CI/CD

3.5.1 CI/CD概述

CI/CD （Continuous Integration / Continuous Delivery）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。

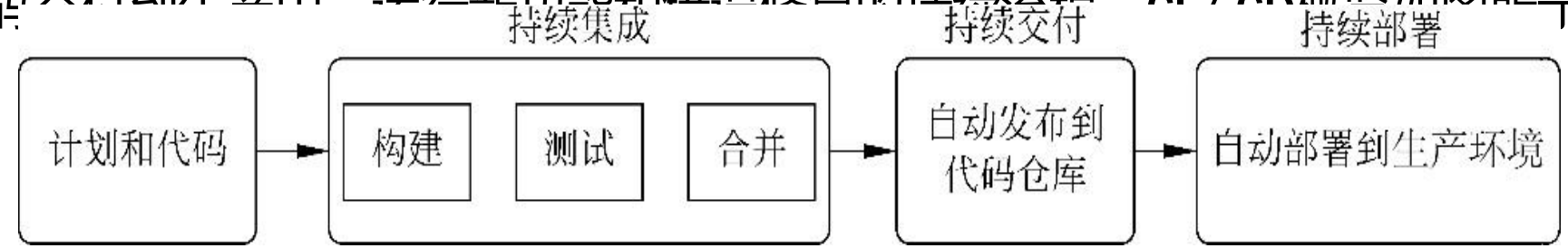
CI/CD 中的 CI 代表持续集成。CD 是指连续交付或连续部署，具体取决于团队选择如何推动代码更改以进行生产。

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合理地合并到新软件中。

（2）CD提供了向最终用户交付代码地快速无缝方法。

CI / CD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概览如图所示。



3.5 CI/CD

3.5.1 CI/CD概述

1. 持续集成 （CI）

持续集成是不断将更新集成到代码库的方法。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。

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更改。

所有进入 CD 过程的代码必须首先通过 CI。

3.5 CI/CD

3.5.1 CI/CD概述

3. 持续测试

持续测试是运行自动测试的方法，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都协同工作）
- 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。

3.5 CI/CD

3.5.2 CI/CD

CI/CD 管道是所有软件在其开发生命周期中遵循的可运行、分步的路径。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。

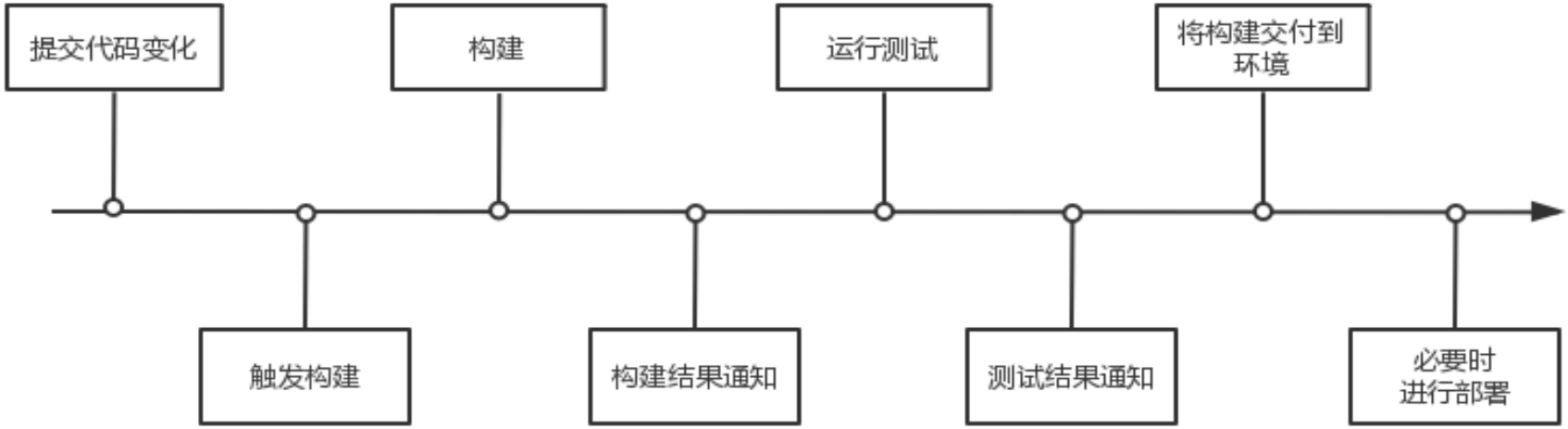
自动化管道为团队提供了多项优势

- 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
- 所有代码更改、测试和部署完全透明

3.5 CI/CD

3.5.2 CI/CD

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/DI管道的 一个例子

3.5 CI/CD

3.5.2 CI/CD

典型的 CI/CD 管道分为四个主要阶段：提交、构建、测试和部署，如下。表所示较复杂的管道除了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

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

3.5 CI/CD

3.5.2 CI/CD的优势

（1）更快、更可靠的版本发布

（2）更高的可见性

（3）早期错误检测快速反馈循环

（4）更快乐的开发和运维团队

3.6 DevOps

DevOps是从敏捷发展起来的。它添加了新的过程和工具，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 DevOps

3.6.1 DevOps生命周期

DevOps生命周期（当以线性方式描述时，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，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。其生命周期如下图所示。

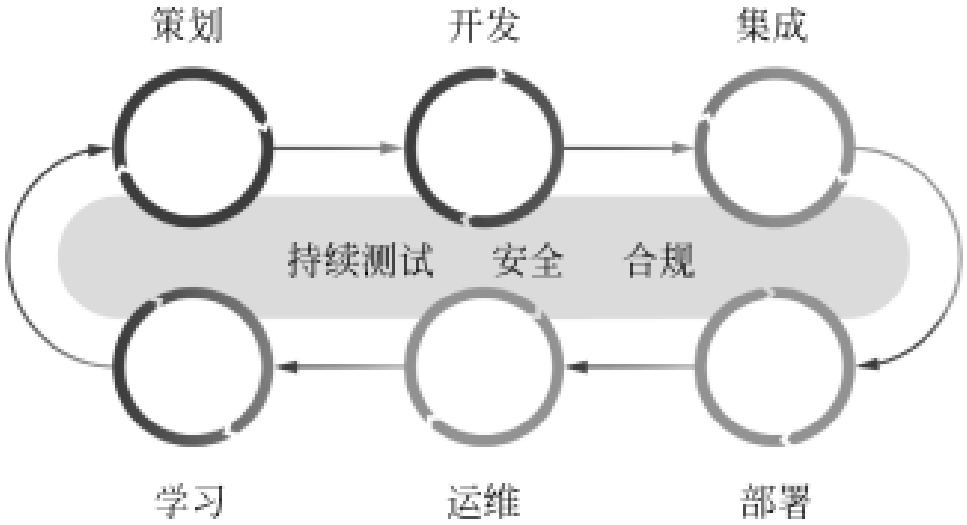


图 3-8　DevOps 生命周期

3.6 DevOps

3.6.1 DevOps生命周期

工作流的名称和数量可以根据所询问的对象而有所不同，但它们通常可以归结为以下六种。

- 策划
- 开发
- 集成或构建
- 部署
- 运维
- 学习

另外三个重要的连续工作流发生在这些工作流之间

- 持续测试
- 安全
- 合规

3.6 DevOps

3.6.2 DevOps文化

DevOps文化可以概括为软件开发的一种不同的组织和技术方法。

- 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。

3.6 DevOps

3.6.3 DevOps工具

- 项目管理工具
- 协作式源代码库
- CI/CD管道
- 测试自动化框架
- 配置管理
- 监控工具
- 持续反馈工具

3.7 敏捷软件开发、CI/CD和DevOps

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CI/CD是实现这两者理念的一种方法。

敏捷开发的核心是，拥抱变化与快速迭代。

持续集成（CI）、持续交付和持续部署（CD）提供了一个优秀的DevOps 环境，对于整个团队来说，好处与挑战并行。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，是因为一下几个因素：

- 容器技术开始成熟，特别是docker等技术的大行其道
- 微服务架构技术的广泛使用
- 敏捷开发流程的深入人心
- DevOps带来的变革

3.7 敏捷软件开发、CI/CD和DevOps

DevOps带来的变革具体为:

- **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**
- **研发：专注研发、高度敏捷、持续集成**
- **产品交付：高质量、快速、频繁、自动化、持续交付**

小结

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；具体讲述流行的敏捷软件开发方法，如Scrum、看板（Kanban）和XP；然后讲述CI/CD以及DevOps等相关方面的内容。