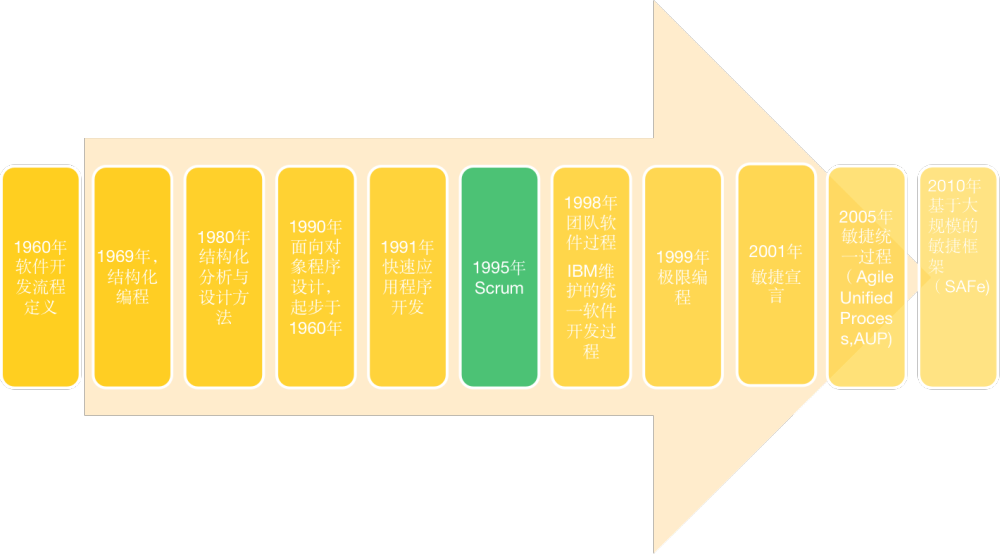
# 第九章 DevOps 开源实践

# 9.1 从敏捷到DevOps

## 9.1.1 软件开发历史回顾



敏捷和DevOps无疑是近年来软件开发领域的重要概念。然而，为了更好地理解它们，我们需要回溯一下软件开发的历史。在这一过程中，我们不仅能够看到开发流程的演变，也能够理解敏捷和DevOps的价值所在。

谈到软件开发流程，早在1960年代，人们就开始定义软件开发流程。然后，在1969年，结构化编程的概念被提出。1980年代，结构化设计方法进一步发展，到了1990年代，面向对象设计开始广泛流行，而它的起源实际上可以追溯到1960年代。到了1991年，快速应用开发（RAD）的概念被提出。然后在1995年，ACM的计算机大会上首次在论文中提出了Scrum的概念。这一概念随后与工业界和学术界的合作，逐渐发展成为现代Scrum的概念，因此我们将这个事件视为Scrum的起源。

在1998年，IBM推出了统一的软件开发过程（Unified Software Development Process）。这一过程最初由Rational公司维护，后来Rational公司被IBM收购。1999年，极限编程（Extreme Programming，XP）开始流行。然而，真正让敏捷开发理念引起人们广泛关注的是2001年发生的一次重大事件。在美国犹他州的雪鸟滑雪圣地，一些敏捷爱好者和软件开发工程师举行了一次会议，并发布了《敏捷宣言》。这份宣言将敏捷开发的各种理念统一起来，为敏捷提供了一套更加统一和全面的实践指南。

敏捷开发的兴起，一方面是因为开发过程中快速应用开发的需求，一方面也是由于软件开发流程的复杂度和需求的不断变化。敏捷开发重视快速响应变化，优化开发流程，提高开发效率，因此在软件开发领域迅速流行起来。

敏捷宣言强调四个主要的价值观：个体和互动优于流程和工具；可工作的软件优于详尽的文档；客户合作优于合同谈判；以及响应变化优于遵循计划。这四个价值观反映了敏捷开发的核心理念：快速响应变化，注重实际效果，以及优化开发流程。

Scrum和极限编程（XP）是敏捷开发中的两个重要框架。Scrum是一种强调团队合作的迭代和增量的软件开发框架。而极限编程则是一种重视代码质量和团队合作的开发方式，它强调通过频繁的发布和测试来快速响应需求变化。

然而，尽管敏捷开发在提高开发效率和响应变化方面取得了成功，但它并不能解决所有的问题。这就引出了DevOps的概念。DevOps是一种强调开发（Development）和运维（Operations）团队紧密合作的文化、运动或实践，以实现快速、高效的软件开发和交付。

通过以上的讨论，我们可以看到，从1960年代的软件开发流程的定义，到结构化编程，再到面向对象设计，快速应用开发，Scrum，以及最后的敏捷开发，整个软件开发领域的历史就是一场不断追求效率，快速响应变化，优化开发流程的历程。在这个历程中，我们不断学习、改进，最终形成了今天我们所熟知的敏捷和DevOps。

## 9.1.2 敏捷开发的起源和价值观

在上世纪90年代末，一系列的新的软件开发方法开始出现，这些方法尝试通过更高效、更轻量级的方式来开发软件。这些方法被总称为"轻量级方法"，包括极限编程（XP）、Scrum、自适应软件开发（Adaptive Software Development, ASD）、Crystal等。这些方法各自独立发展，但都强调面对面的沟通、持续交付的软件、团队协作和响应变化。

2001年，17位重要的轻量级方法的创始人和实践者聚集在美国犹他州的雪鸟滑雪圣地，讨论了如何改进软件开发的实践。这次会议的结果是《敏捷宣言》的诞生，该宣言明确提出了敏捷开发的四个核心价值观，及其背后的十二条原则。

这四个核心价值观是：

1. 个体和互动优于流程和工具：虽然流程和工具是必要的，但是敏捷开发认为，优秀的个体和良好的团队互动才是成功的关键。这是因为在开发过程中，需要人们来解决问题、创新、适应变化。

1. 工作的软件优于详尽的文档：虽然文档是重要的，但敏捷开发强调最终的目标是交付能够工作的软件。详细的文档不能替代实际的软件，而且过度的文档可能会成为开发过程中的负担。

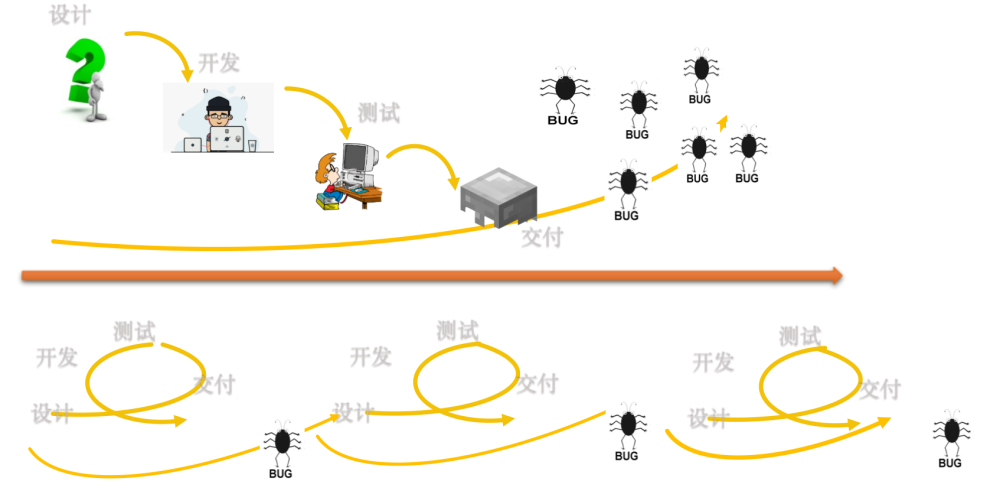
1. 客户合作优于合同谈判：虽然合同是必要的，但敏捷开发更注重与客户的持续合作。敏捷开发鼓励开发团队与客户保持紧密的沟通，以便更好地理解他们的需求，更快地做出调整。

1. 响应变化优于遵循计划：虽然计划是重要的，但敏捷开发认为，在面对变化时，应该能够灵活地调整计划。因为在项目开发过程中，需求总是在不断变化的，敏捷开发就是为了更好地应对这种变化。

这四个价值观及其背后的十二条原则，为敏捷开发提供了明确的指导。它们促使开发团队去优化自己的开发实践，以提高软件开发的效率和质量。从此，敏捷开发开始在全球范围内得到广泛的应用，并且已经成为了现代软件开发的重要理念和方法。

## 9.1.3 敏捷开发和瀑布模型的比较

瀑布模型和敏捷模型是软件开发过程中最常用的两种模型。它们之间存在着明显的区别，这些区别主要源于对软件开发过程的不同理解和对客户需求的不同应对方式。



瀑布模型：瀑布模型是一种传统的软件开发模型，它将开发过程分为一系列的阶段，每个阶段都依赖于前一个阶段的完成。这些阶段包括需求分析、设计、编码、集成、测试、部署和维护。在瀑布模型中，每个阶段都需要进行详尽的文档化，并且一旦一个阶段完成，就不能再返回到前一个阶段进行修改。因此，瀑布模型更适用于需求明确、变化较少的项目。

敏捷模型：与瀑布模型不同，敏捷模型更加强调团队的协作、客户的参与和对变化的响应。在敏捷模型中，软件开发被看作是一个迭代和增量的过程，而不是一次性完成所有阶段的线性过程。敏捷模型鼓励频繁和早期的软件交付，以便尽早获得客户的反馈，并对需求进行调整。敏捷模型更适用于需求复杂、变化较多的项目。

当我们将瀑布模型与敏捷开发模型进行比较时，可以从多个角度看出两者之间的区别：

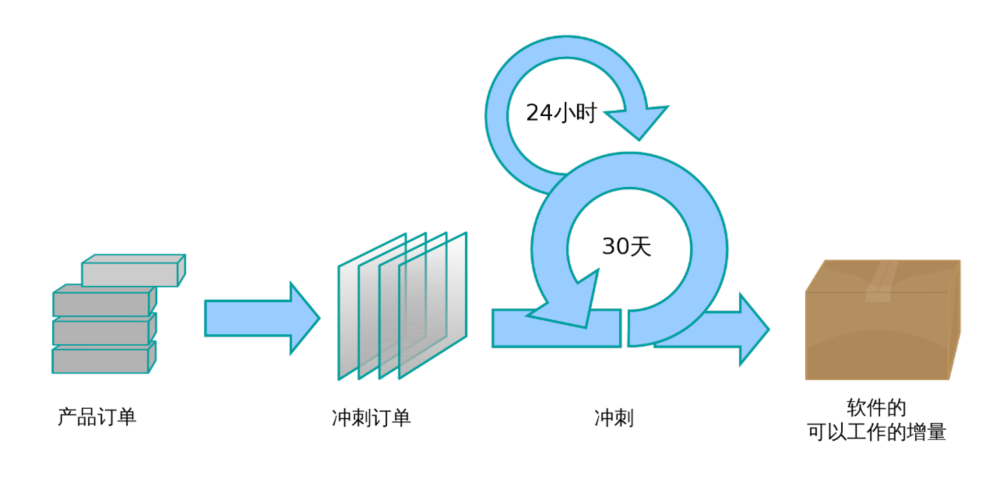
1. 项目流程与阶段性：瀑布模型的开发流程就如同瀑布般连续流动，但每个阶段都是分隔开来的，项目从需求收集阶段开始，然后经过设计、编码、测试，最后到达部署和维护阶段，每个阶段都有明确的起止，而且按照特定的顺序执行。这种线性流程虽然在项目管理上具有明确性和易于理解的优点，但也导致了一旦需求变动，就需要重新返回到流程的起点。这种模型对于需求稳定，项目规模较大，时间预算充足的项目较为适用。

相对地，敏捷开发模型强调的是快速迭代与频繁交付。在敏捷开发中，复杂的系统被划分为多个可管理的小部分或迭代周期，每个迭代周期都包含了需求分析、设计、编码、测试和部署等环节。也就是说，每个小版本都可以视为一个完整的软件生命周期。这样做的好处在于，开发团队可以在较短的周期内完成软件的开发并交付给客户，然后根据客户的反馈迅速调整，不断优化和完善产品。

1. 风险管理与问题解决：由于瀑布模型的线性特性，问题通常在开发阶段后期被发现，此时要修改就需要消耗大量的时间和资源，风险较大。而敏捷开发模型则通过频繁的迭代，早期暴露问题，使得问题可以在早期阶段得到解决，大大降低了风险。

1. 对变化的响应：瀑布模型一般在项目开始时就定义好了所有需求，对于需求变更的处理效率相对较低。而敏捷模型由于其迭代的特性，对于需求的变化可以快速响应，对新需求进行快速的迭代并投入生产。

综上，瀑布模型和敏捷开发模型各有其优势和适用场景，关键在于如何根据实际项目需求和环境选择最合适的开发模型。



我们进一步了解一下Scrum，这是一个敏捷开发中最广泛使用的一种实践模型，它提出了一种具有高度适应性和灵活性的项目管理框架，旨在支持团队协作，并针对快速改变的需求提供解决方案。我们这里不仅仅讨论敏捷的具体实现，也不讨论敏捷中人员的分工，我们只看敏捷开发模型的整个流程。在Scrum中，首先将产品订单转化为冲刺订单，然后对这些冲刺订单进行迭代周期的规划，迭代周期可以是长达30天或短至24小时的时间段。然后不断进行迭代，并快速交付给用户。用户面对的是不断增量变化的系统产品，与我们之前所说的敏捷开发非常一致。

Scrum的基本框架相对简单。Scrum中的核心是一个称为“冲刺”的时间盒，这是一个固定长度（通常为1-4周）的开发周期，每个冲刺的目标是生成一段可交付的、有价值的软件。在开始每个冲刺前，团队会进行一个冲刺计划会议，确定这个冲刺期间需要完成的工作。

这些工作项来源于产品待办事项，这是一个由产品负责人维护的、按照优先级排序的需求和功能列表。在冲刺计划会议上，团队会选择产品待办事项中的一部分，转化为冲刺待办事项，成为这个冲刺需要完成的工作。

在冲刺期间，每天都会进行一个短小的团队会议，这被称为每日站会。在每日站会上，每个团队成员会分享他们前一天完成了什么，今天计划做什么，以及是否遇到任何阻碍。这有助于同步团队的工作进度，发现并解决问题。

在冲刺结束时，团队会进行一个冲刺回顾会议，团队会检查在冲刺中完成的工作，以及未完成的工作。这是一个回顾和反思的机会，团队会讨论什么工作得到了好的完成，什么地方需要改进。另外，团队也会进行冲刺回顾会议，这是一个检查并改进过程本身的机会。

在Scrum中，不断迭代、及时反馈和持续改进的理念贯穿始终。通过这种方式，Scrum能够帮助团队在复杂、不确定的环境中，有效地进行项目管理和产品开发。

## 9.1.4 DevOps的起源和理念



DevOps的起源可以追溯到2009年比利时的第一届DevOps Day。这个概念最初是由一个独立技术咨询顾问提出来的，他在参与一个大型政府项目的过程中发现，开发团队和运维团队之间存在冲突和痛苦的角色转变。他提出，为了达到每天进行10次以上的部署，开发团队和运维团队需要更好的合作。这个想法引起了社区的共鸣，最终在Twitter上发起了DevOps Day的活动，因为Twitter的字数限制，所以最终将活动名称简化为DevOps。

DevOps的核心理念是实现开发（Development）和运维（Operations）之间的沟通、文化和协作，以此来消除开发和运维之间的矛盾，从而提高软件的交付效率和质量。其重要性在于，即使我们采用了敏捷开发，但如果开发团队和运维团队之间的合作不畅，那么最终的产品交付仍可能受到影响。

DevOps的实现方法则强调通过自动化的软件交付和架构变更的流程，以实现更快速、可靠的软件构建和发布。在DevOps的实践中，持续集成（Continuous Integration）和持续交付（Continuous Delivery）是两个关键概念。持续集成是指开发人员将代码合并到主干的频繁动作，以便更早发现问题并解决。持续交付则是通过自动化的构建、测试和部署流程，确保软件在任何时候都能在一个稳定的状态下进行发布。

DevOps不仅仅是一种工作方式或是工具的使用，更是一种文化。它鼓励开发团队和运维团队之间的紧密协作，消除二者之间的隔阂，提升软件开发的效率和质量。就如同敏捷开发，DevOps也是对快速变化的需求的一种应对能力。尤其在当前微服务和云原生架构越来越流行的情况下，DevOps已经成为实现高效软件开发和交付的重要方法论。

# 9.2 DevOps和开源的碰撞

## 9.2.1 DevOps中的工具和开源工具的重要性

DevOps是一种软件开发方法，其核心理念是使开发人员(Dev)与运营人员(Ops)更紧密地合作，实现更快速、更可靠的软件构建和部署。在这个过程中，工具的作用是至关重要的。

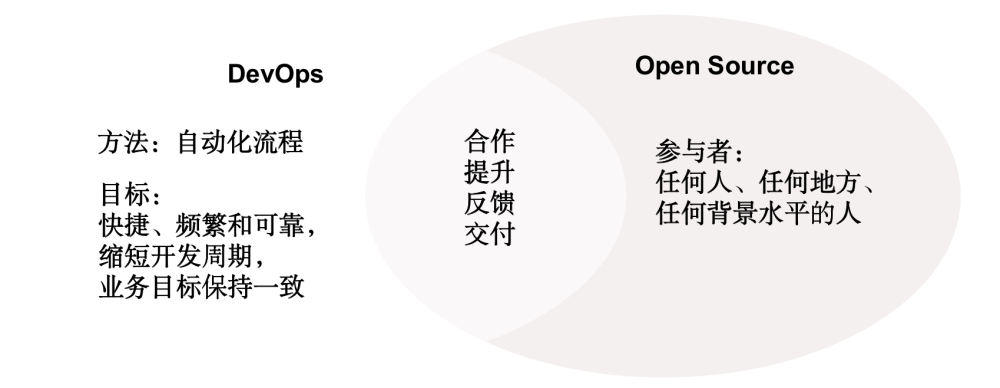


一份由Digital AI公司整理的“DevOps工具周期表”上，列出了120个与DevOps相关的工具，这些工具涵盖了DevOps的整个流程，从需求定义、代码托管、测试、问题追踪，到发布阶段的云服务提供商、数据库管理、发布管理，甚至包括源代码管理和安全相关的测试。这些工具根据DevOps的整个流程进行了分类，不同的颜色代表不同的分类。此外，还对这些工具进行了其他几个方面的分类。第一个是工具是否开源？第二个是工具是否免费，还是需要付费或者来自公司级产品？这些方面的分类使我们可以看到，实际上这些工具中有三分之一是开源的，这三分之一的开源工具涵盖了持续集成和DevOps实践过程中使用的工具链。除了开源工具之外，剩下的三分之二中，大约有50多个工具是公司级产品，例如NPM安装包、云服务平台CloudPlan等。

这表明，开源工具在DevOps中占据了重要的位置。它们在代码管理、自动化构建和部署、问题管理和安全检测等方面提供了强大的支持，从而使得DevOps能够有效地实现其理念。因此，无论是在实施DevOps的公司，还是在开源社区中，我们都可以看到开源工具的广泛应用。

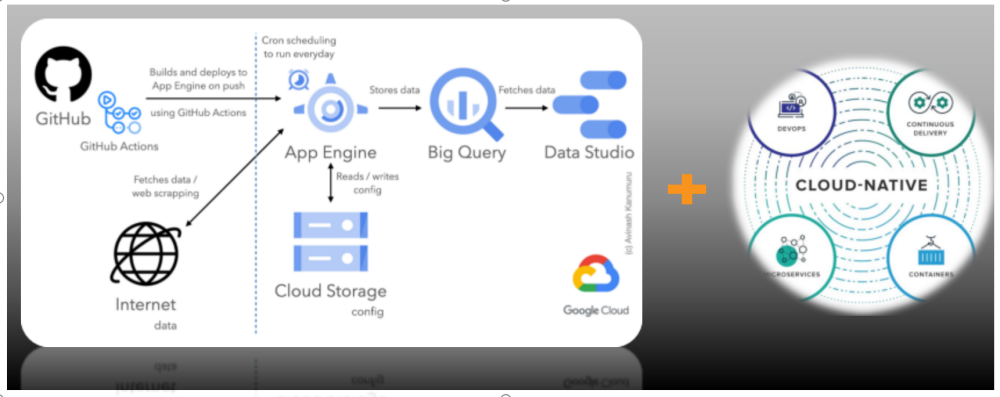
开源和DevOps是相互依赖，互相促进的。DevOps强调方法、文化和工具的使用，追求自动化的流程和缩短开发周期，以确保业务目标的一致性、更快速和可靠的交付。然而，开源项目具有一个特点，就是它的参与者可以来自任何地方，他们的背景和知识水平可能各不相同。开源项目从诞生之初就暴露在众人面前，无论是开发人员还是最终用户，都可以对其进行使用和参与。因此，开源项目需要依赖DevOps的工具链来保证项目的高质量和创新力。

## 9.2.2 开源和DevOps的相互依赖



云原生的概念并不新鲜，但它对于开源与DevOps的影响是深远的。云原生是一种构建和运行应用的方法，使得应用可以充分利用云计算的优势。这意味着开发者可以轻易地将应用部署到云环境中，无论是公有云、私有云还是混合云。特别是对于基于Web服务端的开源产品而言，它们可以实现一键将代码从测试环境部署到真正的云端生产环境。这一点在DevOps中尤其重要，因为DevOps追求的是快速、持续的交付和迭代，而云原生技术则可以大大加快这个过程。

在云原生的背景下，容器化和微服务成为了推动开源与DevOps融合的关键因素。容器技术，如Docker和Kubernetes，使得开发者可以将应用和其运行环境打包在一起，确保应用在不同环境中的一致性。这极大地简化了应用的部署和移植过程，是DevOps中持续交付的基石。同时，微服务架构使得应用可以分解为一组小的、独立的服务，每个服务都可以独立部署和扩展。这使得开发团队可以更快速、更灵活地进行开发和迭代，符合DevOps的敏捷精神。



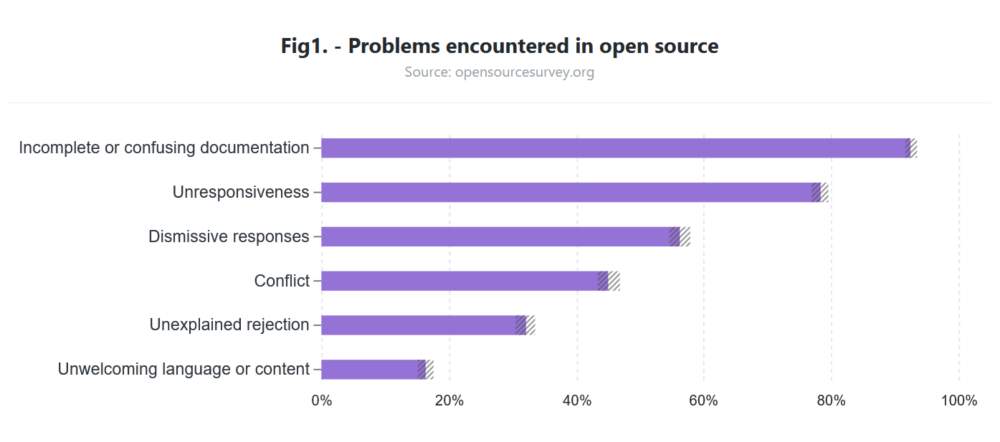
这些因素推动了开源与DevOps的深度融合，形成了我们现在看到的云原生DevOps。在这种新形态中，开发、部署和运维的界限变得更加模糊，开源软件在整个生命周期中都发挥着关键作用。

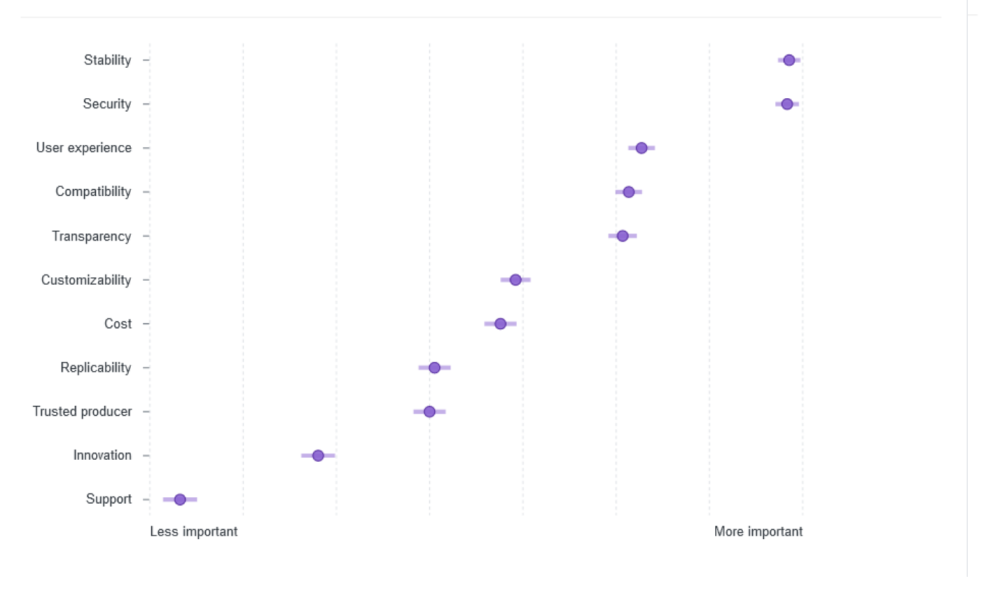
开源不再仅仅是一种软件开发和分发的模式，而是一种促进协作和创新的文化。它推动了DevOps的发展，同时也受益于DevOps持续交付、快速反馈的理念。与此同时，云原生的理念也在推动着DevOps进一步的演化，使得它更加灵活、更具可扩展性。

在这个过程中，我们看到了开源与DevOps的深度融合，也看到了开源如何推动DevOps的发展，以及如何通过与云原生的合作，将DevOps推向了新的境界。这是一次充满变革的历程，我们有幸能够参与其中，见证DevOps和开源碰撞所带来的新形态的诞生。

# 9.3 开源项目面临的质量问题

## 9.3.1 优秀的开源项目的特征和管理要素





在深入探讨“优秀的开源项目”的定义之前，我们有必要先回顾一份来自2017年GitHub的调查报告。该报告综合了全球开发者在使用或参与开源项目时面临的一系列挑战。其中最常见的问题包括文档的不完整或描述的模糊，以及对于问题或需求缺乏及时的反馈。尽管也有一些问题涉及到冲突、不受欢迎的语言或内容，但这些问题相对较少。

这些反馈为我们描绘了优秀开源项目的基本框架。首先，一份详尽且完整的文档，包含使用指南和API文档等，是不可或缺的。这样的文档为用户提供了清晰的指引，使他们能够更轻松地理解和使用项目。其次，项目团队需要有能力及时响应用户的问题和需求，提供必要的帮助。

进一步的，我们需要考虑用户在使用开源软件的过程中，他们最渴望得到的是什么？稳定性和安全性是这两个核心需求。除此之外，用户体验、兼容性、透明度以及个性化选项也是用户关注的重要方面。

将以上因素融入到我们对“优秀开源项目”的定义中，我们可以得出以下结论：一个优秀的开源项目不仅需要有完备的文档和及时的响应机制，而且必须保证软件的稳定性和安全性。稳定性意味着在各种环境和条件下，软件都能可靠地运行；而安全性则是指项目能够保护用户数据，抵御各种可能的安全威胁。为了实现这两个目标，持续的更新和维护，严谨的质量管理体系是必不可少的。

要概括一个优质的开源项目，我们可以根据以下四个核心要素来进行评估和管理：

1. 完善的文档管理：文档是开源项目的重要组成部分，能够帮助用户更好地理解和使用项目。因此，一个优秀的开源项目应该拥有详细、清晰且易于理解的文档。它应该定期更新，确保信息的准确性，与代码的同步，以及符合用户的需求。

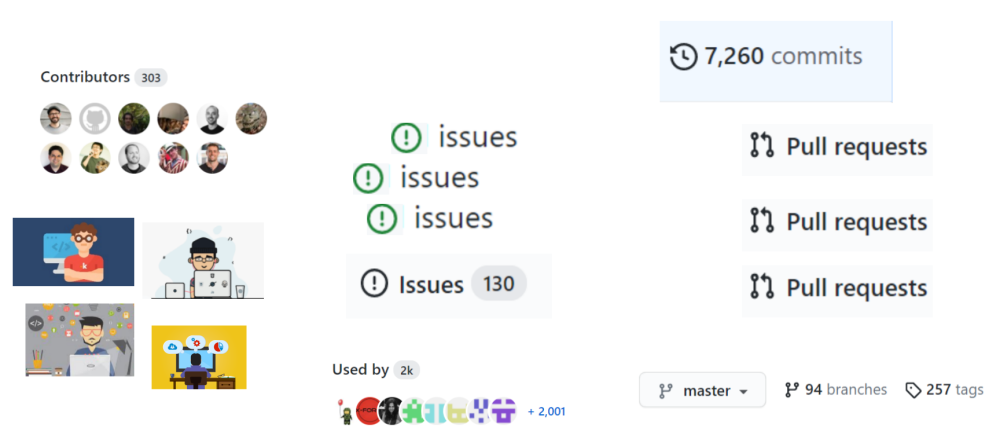
1. 高质量的代码管理：代码管理是确保开源项目稳定性和安全性的重要环节。一个健全的代码管理体系包括严格的版本控制、代码审查、以及全面的测试流程，可以确保代码质量的高标准，并且能够及时发现并修复代码中的问题和漏洞。

1. 积极的社区响应：开源项目离不开社区的参与和贡献。优秀的开源项目需要有一个积极、活跃的社区，能够及时响应用户的问题、需求和反馈，鼓励社区成员的积极参与，促进项目的持续改进和发展。

1. 文化建设和许可证管理：良好的开源文化能够营造一个积极、开放、共享的环境，鼓励社区成员的参与和贡献。此外，选择合适的开源许可证，可以确保代码的合法性和可重用性，保护项目和贡献者的权益。

以上四个要素共同构成了优秀开源项目的质量管理基础。它们有助于保证项目的稳定性、安全性，以及提高用户满意度。在接下来的章节中，我们将深入探讨如何构建一个高质量的代码管理体系，以及如何进行有效的代码迭代和开源发布流程。

## 9.3.3 开源项目面临的问题



在探索优秀的开源项目的品质管理要素之后，我们转向了开源项目所面临的挑战。有许多问题是无法避免的，但是通过有序的管理和合理的规划，可以把这些问题转化为改进和进步的契机。

首先，从参与者的角度来看，开源项目可能涉及到来自世界各地，背景各异的众多贡献者。对于一个拥有上百甚至数千名贡献者的项目来说，如何有效地管理参与者、整合他们的提交(commit)以及处理合并请求(Pull Request)是一个重大的挑战。另外，随着开源项目的使用人数增多，用户提出的问题和需求也会随之增长。这就需要我们构建一套系统，以高效管理这些问题，处理"Pull Request"，并妥善安排项目进度，以保证项目的顺畅执行。

此外，开源项目也面临着安全和风险管理的问题。以新思科技在2021年发布的开源安全和风险报告为例，可以看出，无论是医疗保健、金融服务还是零售电商领域，大部分的代码库中都包含有开源代码，而其中存在漏洞的比例相当高。这表明，开源项目在安全和风险控制方面面临着严峻的挑战。

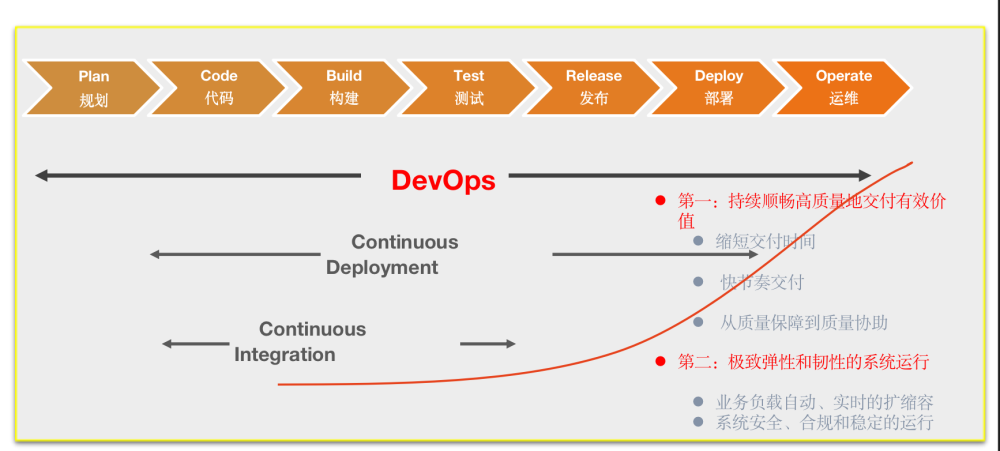
那么，我们应如何面对这些挑战，将开源的风险控制在可接受的范围内呢？首先，确保开源项目的安全性应是开发者、用户乃至整个开源社区的共同责任。每一位参与者都应该积极关注并报告潜在的漏洞，同时采取适当的安全措施保护自身的系统和数据。与此同时，开发者需要定期更新和维护开源组件，因为新版本往往包含了修复的漏洞和安全补丁。

组织和企业也需要采取一系列措施来管理开源风险，包括进行风险评估，审查和选择可信赖的开源组件，实施安全测试和代码审查，以及建立漏洞修复和安全更新的流程。通过这些措施，我们可以在享受开源项目带来的便利的同时，降低潜在的安全风险。

总的来说，面对开源项目的挑战需要我们以更广泛的视角和更严谨的态度来面对。这不仅需要开源社区、开发者和用户的共同努力，也需要我们建立更完善的管理体系，以应对不断变化和提升的开源环境。

# 9.4 代码的持续集成测试管理体系

## 9.4.1 持续集成测试管理体系概述



现在我们将重点关注如何通过构建一套持续集成测试管理体系来解决开源项目面临的挑战。在此之前，我们在DevOps的讨论中已经简要地提到了这个概念。这一体系的核心理念是，通过持续集成（Continuous Integration，简称CI）、持续部署（Continuous Deployment）以及持续交付（Continuous Delivery）等方法，将代码的开发、测试、交付以及部署等过程自动化，从而更有效地控制质量、提升效率。

在这个体系中，一个关键的概念是持续集成（CI）。持续集成是指，在开发人员完成代码编写并提交到代码仓库后，系统会自动对这些新的代码进行编译、测试和合并。这个过程是自动进行的，可以极大地提高工作效率，减少由于手动操作带来的错误。

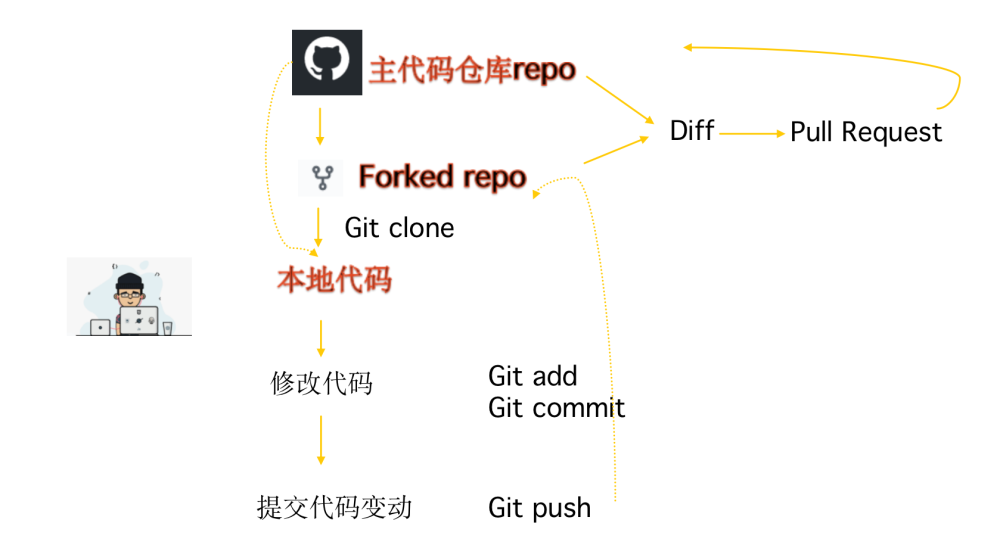
另一个重要的概念是持续部署，它是持续集成的延伸。在完成持续集成的基础上，持续部署将代码的构建、测试、发布和部署等过程全部自动化，使得新的代码能够在短时间内被安全、有效地部署到生产环境中。

持续交付则更加强调在代码被发布后，如何将其有效地交付给最终用户。持续交付的过程也是自动化的，它可以帮助我们在保证代码质量的同时，快速地满足用户需求。

我们可以看出，在DevOps的全过程中，持续集成、持续部署和持续交付都扮演着重要的角色。通过这些方法，我们可以实现持续、顺畅、高质量的代码交付，同时保证系统运行的安全、合规和稳定。因此，建立一个有效的持续集成测试管理体系，对于提升开源项目的质量和效率具有重要意义。

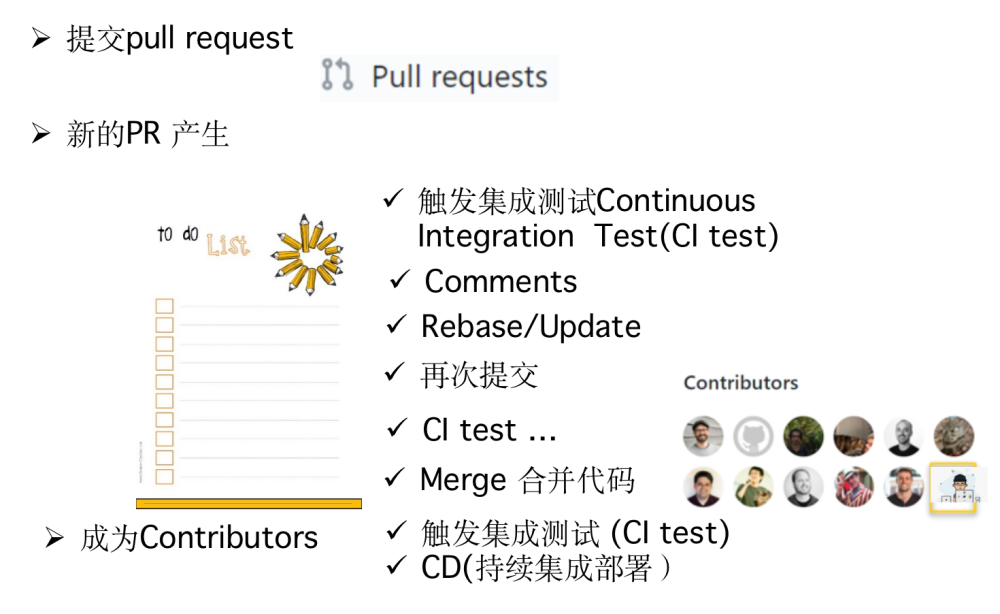
## 9.4.2 持续集成的原理与流程

持续集成的概念基于一种操作流程，这个流程是由多步动作构成的：从开发者提交代码开始，代码经过一系列的处理，最终完成集成。GitHub定义了持续集成为频繁地将代码提交到一个共享的代码库中的过程，而首席工程师Martin Fowler对此的理解也与此相符。让我们来详细解读这个流程的每一个步骤。



首先，如果你是一个开发人员，当你提交合并请求（pull request）时，你需要在GitHub上进行交互。这个过程开始于一个公共的共享的主代码仓库。接下来的步骤包括fork这个仓库到自己的GitHub仓库，git clone这个仓库到本地，对本地的代码进行修改，关联你的fork的代码和主代码仓库，进行git add、git commit等操作，然后使用git push将修改推送到你的forked repo，最后提交一个Pull request。这个过程的最终目标是将你的代码合并到主代码仓库。

提交Pull request之后，将开始合并过程。这个过程包括触发持续集成的测试，代码审查，可能的代码修改，以及重复的评论、改进和测试，直到评审人认为你的代码可以合并。然后你的代码会被合并，可能进行持续集成测试和持续部署。作为新的贡献者，这个流程从你的Pull request到最终代码合并，实现了持续交付和持续部署的流程。



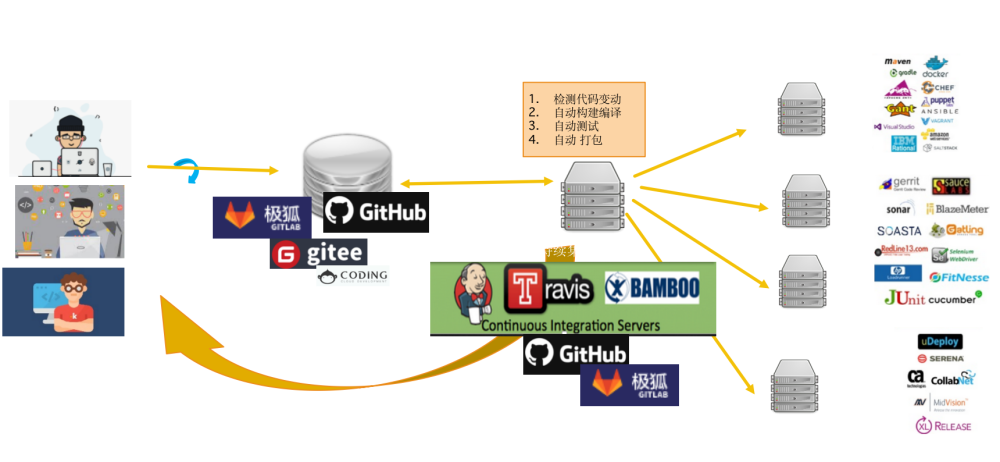
在描绘了这个流程之后，让我们看看GitLab对整个工作流程的推荐。这个流程开始于创建一个issue，然后创建一个合并请求（merge request），最后进行提交并将代码推送上去。此时，持续集成（CI）开始运行自动化测试用例。然后进行代码审查和讨论，这是一个不断变动、讨论、改进和持续集成的运行的循环过程。最后，你的变动被批准后，进行代码合并，关闭你的问题，进行持续部署，然后线上的运营开始。此流程与之前描述的是一致的，只是讲得更加详细，而GitLab将整个过程绘制出来。因此，这个工作流程被业界公认为推荐的工作流。。

持续集成带来的优点：

1. 快速发现错误：由于每次完成一点更新后，就将其集成到对应的分支，持续集成可以帮助我们快速发现错误。同时，错误定位也变得更加容易，因为每次集成的变动都相对较小，可以更精确地确定是哪个更新引入了错误。

1. 防止分支大幅偏离主干：如果不频繁进行集成，而分支或主干却在不断地进行更新，这将会导致未来集成的难度大增，甚至出现无法完成集成的情况。持续集成的流程可以确保主干和各个分支的同步，防止分支过度偏离主干的风险。

这两个因素都极大地提高了开发效率和代码质量，降低了因代码集成导致的风险，这也是为什么持续集成在现代开发流程中扮演着如此重要的角色。



最后，让我们从工具链的角度来看持续集成。当开发者提交代码时，代码会进入代码仓库，如GitLab、Git、Coding、GitHub等。提交后，会触发一些集成服务器，如GitLab自己的服务器或者第三方的服务器，如Travis CI、Jenkins等。这些服务器会检测代码的变动，进行自动构建、编译，以及自动的测试和打包。完成后，服务器会给出反馈结果，通知开发者是否成功，是否通过了测试。然后再通知开发者，如果通过了，可以进行后续步骤；如果未通过，可能需要进行代码调整。当代码完全通过并合并后，就可以进行打包发布了。

## 9.4.3 持续集成测试流程管理工具

我们将专注于三种在持续集成测试流程管理工具平台上广受欢迎的工具：GitHub的Actions，第三方平台Travis CI，以及知名的Jenkins。

首先，我们先研究一下GitHub Actions。Actions是GitHub自有的工具，允许你通过自定义代码来自动化和执行整个软件开发流程。官方的定义描述为：GitHub Actions可以用来自定义和执行你的软件开发工作流程，这些工作流程可以在GitHub提供的服务器上或者你自己的服务器上运行，通过使用YAML文件来描述CI/CD流程。总的来说，GitHub Actions的核心就是用YAML文件描述CI/CD流程，然后在GitHub的服务器或你自己的服务器上执行这些流程。

接着，我们来看看第二个工具，Travis CI。Travis CI是一个第三方的持续集成平台，它提供了持续集成和部署的服务。通过将其与GitHub等代码托管平台集成，并在代码仓库中添加配置文件，你就能定义你的持续集成流程。

最后，我们来看一下Jenkins。Jenkins是一个领先的开源自动化服务器，它支持在各种平台上运行，且提供了丰富的插件生态系统，可以支持各种工具和技术的集成。通过Jenkins，你可以自定义和管理CI/CD流程，配置构建和测试环境，并监控和报告构建的状态。

1. **GitHub Actions**

在GitHub Actions中，我们可以用YAML文件来定义和描述持续集成流程。以一个Python包的持续集成流程为例，YAML文件中的各个字段有特殊的含义。



首先，name字段用于设定Action的名称，例如在这个示例中，我们命名为"Python PA"，这表示这是一个用于Python包的Action。

其次，on字段用来指定需要监听的事件。在这个例子中，我们设置为监听push和pull request两个事件，并且只针对master分支。如果你需要监听多个分支的事件，可以在这个字段中列出所有需要监听的分支。

接下来是jobs字段，这是描述具体持续集成配置信息的地方。例如在这个示例中，我们设置了一个叫做"build"的job。在这个job中，runs-on字段设定了构建环境，这里我们使用了最新的Ubuntu版本。strategy字段指定了需要进行测试的Python版本，这里我们设置为2.7，当然你可以添加其他版本来进行多版本测试。

然后，在steps字段中，我们定义了要执行的一系列步骤。这些步骤包括安装必要的工具和配置测试环境。在这个例子中，我们首先升级了pip工具，然后安装了两个工具，一个是Python相关的静态代码分析工具，另一个是Pytest，用于执行Python单元测试。如果你需要查询这些工具的具体名称和参数，可以在GitHub Actions的官方网站上找到相关信息。

GitHub Actions也提供了许多模板，这些模板覆盖了各种编程语言，方便你构建自定义的Action。这样一来，无论你使用哪种编程语言，你都能方便快捷地使用GitHub Actions来构建和管理你的CI/CD流程。

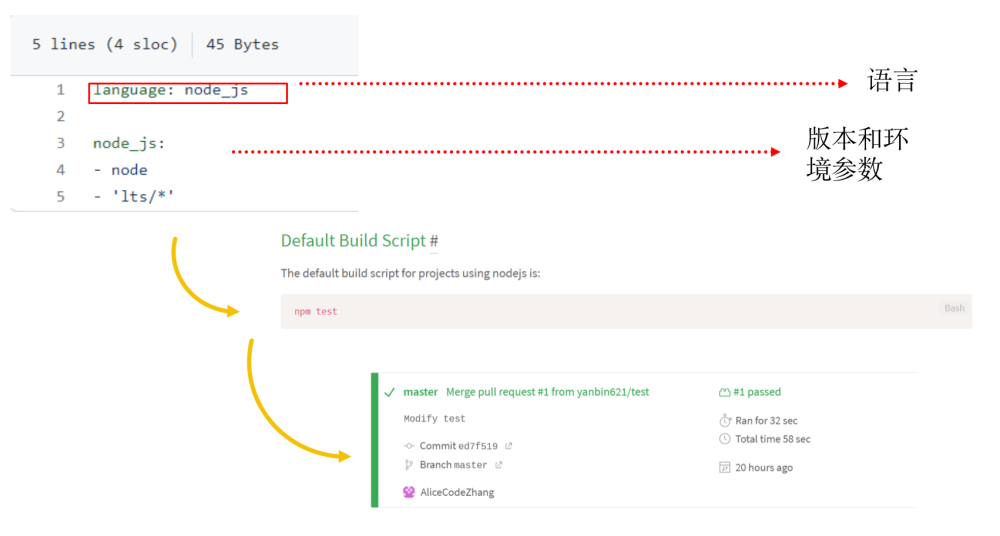
1. **Travis CI**

Travis CI是一个广受欢迎的第三方持续集成服务，主要用于构建和测试GitHub上的CI/CD流程。当GitHub上有新的pull request生成时，GitHub会通知Travis CI，此时，Travis CI可以开始进行相应的构建和测试任务。完成后，Travis CI会将结果反馈给GitHub，最终你可以决定是否将pull request合并到主分支。这是Travis CI典型的使用流程。

Travis CI同样使用YAML文件来描述CI/CD流程。例如，我们这里有一个针对Node.js应用的YAML文件示例。文件中，我们首先定义了项目的编程语言为Node.js，并指定了Node.js的版本为2，另外还定义了环境参数。

有了这样的配置，Travis CI就能根据这个YAML文件运行CI/CD流程了。值得注意的是，Travis CI为不同的编程语言提供了默认的构建脚本。对于Node.js来说，默认脚本会运行npm test命令来执行测试。这意味着，只需要在YAML文件中编写好测试用例，Travis CI就能自动运行这些测试。这大大简化了Node.js的持续集成过程。

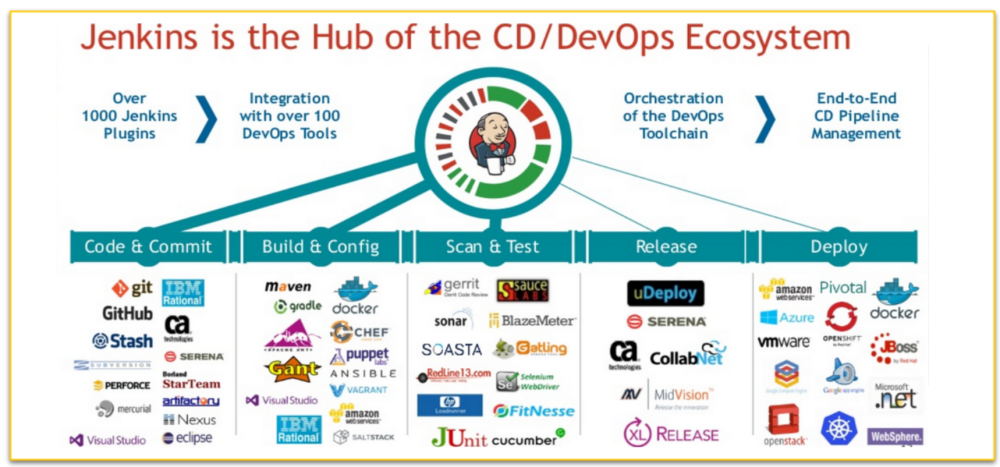
最后，Travis CI会生成构建报告，报告中包含了构建的状态、运行时间以及其他有用的信息。这些信息可以帮助开发者快速理解和调试可能存在的问题，从而提高开发效率。



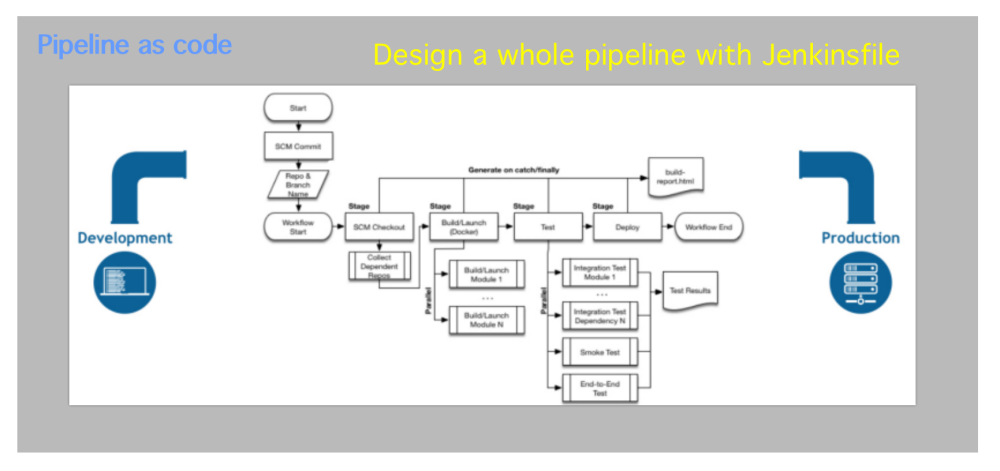
1. **Jenkins**

Jenkins是一种历史悠久的开源自动化服务器，用于实现持续集成（Continuous Integration，CI）和持续部署（Continuous Delivery/Deployment，CD）。无论是开源产品还是私有产品，都可以使用Jenkins作为其持续集成管理工具。

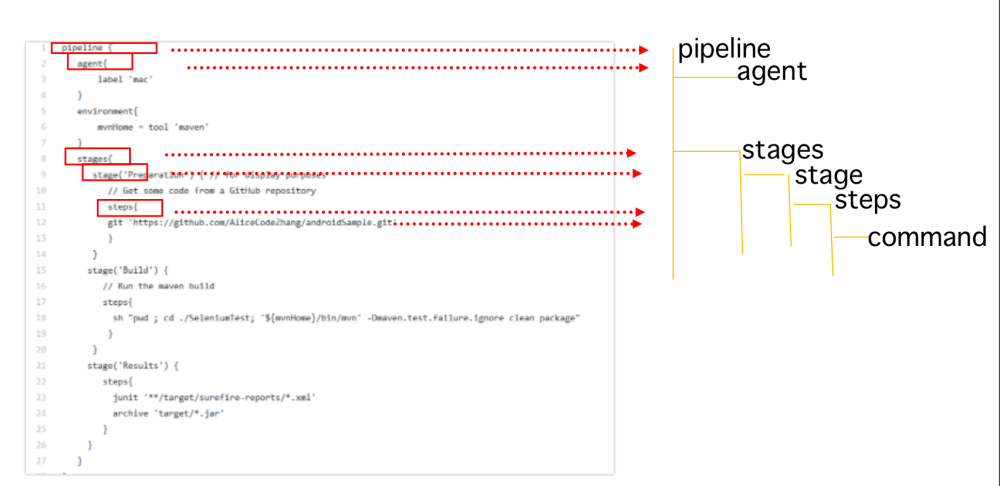
Jenkins社区非常活跃，有超过25万全球用户，600多位项目贡献者，展示出其强大的社区力量。此外，Jenkins是持续交付基金会（Continuous Delivery Foundation）的成员，提供了1800多个插件，能支持整个CI流程，并支持所有源代码管理和版本控制工具。这意味着Jenkins适用于几乎所有类型的项目，无论是开源项目还是私有项目。



Jenkins在CI/DevOps生态系统中起着中心枢纽和桥梁的作用。通过1800多个插件，它支持整个DevOps工具链，并实现从端到端的CI流程管理。这包括最初的代码提交（如Git和GitHub）、构建工具（如Maven、Gradle）、代码扫描和测试、发布步骤以及部署等。



Jenkins在其发展早期就提出了基于Pipeline的持续集成方式。这种方式用一个叫做Jenkinsfile的文件来描述整个CI/CD流程，这种文件是一种特殊的代码文件。你可以通过编写Jenkinsfile来设定如何进行代码提交、工作流程中的各个阶段（如代码检出、编译、测试、部署）等。Jenkins可以解析这个文件，并执行其中定义的流程。



Pipeline的语法非常简单。首先，你需要使用"pipeline"关键字来标识一个新的Pipeline。然后，你可以使用"agent"关键字来指定执行Pipeline的环境。"stages"关键字则用于定义一系列的阶段，每个阶段可以包括一系列的步骤。

此外，Jenkins也支持分布式运行。你可以设置多个Jenkins节点，每个节点可以运行在不同的环境（如Windows、Mac、Linux等）。然后，Jenkins的主服务器（也就是Master）可以将任务分配给这些节点执行。这样，你就可以在不同的环境下运行你的CI/CD流程了。

选择正确的构建工具对于提高开发效率和改善开发体验至关重要。在当前的市场上，有许多不同的构建工具可以供选择，包括GitHub Actions、Jenkins、CircleCI等，每个工具都有其自身的优点和缺点。

GitHub Actions是GitHub提供的持续集成服务，对于使用GitHub的开发者来说，其最大的优点是配置简单且无需额外的服务器。GitHub Actions还支持自我托管的服务器，这对于需要在私有网络环境中运行CI/CD流程的组织来说非常有用。GitHub Actions对私有仓库也提供了支持，尽管使用共享Actions服务器需要付费，但也可以选择自建基础设施。

另一方面，Jenkins是一款开源的、具有强大插件系统和社区支持的持续集成工具，它允许开发者进行大量的定制，并可以部署在自己的服务器上。这使得Jenkins在许多公司中受到运维人员的喜爱，因为它们能够完全控制自己的环境。

最后，CircleCI是一款第三方的CICD平台，它的配置非常简单，并且可以与GitHub无缝集成。然而，对于开源项目，它在免费支持方面有一些限制，例如Runner的使用限制。

总的来说，每个工具都有其自身的优点和缺点，应根据具体的需求进行选择。GitHub Actions在简单配置和免费支持方面表现出色，Jenkins在插件丰富和自定义性方面表现优秀，而CircleCI则是一款简单易用的第三方CICD平台。无论选择哪种工具，都需要确保它能够满足项目的需求。

# 9.5 基于开源的代码质量管理体系

## 9.5.1 传统软件测试体系的变革和发展



在许多传统的测试体系中，如金字塔测试理论，主张尽量减少对 UI 的测试，增加单元测试，并重点覆盖 API 组件和集成测试。这样的体系主张可以提高整个测试执行的速度，从而减少人力资源和金钱等资源的消耗。

然而，随着产品形态的不断变化，以及每个公司的产品架构的差异，会衍生出许多其他相关的测试体系。比如在 JavaScript Web 的测试中，我们可以看到一个杯子模型。在这个模型中，首先进行静态代码扫描以支持代码质量，然后进行单元测试覆盖，最后是端到端的测试。这一模型认为，大部分的测试工作是在集成测试中进行的，这一部分非常重要。

再来看微服务的测试，我们可以看到一个蜂窝方式的测试架构。在这个架构中，集成测试是最重要的模块，测试案例要足够多，覆盖率要足够高。两端是实现底层的内容。这里的关键是，我们需要根据不同的产品形态，摸索出适合自己产品的测试构建方式。

同时，我们需要理解，一个完整的质量管理体系不仅仅涉及测试，还包括产品质量管理的方方面面。这一体系涵盖了研发过程的监控、需求评审和记录、测试用例管理、代码编写规范、分支管理以及代码准入检测等。之后，还包括自动打包和回归检测等自动化流程，然后是各种各样的测试方法，包括模拟测试、手动测试、接口测试、埋点测试等。最后是产品发布阶段，涉及自动发布、灰度发布和用户反馈的监控。在线上阶段，还需要进行业务监控，包括埋点监控、性能监控、舆情监控和用户监控等。最后是与运营相关的一些测试，以确保整个质量管理体系的完整性。

总的来说，传统的测试体系正在经历一场重大的变革，以适应不断变化的产品形态和技术环境。无论是从金字塔到杯子，还是到蜂窝，我们都需要根据我们的产品和环境来定制我们自己的测试策略和体系，同时也要理解，测试只是质量管理的一部分，一个全面的质量管理体系还需要包括需求、编码、分支管理、自动化、发布和运营等多个方面。

## 9.5.2 开源项目的质量管理体系

在传统的质量管理体系和开源项目的质量管理体系之间，存在一些明显的差异。这主要体现在开源需求问题评审和开源测试体系这两个关键部分。

对于开源需求问题评审，开源项目常常使用 issue 跟踪系统来管理需求和问题。这种方法通过在 issue 中使用不同的标签和标记（例如：文档，增强需求等），以便于需求的分类和记录。这样，社区成员就能在 issue 下进行讨论，提出解决方案，而需求和问题之间的关联也可以方便地追踪和追溯。

然后，我们来看看开源测试体系。这是开源质量管理体系中的关键组成部分。在开源项目中，测试方法和传统项目有所不同。常见的测试方式包括使用自动化测试工具和测试框架来执行单元测试、集成测试和功能测试。开源社区通常鼓励社区成员参与测试，并提供测试报告和反馈。此外，开源项目还可以利用持续集成和持续交付的实践，自动化执行测试，确保代码的质量和稳定性。

但是，对于开源项目来说，质量管理体系还需要考虑社区参与的特性和开放性。由于开源项目通常具有跨平台、多版本和多环境的特点，因此必须保证测试的全面性和覆盖性。在测试过程中，还可以利用社区的力量，鼓励社区成员贡献测试工具、用例和测试数据，以提高测试效率和质量。

总的来说，基于开源的质量管理体系可以吸取传统质量管理体系的经验，同时针对开源项目的特点进行调整和补充。通过使用 issue 跟踪系统来管理需求和问题，以及采用自动化测试和持续集成的做法，可以有效地提高开源项目的质量和稳定性。与传统项目相比，开源项目更注重社区的参与和开放性，因此需要利用社区的力量来推动质量管理的持续改进。

首先，开源项目的需求评审通常会使用 issue 跟踪系统进行管理。一旦需求被提出，开发者可以提交一个 Pull Request（PR）以提供代码实现。这个 PR 接着会通过自动评审流程，进行自动化测试、持续集成等环节的评估。如果自动评审通过，会进入人工评审阶段，人工评审团队将审查代码变更，确保其满足需求，并与开发者进行讨论。这个过程通常是循环迭代的，直到问题得到解决并通过评审。最后，在人工评审通过后，将触发一个评审请求（Review Request）。这个评审请求需要至少一个评审人员（Reviewer）的批准，并通常会设有多个待审查人员（Pending Reviewers）。只有在所有待审查人员都批准后，代码才会真正合并。

其次，版本控制在开源项目的发布中非常重要。通常，每个分支会进行版本控制，代表一个发布版本，如版本号5.0.24.3。每个分支的每个标签（Tag）都代表一个具体的发布版本。在发布后，会提供新功能的描述，并生成最终的源代码发布。值得一提的是，GitHub的Marketplace提供了一个平台，用于发布自定义的Actions。这些Actions是持续集成流程中的一部分，发布在Marketplace上可以确保所有使用者都能获取到相同的版本，使得用户可以使用这些Actions来运行测试用例，确保他们的代码质量和提交符合预期。

对于issue、评审、分支管理，整个过程实际上是 issue 到 code 的一个过程转变，过程中存在两个角色：contributors和reviewers。contributors不仅会提出issue，还会提供代码。当contributors提交issue时，需要注意标题的简洁规范和内容的清晰。例如，报告bug时，需要说明问题的大概原因、相关版本、硬件环境、重现步骤，以及提供相关的日志和截图信息。同时，reviewers需要及时进行评审，覆盖issue涉及的所有问题，将其分类到合理的模块，并设置好目标里程碑（milestone）。如果相关的PR已经合并，那么也需要及时关闭bug。

总结一下，基于开源的质量管理体系在需求评审和版本控制方面具有独特的实践。通过使用issue跟踪系统进行需求管理，结合自动化评审和人工评审，开源项目可以确保代码质量并满足需求。同时，通过版本控制和使用GitHub Marketplace，可以提供可靠的版本发布和定制化的持续集成流程，使开源项目更可靠且易于使用。

## 9.5.3 开源项目的测试体系

现在我们将讨论测试体系，将其分为几个主要部分。



首先是静态代码扫描类测试工具，包括静态代码扫描和代码格式检测。这些工具可以检查代码格式以及进行动态代码扫描、安全检测和漏洞检测。此外，现在还提到了第三方依赖检测。由于安全漏洞问题日益严重，人们越来越重视将这些扫描类检测纳入整个持续集成和持续交付（CICD）流程中。这些扫描可以根据不同的编程语言进行分类，例如JavaScript、Android和C++，市场上有支持这些语言的工具。另外，公司对某个安全扫描工具的认可也很重要，因为有些工具本身可能存在一些问题或漏洞，公司通常会有认证方式来确保可靠性。

其次是功能测试，确保代码没有安全问题和漏洞等。在功能测试中，需要进行单元测试、API接口测试和端到端测试，以确保代码的功能正常运行，并且没有太大的偏差。除此之外，还需要进行性能测试，例如验证汽车的燃油消耗是否低、手机的电池消耗是否合理以及浏览器的启动速度和网页加载速度等。性能测试还包括页面加载速度和每秒承载访问量等压力测试、稳定性测试和兼容性测试，特别适用于移动设备和客户端应用程序。对于移动设备，需要测试其在iOS和Android上的兼容性，并确保与不同品牌和厂商的设备兼容性良好。

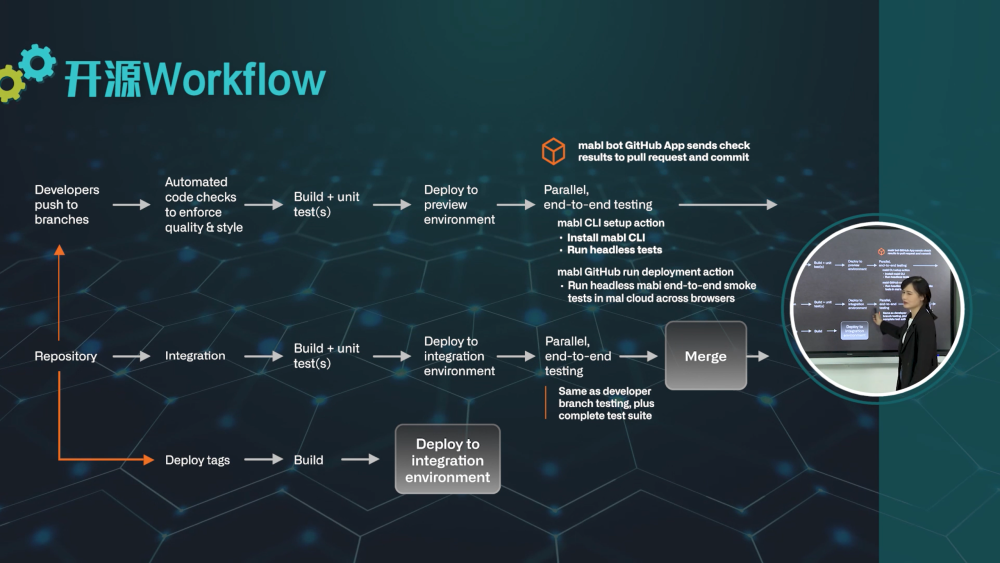
我们还需要考虑将这些测试流程整合到持续集成中的步骤中，这需要一定的设计策略。传统的公司测试中通常会有几种概念，例如冒烟测试（SMOG测试），它是对变动进行小规模测试的一种方式；还有Sandy测试，它涉及对与变动相关的部分进行测试；另外还有回归测试，用于验证变动是否导致了原本正常的功能出现问题；最后，在发布之前可能会进行全面的测试。

在开源项目中，如果没有专门的测试团队，通常会进行不同的设计和测试流程。让我们看一下一个工作流程的示例，它将不同的设计和测试分为不同的测试流。

首先是第三个步骤，即PR验证（PR validation）。当有一个Pull Request提交时，会将相关的测试放入一个CI中，通过这个工作流程进行运行。接下来是第四个步骤，当决定进行版本发布时，会将测试放入发布容器的CR（Continuous Release）中，其中包括端到端的一些测试。这些测试会在多个环境中运行。

第三个步骤是运行测试的工作流程（run test workflow）。在这里，它使用一个DOC image环境作为最终产品进行测试。这是发布的CI。另外，还有一个Lightning image，为什么每天晚上运行呢？因为知道每次更新DOC image可能需要较长时间，所以每天晚上会进行一个Lightning image的变更CI。此外，还有一个load CI。根据产品的形态，可以定义不同的CI流程。对于每个Pull Request的合并，可以打上一些标签，例如发版时运行release的CI，或者提交时每天晚上运行全面的测试。

例如安全测试和性能测试通常需要较长的时间，如果有人提交了一个Pull Request，而测试需要一整天才能完成，就不会等待测试结果。因此，这些测试通常会放在Nightly或发布测试的CR中。但是，如果每个Pull Request都包含了所有的单元测试、API测试、静态代码扫描等快速测试，那么它们就会被包含在内。根据产品的形态和需要，每个动作的形态都需要定制不同的CI流程。



让我们来看一个来自GitHub的例子，这是一个名为Mabl的AI测试工具，他们是如何定义他们自己的CI工作流程的。首先，对于开发者提交到分支上的代码，它会自动进行代码检测和语法检查，然后进行编译和运行单元测试。接下来，它将部署到预览环境，并进行一些端到端测试。同时，它会启动Mabl的CLI工具，运行一些幸福度测试。完成这些步骤后，它会生成一个关键标签的APP，并将检查结果提供给所有相关的Pull Request的作者，以查看测试结果是否通过。

如果你是一个集成动作，它也会编译和运行单元测试，然后部署到集成环境，并进行一些端到端测试。然而，我们可以看到这两个流程是相似的，只是部署的环境不同。一个是部署到预览环境，另一个是部署到集成环境。最后，如果你进行部署，并在标签中指定了不留的标签（deploy的tag），它将进行编译并部署到集成环境。因此，它使用不同的环境进行隔离。这是因为如果你是一个Web应用程序，那么你肯定至少有两个分支，一个是测试分支，一个是预览分支，最后一个是线上分支。所以，在每次线上更改之前，你必须非常小心。这就是为什么会有AB测试和灰度发布。在预览环境中，你可以持续进行一些PR合并、编译和其他动作，这就是Mabl工具流的设计。



在开源项目管理中，要实现整体的代码质量管理，可以划分为几个部分。首先是代码质量，需要进行持续集成和持续交付（CICD），包括代码扫描、单元测试、功能测试和兼容性测试等。对于一些无法在CICD中运行的测试用例，如压力测试和性能测试，可以创建一个独立的CI流程，例如Lightning流程。另外，在部署和发布时，可以创建一个独立的CI流程，其中包含上述所有测试，以及一些压力测试和性能测试，从而形成真正的部署CI和持续交付（CD）流程。

同时，还需要进行缺陷和需求管理，无论是需求还是缺陷，需要对issue进行标签标注、里程碑管理等。与代码质量相关的其他方面包括代码分支管理，如标签打法和分支发布等。除了以上方面，还需要进行项目管理，包括issue管理、需求管理等。项目管理还涉及文档，如wiki、文档标签和描述等。回到我们最初讨论的什么是优秀的开源代码，文档的重要性不可忽视。最后，可能需要启用讨论功能，这是自由选择的。这些范畴属于项目管理。因此，对于整体的开源代码管理，包括代码质量、代码分支管理和项目管理等，只有将这些方面都整理清楚并做好，才能使开源项目和开源生产流程取得良好的结果。