中南大学

硕士学位论文

持续集成在现代软件开发中的应用与研究

姓名：徐仕成

申请学位级别：硕士

专业：计算机应用技术

指导教师：杨邦荣

20070501

摘 要

在软件开发过程中，尤其在需求经常变化的软件项目中，经常因集成过程出现问题导致项目拖延或者崩溃，集成问题已经成为软件开发过程中存在的主要风险之一。为了改进传统集成方式的不足，现代敏捷软件开发提出了持续集成。持续集成实践的应用可以最小化了集成工作的风险、明确项目开发进展状态、改进软件开发过程及提高软件项目的交付质量。

但目前对于持续集成实践还存在一些问题：缺乏对持续集成整体了解；缺乏对整个持续集成过程支持的解决方案；应用不够，缺乏实际项目实践指导。本文就是基于这些问题展开对持续集成研究。

本文在总结比较几种经典软件开发模型中集成方式优缺点基础上，从理论根源分析了持续集成思想的来源，并且对于其关键技术，统一源代码、自动化过程和验收测试的实现原理作了深入研究，总结了在软件开发过程中应用持续集成的价值。接着本文在一组开源工具的基础上，开发实现了持续集成插件，并且将其应用于项目管理中，提供了一种支持整个持续集成过程的工具解决方案。该方案基于开源项目，可扩展性强，在实际企业应用中运行良好，具有很强的实用价值。

本文通过用户管理系统项目实践了持续集成的整个过程，给出了持续集成中源码配置、构建脚本编写、代码规范检查、验收测试、持续反馈和自动化部署的方案，并且进一步总结实施持续集成的建议。在论文的结束部分，对本文进行了总结以及说明了关于课题进一步的研究方向。

关键词 自动化过程，验收测试，持续集成，项目管理平台

ABSTRACT

In the process of software development, the integration problem often leads to the delay or collapse of the projects, especially those requirements are often changed. Integration process has become one of the main risks in software development. In order to improve the disadvantage of traditional integration, agile software development suggests using continuous integration. The practice of it can minimize the risk, grasp the project rhythm, achieve better management, and enhance the delivery of software quality.

However, there are a lot of problems on practising continuous integration currently. Firstly, programmer may not know the all and the one. Secondly, we lack a solution to support the whole process of it. Lastly, we lack guidance because of few practices. This paper is doing research into these problems.

Based on the summary and compare the ways of integration in classic software development model, this paper analyses the origin thoughts of continuous integration. Then, the paper analyses the basic theory of it, for the key practices, such as the single source code, automated process, and bug verification test, this paper makes a deep research, and it gives the value of using it in software development. In succession, this paper implements a plug-in of congtinuous integration on the basis of a series of open source framework. To make it further, it put forward a solution to support the whole process of continuous integration by using the plug-in into project management system. The platform is based on open source framework, has a wide extension. It runs well in practical enterprise applications and has a great value.

The paper practises the entire process of continuous integration through the User Management System Project, and gives the solution of configuring source repository, coding build script, checking code specification, build verification test, continuous feedback and automated deploy. Then, it gives the suggestions of using continuous integration in practice. In the end, we summarize the work of our paper and introduce directions for further study.

KEY WORDS automation process, build verification test, continuous integration, project managerment system

原 创 性 声 明

本人声明，所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了论文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中南大学或其他单位的学位或证书而使用过的材料。与我共同工作的同志对本研究所作的贡献均已在论文中作了明确的说明。

作者签名：徐仕成　日期：2007年5月19日

关于学位论文使用授权说明

本人了解中南大学有关保留、使用学位论文的规定，即：学校有权保留学位论文，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以采用复印、缩印或其它手段保存学位论文；学校可根据国家或湖南省有关部门规定送交学位论文。

作者签名：徐仕成　导师签名：杨和荣　日期：2007年5月19日

第一章 绪 论

1.1 引言

软件技术同计算机硬件技术一样发展了几十年，但软件的生产水平还远远不能满足社会需求。软件开发的过程从来就不是轻松愉快的，人们对于软件开发过程中的问题一直没有停止关注过[1]。从最初瀑布模型的提出，到最新的敏捷软件开发技术都无一不是为了解决软件开发过程中的问题。近年来，关于软件开发方法的研究，进展非常迅速，但是在实际开发实践过程中却并非如此。许多程序仍然是错误百出，充斥着过时的技术，从而无法满足用户需求。软件工业界和学术界的研究者们，基本上已经解决了七十年代和八十年代在编程中遇到的问题，并发展了相应的技术，但是直到现在，这些技术中的大部分仍然没有在软件项目中广泛采用。在软件开发的领域里有各种各样的“最佳实践”，它们的价值经常被人们谈起，但是似乎很少有真正得到实现的[2]。首先，由于这些最佳实践主要都来源于一些顶尖的软件开发团队中，普通的程序员由于缺乏信心，认为这些实践很难得到实现；其次，任何一项新技术的接受都需要一定的代价，由于缺乏对这些实践的整体了解以及害怕付出，从而排斥这些新技术。Sridhar Raghavan 和 Donald Chand 的研究表明[3]，一项新技术从诞生到被工业界广泛采用大约需要 5 到 15 年的时间。

集成问题一直是软件开发过程中存在的主要风险之一，很多软件项目的失败都是由于在最后集成时出现问题[4]。持续集成作为敏捷软件开发中的一最佳实践，就是其中之一，其价值已被越来越多的团队所接受，但很少有项目真正使用它，主要原因是由于实施人员缺乏对持续集成整体的了解，而现实中又缺乏对整个持续集成过程支持的解决方案。

1.2 课题来源

本课题来源于对笔者在文思创新软件技术公司（Worksoft）一个失败的软件外包项目的总结和分析基础上。该项目是一个典型的对日外包项目，日本分包商只提供与分包内容相关的各个模块功能书，而整个系统的需求说明不是很明确。由于缺乏对外包项目中经常性的需求变更的了解[5]，项目一开始就按照每个分包内容模块来开发，完成分包模块后再进行统一集成测试。前面开发模块的过程进行非常顺利，但在准备集成时，就出现了严重的问题：

(1) 各自在自己机子上工作很好的模块集成在一起时却不能很好的工作，集

成时出现的情况根本不是想要看到的结果，甚至在开始的时候还出现了编译错误。当在调试 Bug 的时候，问题越来越严重，通常是解决一个 Bug，却带来更多的 Bug，每天开会讨论的主要内容总是怎么去解决 Bug 与加班，但问题总不能得到解决。

(2) 在集成之后意识到自己理解需求的确存在偏差，但此时来修改，将代价太大，只能不停地打补丁，不停地加班，项目成员极其狼狈与疲惫，彼此推卸责任，而且谁都不想去沟通交流，项目组工作气氛很压抑。后面用户提出的一个小小的需求变更更是需要导致整个整体框架的改变。

本应该在 9 月底交付的项目，后来拖到了 12 月份，而且交付的软件产品质量很成问题，后期维护也相当困难。致使项目组，客户都存在很多怨言，而且公司也不满意。分析项目失败原因，尽管日本分包商存在很多问题，比如只重视合同和验收的两头管理，而忽视对软件构架、软件开发过程、软件维护等其他软件外包阶段的监督和管理。但主要的问题还是在我们承包方这边，我们采取的传统的软件开发方法，忽视了软件开发过程中的风险，缺乏与日本分包商以及用户的沟通与交流，而其中最为重要的一点是我们忽视了软件开发过程中风险最大的一个环节——集成。由于没有采取经常性的集成，而是采取了传统瀑布模型中的集成方式，导致进入“集成地狱”（Integration Hell）[6]，最终该项目失败。

在总结了以上原因基础上，为了改进软件开发过程，避免因集成问题而导致项目的失败，我们在后面的项目开发中采用了经常性的集成，而且按照极限编程思想的观点，如果集成是好的，那我们应该进行经常性的集成，每时每刻都进行，即持续集成。持续集成能最小化软件开发中集成的风险，改进软件开发过程。

在进行持续集成的实践中，发现持续集成思想由来已久，RUP（Rational 统一过程）开发中“迭代递增”软件开发方法与持续集成的思想就极其相似，微软公司软件开发过程中的优秀实践“每日构建”可以说就是持续集成的雏形。

人、过程和工具永远是一个项目成功的关键，一个好的思想总需要过程和工具的支持，而关于支持持续集成的过程与工具解决方案目前还很少见，于是在分析了持续集成的实现原理后，基于持续集成思想，在一组开源工具基础上，设计开发了持续集成插件，并且将其应用于项目管理中，搭建了项目管理平台 PMS，用来支持软件项目实施持续集成整个过程管理。

1.3 课题研究现状及研究意义

1996 年 Kent Beck 提出了极限编程 XP 的概念，在他编写的 Extreme Programming Explained: Embrace Change 一书中总结了 XP 的 12 个最佳实践，持续集成这个术语最初就源于其中一个最佳实践，参见文献[7]。尽管持续集成概

念是由 Kent Beck 提出，但是他在书中只是提出其大概思想，还由于 XP 刚出现时受到很多质疑，认为采用持续集成实践就必须采用 XP 的软件开发模式，而 XP 开发模式中很多实践（如结队编程）都不为业界所接受，人们对持续集成实践产生了很多误解[8]。

2000 年，XP 思想发起者之一的著名软件大师 Martin Fowler 专门以“持续集成”为题写了一篇非常著名的文章，该文章以 ThoughtWorks 公司软件项目持续集成实践为基础详细介绍了持续集成的价值[9]。他认为无论是否采用 XP 的开发模式，都应该采用持续集成的开发实践，但是他在文章当中只提到了持续集成实现的几个要点以及其基础实践，没有说明怎样来实现这些实践，在实际项目应用持续集成，还是存在很多问题。尽管由于这篇文章的流行，使得很多项目开发团队认识了持续集成的价值，但真正应用于项目实践中还是很少。

国内对于持续集成的应用实践开始于最近两年，针对持续集成的讨论一直敏捷中国 AgileChina 用户组的热门话题，但讨论的内容多以问题出现，缺少解决方案。

为了更多的软件开发人员重视这一实践，最近有国际软件组织专门召开了 CITCONF 会议[10]，而且今后每年都会举行，会议的主题就是讨论关于持续集成的应用。Jesper Holck 和 Niles Jorgensen 也在一篇调查报告中指出：持续集成的应用为 FreeBSD 和 Mozilla 这两个大型项目提供了质量保证[11]。

总的来说，对于持续集成实践的应用当前存在的主要问题为：

(1) 开发人员已经认识到持续集成的重要价值，但缺乏对持续集成整体了解，不敢也不知如何实施持续集成。

(2) 业界缺乏对整个持续集成过程支持的解决方案。

(3) 持续集成应用不够，缺乏实际项目实践指导。

针对这几点问题，本课题希望通过对持续集成的研究以及在 Worksoft 实际项目中实施持续集成的成功经验，使广大的开发人员可以真正认识持续集成，理解它的实现原理，了解它的价值，以最终在软件项目的开发过程中采用持续集成实践。

在软件开发过程采用持续集成的开发及管理方法，可以改善软件开发过程，提高软件项目成功的交付能力，对于发布高质量软件产品起到重要作用。

1.4 论文研究工作

本文在深入研究持续集成实现原理的基础上，开发实现了持续集成插件，进一步地将其应用到项目管理中，提出了一种支持持续集成整个过程的解决方案，并且通过具体项目实践了持续集成的整个过程。本文中关于持续集成的应用部

分，都是建立在 Worksoft 持续集成插件开发以及其应用的项目管理平台 PMS 实际项目之上。本文前面两章是关于持续集成的基础理论部分，后面两章是关于持续集成的应用实践。论文主要研究工作如下：

(1) 以集成过程为视角分析了软件项目中出现的问题，说明软件集成方式在软件过程中的重要意义。

(2) 深入研究了几种经典软件开发模型中的集成方式，包括“Big-Bang”集成模式、“迭代递增”集成模式以及“每日构建”模式，并且比较这几种集成模式的优缺点，从理论根源分析持续集成思想的来源。

(3) 针对开发人员缺乏对持续集成的整体了解等问题，从技术层面分析了持续集成实现要点及关键实践，对于其中自动化过程和测试过程的实现原理进行重点研究，并总结了在软件开发过程中应用持续集成的价值。

(4) 针对当前应用缺乏支持持续集成整个过程的解决方案，开发实现了持续集成插件，并且将该插件应用于项目管理中，搭建了项目管理平台 PMS，用来支持软件项目实施持续集成整个过程管理。

(5) 针对持续集成应用不够，缺乏实际项目实践指导，给出了用户管理系统项目应用实践持续集成的整个过程，并对于其中实现要点给出了可行性参考方案。

(6) 根据应用实践总结实施持续集成的经验，并对课题的进一步研究方向提出参考意见。

第二章 软件集成方式比较

本章将首先说明集成方式在软件开发过程中的重要意义，然后分析几种经典软件开发模型中的集成方式，从瀑布模型中的 Big—Bang 集成模式，到 Rational 统一过程方法中的“迭代递增”模式，以及微软软件过程中的“每日构建”模式，最后到现代软件敏捷开发方法中的持续集成。比较总结了每种集成方式的优缺点，从理论根源分析现代软件开发方法中持续集成思想的来源。

2.1 软件集成方式的意义

软件集成是指一个软件开发过程[12]，在这个过程中，通过管理和技术手段，将已开发出的软件组件或代码单元，整合成完整的软件产品，并对该产品进行测试和验证以确保产品符合用户需求，通常在一些书籍中也称之为系统集成或集成测试。对于一个小的工程，集成可能只需要花费一上午时间就可以将现有分支程序组合在一起；对于一个大的工程，它可能需要花费几个星期甚至几个月，但无论任务规模大小，它们应用的原理是相同的。在集成时通常要考虑以下问题[13]：

(1) 在把各个模块连接起来的时候，穿过模块接口的数据是否会丢失。

(2) 一个模块的功能是否对另一个模块的功能产生不利影响。

(3) 各个子功能模块组合起来，是否达到预期要求的功能。

(4) 全局数据结构是否有问题。

(5) 各个模块的误差累积起来，是否会放大到不能接受的程度。

(6) 单个模块的错误是否会导致数据库错误。

集成在软件开发过程中有着非常重要的意义，任何软件产品要想工作的良好，都需要进行软件集成的过程，在采用任何的软件开发方法中都必须经历软件集成。在建筑工程领域，经常可以看到因为拙劣的集成而导致建筑倒塌，建筑工程师都很强调集成的方式。但是，在软件工程领域，集成方式的意义却经常被忽视。Steve McConnell 总结了一个好的集成方式可以带来下列好处：

(1) 易于诊断错误。

(2) 更少的错误。

(3) 在短时间内形成首次可工作系统。

(4) 增强信心。

(5) 增加工作完成的机会。

(6) 更可靠的预测计划。

(7) 更准确地了解项目情况。

(8)提高代码质量。

尽管集成方式如此重要，尽管很多项目的失败都是由于集成原因，但集成问题一直被忽视。这有多方面的原因，其中最为主要的是每一种集成方式都涉及到软件开发过程的整个流程，而且与软件开发中的过程模型有着密切的联系。从集成方式看，瀑布模型中的“Big—Bang”集成模式，Rational 统一过程方法中的“迭代递增”模式，以及微软软件过程中的“每日构建”模式，最具有阶段代表性。分析这些集成模式，可以得出现代敏捷软件开发方法中持续集成思想来源。

2.2 几种经典软件开发模型的集成方式

2.2.1 瀑布模型与“Big-Bang”集成模式

尽管现在看瀑布模型存在很多缺点，但它对于软件生命周期的阶段划分，有利于人们研究每一个阶段的规律和改进开发手段，其几乎成了现代软件开发过程的基础。瀑布模型最初由 Winston Royce[14]提出，它将软件开发过程划分为以下几个阶段：系统需求分析，系统设计，编码，系统集成测试和运行维护，每一阶段工作的完成需要通过文档确认，其如下图 2-1 所示。

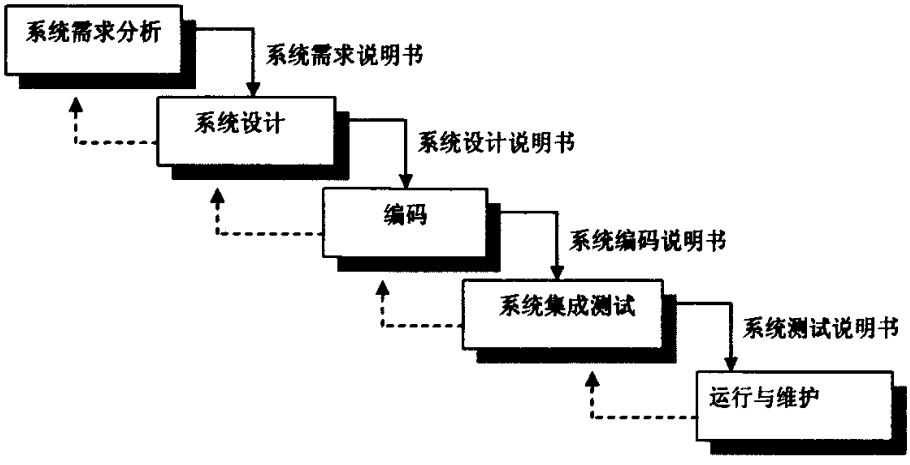


图 2-1 瀑布模型

从图 2-1 中不难看出，瀑布模型的本质是一种线性顺序模型[15]。它将软件开发的各个阶段严格划分，并要求在开始下一阶段工作之前必须完成上一阶段的所有工作，即前一阶段的输出是后一阶段的输入，各阶段之间存在着严格的顺序性和依赖性。其开发过程是一个严格的“下导式”过程，而确认过程是严格的“追溯式”过程，后一阶段出现了问题要从前一阶段的重新确认来解决。因此，问题发现的越晚，解决问题需要付出的代价就越高。

分析瀑布模型，其集成方式可以归纳为以下两个步骤：

(1) 理解各个模块需求，然后设计、编程、检查和调试，这个步骤是单元模块开发；

(2) 将各单元模块合并成一个非常大的系统，即系统集成。

当各模块在系统中首次被结合在一起时，肯定会出现很多新的问题。由于这些模块程序还没有在一起工作过，某一个模块出现的错误可能会导致整个应用程序所有的模块都需要调试。如下图 2-2 所示，模块 F 出现的问题，导致整个应用程序的错误。为了发现问题的根源，所有的的 8 个模块都需要调试。正是由于这个原因，通常把这种集成方式称之为“Big-Bang”集成[16]。

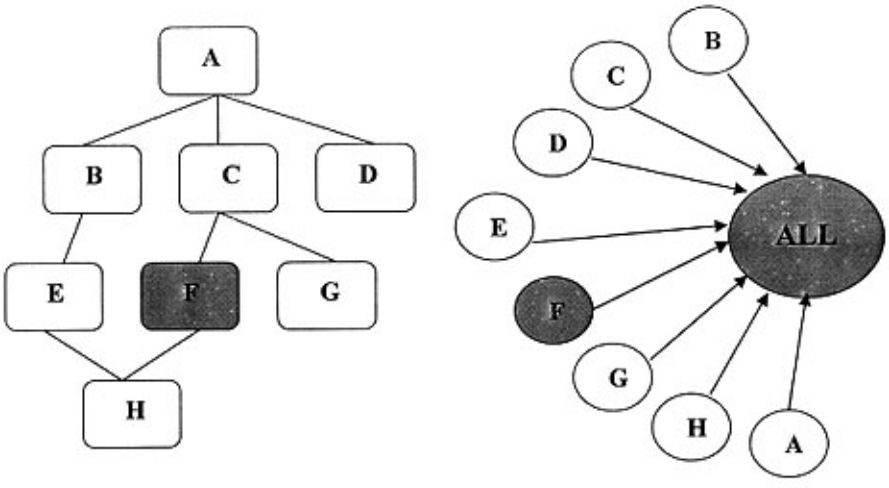


图 2-2 Big-Bang 集成模式

分析“Big-Bang”集成模式，它有如下缺点：

首先，“Big-Bang”的集成方式是建立在项目开发人员工作彼此没有交集的情况下进行的，这种现象在早期的软件开发过程中是很常见的，但是现在的软件开发过程，随着软件规模的扩大、分工合作的加深，开发人员之间的相互依赖程度越来越高，几乎不存在相互独立的模块。

其次，很多 bug 在项目的早期就有可能存在，但却到最后集成的时候才发现问题，由于项目后期的复杂性，开发人员很可能需要在集成阶段花费大量的时间来寻找 bug 的根源，而且有时即使找到了 bug 的根源，但由于涉及的模块太多，可能需要改动整个架构，这样既浪费开发人员的时间又耽误了项目的进度。

再次，采用“Big-Bang”的集成方式，项目的开发进度和开发现状很难把握，往往是由团队成员自己估计完成的百分比，但这种估计的不准确性会给项目带来很大的风险。

对于一个小的程序片段或者是一个很小的应用程序，整个系统仅由很少的两个或者三个小模块组成，采用“Big-Bang”集成可能是一种比较好的方法。但是在更多的情况下，项目的情况要复杂的多，需要寻找其他更好的集成方式。

2.2.2 Rational 统一过程与“迭代递增”集成模式

现代工程的记录表明：开发人员需要将 50%的时间花费在系统的调试上，而 39%的错误是模块之间的接口错误。如果容易确定错误位置，就能在软件开发过程中最大限度地提高调试效率。“Big-Bang”集成模式强调使用一站式来完成整个系统的集成，很难发现系统模块错误的根源，为了解决这个问题，现代的软件开发过程通常采用“迭代递增”的集成方式，其中以 Rational 公司的 Rational 统一过程[17]（Rational Unified Process，后面简称 RUP）最为突出。

RUP 认为，当今软件系统非常复杂，使用连续的开发方法（如瀑布模型）是不可能的，需要一种能够通过一系列细化、若干个渐进的反复过程而生成有效解决方案的迭代方法。在 RUP 方法中，通过迭代将整个项目的开发目标划分成为一些更易于完成和达到的阶段性小目标，这些小目标都有一个定义明确的阶段性评估标准。迭代就是为了完成一定的阶段性目标而所从事的一系列开发活动，在每个迭代开始前都要根据项目当前的状态和所要达到的阶段性目标制定迭代计划，整个迭代过程包含了分析、设计、编码和测试等各种类型的开发活动，迭代完成之后需要对迭代完成的结果进行评估，并以此为依据来制定下一次迭代的目标，如图 2-3 所示。

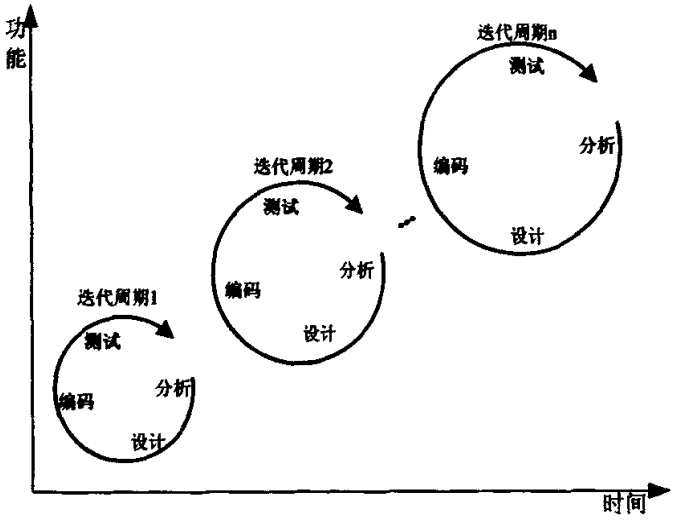


图 2-3 RUP 迭代递增过程模型

采用迭代的 RUP 方法中，其集成模式分为以下三个步骤[18]：

(1) 开发系统中一个小的功能模块。它可以是最小的功能块，但应该是系统一个关键部分。彻底地检查，调试这部分，然后将该模块作为一个核心，在它的基础上设计系统的其它部分模块；

(2) 设计、编码、检查和调试程序；

(3) 将这些新程序代码集成在已经彻底测试的核心模块上。检查、调试核心模块和这些新程序的组合，在加入新程序之前，一定要确保组合工作正确，如果其余工作已被完成，重复过程从第二步开始。

可以看出整个集成过程是一个递增的过程。项目的每一次集成，就向系统的最终目标靠近一步，Steve McConnell 将这个过程形象的比如为好像雪球从山上滚下时不断增大一样，通常把这种集成方法称之为“迭代递增”集成。

相比“Big-Bang”集成模式，采用“迭代递增”集成，可以非常容易确定错误位置。当出现新问题时，错误一定出在新程序功能模块上，无论是新程序与其余程序接口包含的错误，还是新程序和以前被集成的程序相互作用产生的错误，都能准确地知道应该检查哪里。更进一步，由于一次只出现少量问题，将能减少由于多个问题的相互作用，或一个问题掩盖另一个问题的危险。在接口产生的问题越多，“迭代递增”集成方式的优越性越显著。既然在许多工程中，开发者将50%的时间花费在调试上，容易确定错误位置带来的益处，在性能和生产中都将最大限度地提高调试效率。

尽管“迭代递增”集成方式有很多优点，但是在软件开发过程中实际实践时，也有很多问题，其中最为典型的就是“迭而不增”与“增而不迭”两个误区，如图 2-4 所示。

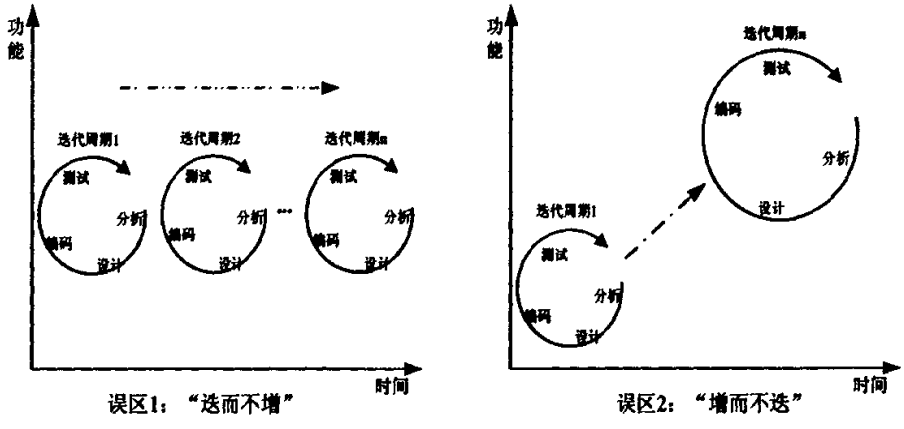


图 2-4 “迭代递增”集成模式的两个误区

由于“迭代递增”集成模式并没有量化每次迭代之间的时间间隔，更不可能量化每次增加任务的工作量，因而经常会造成两次迭代版本之间并没有新功能的

增加或者很长时间不进行新功能模块集成测试这两个问题。“迭而不增”模式只是毫无效率的简单重复，而“增而不迭”又回归到线形模型（如瀑布模型）中的集成方式。微软过程模型中的“每日构建”集成模式可以看着是这两个问题的一种解决方案。

2.2.3 微软过程模型与“每日构建”集成模式

微软过程模型[19]（Microsoft Solution Framework，简称 MSF）是由微软公司根据自身实践经验为企业设计的一套有关软件开发的准则。作为世界上最大的软件公司之一，微软公司三十多年来成功的软件开发史表明了微软过程模型的可实践性与有效性。MSF 过程模型是一种基于阶段的，由里程碑驱动的，递进的软件开发模型。在“生命周期”方面，微软过程分为构想、计划、开发、稳定和发布五个阶段，每个阶段均涉及产品管理、程序管理、开发、测试、发布各角色及其活动，各阶段结束于一个主要里程碑，同时采用递进的版本发布策略。事实上，整个 MSF 过程模型都是由里程碑来推动和管理的。

MSF 过程模型建议项目组首先创建、测试和提供包含核心功能的产品模块，然后再向其中添加功能，提出后续的版本。这一策略通常被称为递进的版本发布策略，每个版本代表一个开发阶段，这种做法更有利于保证产品的质量，也更便于在开发过程中对项目目标进行调整。和 RUP 所强调的不断重复产品的生命周期，以递进的方式推出版本的要求相似，MSF 也要求项目组在开发过程中迅速完成每一次递进过程，并在每一个开发周期中都能切实地增加产品的特性，提高产品质量。

在 MSF 中，最为著名就是它的集成方式——每日构建（Daily Build）。Jim McCarthy[20]认为：在微软的软件开发过程中，如果有什么成功秘诀的话，那就是采用了每日构建的集成方式。尽可能频繁地编译、生成可执行程序，并对每一个生成的程序版本（Build）进行快速测试，以确保产品的每一次改动和每一次检入（Check-in）都不会破坏产品的整体架构，这可以在最大程度上保证整个产品开发过程可管理、可预期，并能增强产品的稳定性。这种每日构建的制度被广泛地应用于微软各种规模的软件开发项目组。

由于对许多大型项目来说，每次构建花掉的时间可能高达几个小时，在白天进行构建可能会消耗过多的计算机资源，对开发造成一定的影响，所以许多大型项目的每日构建是在夜间无人工作或者人比较少的时候进行的。因而很多时候每日构建也被称之为每晚构建（Nightly Build）。

每日构建的过程包括以下步骤[21]：

(1) 定期（如每晚）检查源代码库；

(2) 如果源代码库中发生改变将触发构建和后续的测试；

(3) 给源代码库的当前构建作一个标签，以便将来可以在当前构建的基础上进行重新构建，具体作用参见第三章；

(4) 给相应开发人员发送报告构建情况反馈。

可以看出每日构建将 Big-Bang 集成模式中只有一次的集成工作引入到软件的开发编码过程中，从而使得原先如同噩梦般的集成变成了一件简单的工作。针对“迭而不增”与“增而不迭”两个问题，每日构建将每次迭代之间的间隔时间量化为每天，而每天的工作量则作为迭代之间的增量。这样不仅拥有了“迭代递增”集成模式所有优点，还有效的解决了“迭代递增”集成中容易出现的问题。现代敏捷软件开发方法基于“迭代递增”思想，在每日构建的基础上更进一步，认为在软件开发过程中集成的频率应该更频繁。

2.2.4 现代敏捷软件开发与持续集成

随着技术的迅速发展和经济的全球化，软件开发出现了新的特点，即在需求和技术不断变化的情况下实现快速的软件开发，在此情况下，出现了一些新的开发方法，如敏捷软件开发方法[22]（Agile Methodologies，以下简称敏捷方法）。

敏捷开发以适应性的过程代替传统的预测性的过程，在很大程度上满足了现代商业软件业务复杂、需求多变、时间要求紧迫等特点。敏捷方法以人为核心，它认为开发团队个体的创造力是对付当今越来越复杂的软件开发和频繁变动的唯一办法。针对软件开发过程中的变动，敏捷方法的策略是在整个项目过程中减少变化的成本，它有如下一些实践：1）在 1-3 周内发布第一个版本，以获得快速的反馈；2）采用简单的解决方案，将来更容易修改；3）不断改进设计，提高设计的质量；4）不断进行测试，以便减少后期测试和修改的代价；5）用户参与软件的全过程，并在其中起重要的作用。敏捷方法出现以来，在越来越多的软件开发项目中都获得成功，被证明是一种行之有效的管理思想和实践方法。

尽管“迭代递增”集成模式解决了很多 Big-Bang 模式中的集成问题，有很多优点，但正如在 2.3.2 节分析的那样，在实际项目实践时经常会出现“迭而不增”及“增而不迭”两种错误，必须通过具体措施来避免。MSF 过程模型中的每日构建是一种解决方式，但作为敏捷软件开发方法的代表，XP 过程将此集成方式更进一步，它认为如果一项实践是好的话，就应该将该项实践用到极致。XP 观点认为：

(1) 如果迭代周期短些好，那么我们应该将使迭代时间非常非常短——小时或天，而不是几周或几月。

(2) 如果集成和测试很重要，那么我们应该在一天内多次集成并测试。

于是小版本迭代与持续集成作为两个最佳实践出现在 XP 的 12 个实践里面。XP 社群的观点认为：每天可以构建多次，而不只一次，每日构建只是最低要求，

一个完全自动化的过程应该让项目每天完成多次构建，这是可以做到的。在工作几个小时的开发后，就要对刚才工作的代码进行集成和测试，并快速获得反馈。从敏捷方法的特点不难看出，敏捷方法非常重视反馈。

持续集成的过程与“每日构建”很相似，但还是有很多区别，归纳起来主要有以下几点：

(1) 持续集成强调集成频率。和每日构建相比，持续集成显得更加频繁，目前推荐的最佳实践是每小时就集成一次，但依项目具体情况而定。

(2) 持续集成强调及时反馈。每日构建的目的是得到一个可以使用的稳定的发布版本，而持续集成强调的是集成失败之后向开发人员提供快速的反馈，当然成功构建的结果也是得到稳定的版本。

(3) 每日构建并没有强调开发人员提交（check in）源码的频率，而持续集成鼓励并支持开发人员尽快的提交对源码的修改并得到尽快的反馈。

虽然每日构建和持续集成的本质是相同的，但是它们在集成的频率与对反馈的追求方面的差异也导致了一些开发过程及管理上的差异，Grady Booch[23]因此认为持续集成是一个全新事物。笔者认为从某种意义上说，每日构建只是一种半自动化的定时构建，而持续集成才是真正意义上的自动化构建，这一点将在第三章详细说明。随着 XP 社区在近几年的发展，持续集成这个术语就越来越多地出现在原来每日构建出现的位置，而且渐渐为开发团队所认识。

2.3 本章小结

在本章中，我们分析了从 Big-Bang 集成模式到持续集成思想的转变历程。分析了 Big-Bang 集成模式的缺点，并在总结“迭代递增”集成模式与每日构建实践之上，引出现代敏捷开发中的持续集成思想。可以看出持续集成思想是建立在“迭代递增”集成模式之上，并且借鉴了微软 MSF 过程模型中“每日构建”实践。在下一章中将具体讲解持续集成的理论基础与关键技术实现原理，并总结使用持续集成的价值。

第三章 持续集成基础理论与实现原理

本章从持续集成的定义入手，介绍持续集成的基础理论以及其中关键技术实践，重点分析研究持续集成中自动化过程及测试过程实现原理，最后总结在软件开发过程中应用持续集成的价值。

3.1 持续集成基础理论

Martin Fowler 和 Matthew Foemmel 将持续集成定义为：一个全自动化和可重复的构建，包括测试，一天运行多次。它使得每个开发人员可以进行每日集成，这样可以减少很多集成问题。Grady Booch[23]则将持续集成作为全新的事物，他认为：面向对象开发的宏过程就是一种“持续集成”。按照定期的间隔，“持续集成”过程产生可执行的发布版本，每个发布版本都增加一定的功能。通过快速反馈，项目管理人员可以度量进度和质量，因此可以预见、确定风险，并且可以积极地将风险系在一个正在进行的基础上。所以开发人员也可以这么认为：持续集成是一种意识思维和描述，它通过频繁地集成、增量的软件开发以提高软件质量，减少风险。

持续集成最初是由一组开源工具支持的一个过程。从技术层面上来讲，“持续集成”的含义是指开发团队中的每个成员都尽量频繁地把他们所做的工作提交到统一源码库中，构建服务器自动将新提交的代码与原有项目资源快速创建（Build）成一个新的版本，而且一系列测试会自动运行来验证新提交的代码有没有对项目造成任何破坏。如果测试失败，团队成员能够快速得到反馈，并对其提交的代码进行修改；一旦测试通过，那么所有的团队人员将可以获得最新最好的 Build 版本，并在此基础上进行新任务。持续集成包括以下几大要点：

(1) 访问单一源码库，将所有的源代码保存在单一的地点（源码控制系统），让所有开发人员都能从这里获取最新的源代码（以及以前的版本）。

(2) 支持自动化创建脚本，使创建过程完全自动化。

(3) 每次构建包括自测试（self-testing），而且测试完全自动化。

(4) 使所有人能非常容易得到最新的构建版本，并且能够快速反馈当前项目的构建情况。

(5) 提倡开发人员频繁的提交（check in）修改过的代码。

持续集成的关键是完全的自动化，读取源代码、编译、连接、测试，整个创建过程都应该自动完成。对于一次成功的创建，开发人员要求在这个自动化过程中的每一步都不能出错，而最重要的一步是测试，只有最后通过了测试的创建才

是成功的创建。测试形成的结果，包括成功的测试、失败的测试及其失败测试的细节应当通过某种方式快速反馈给相应的开发人员。通过快速反馈，持续集成非常有利于极早发现编码中的问题，改正当前软件中的错误。

分析整个持续集成过程，可以得出其中包括最核心的三个实践：统一源码库、自动化过程和 BVT 测试（生成验证测试，项目组生成了软件的新版本后，立刻对该版本进行的快速测试，也称冒烟测试）。对于统一源码库，现代化的源码控制工具都能实现该功能，但自动化过程和 BVT 测试是其中的难点，后面两节将具体分析这两个实践的实现过程和原理。

3.2 持续集成的自动化过程实现原理

程序员最不习惯的就是做重复性劳动，而且在做重复性劳动时效率并不一定高，经常会受时间因素，外界环境影响等等，造成说不出原因的错误；而机器最为擅长的就是做一些固定的工作，计算机以恒定的方式完成工作，只要输入不变，不会产生不相同的结果。在软件开发过程中，程序员经常面临一些重复性工作，比如构建与测试代码，应该使计算机完成这些工作，而不是人为的每次重复去构建和测试代码。要让计算机重复做一些工作，就要实现自动化，而且自动化能给团队信心，因为自动化的过程精确、恒定和可重复。由于持续集成的过程强调高频度的集成，因此整个整个持续集成过程包含很多重复性的工作，其中包括编译、连接、测试、部署和反馈等等，那就应该使这些过程自动化。

下面以 Java/J2EE 相关技术为背景，来论述持续集成的自动化过程实现原理。

3.2.1 命令行手工构建

在开发 Java/J2EE 项目中，通常会采用图 3-1 的目录结构。其中 src 和 test 目录包含主要的构建输入数据。所有的产品源文件在 src 目录下，测试文件则置于 test 目录，产品代码用到的所有第三方库文件存放在 lib 目录中，比如一些数据库驱动文件。这些目录的内容可以认为是构建产品的输入，通常应该放入版本控制系统中。当构建运行时，将编译所有 Java 源文件，以生成类文件（.class 文件），build 目录中仅存放构建过程中生成的文件。由于构建输入文件已经存放在版本控制系统中，可以恒定地重复生成构建输出，因而没有必要在版本控制系统中存放 build 目录。

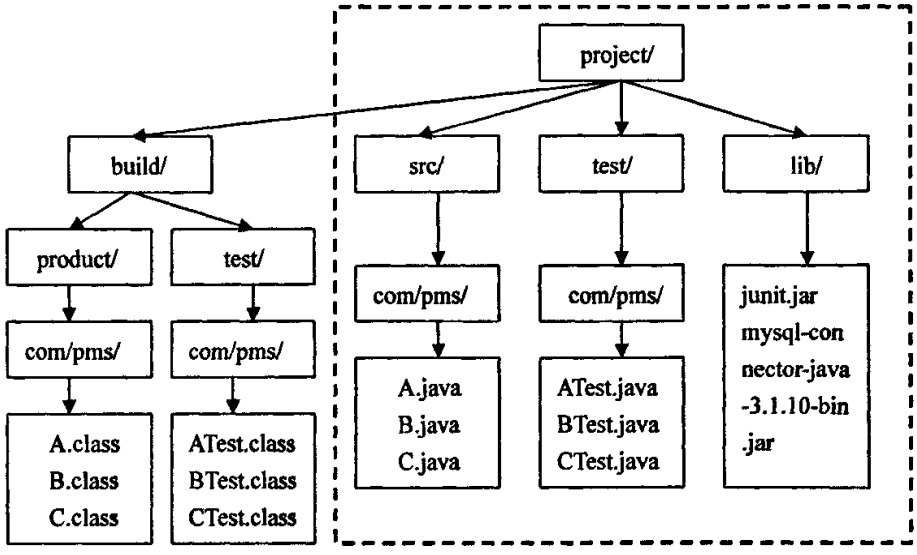


图 3-1 Java/J2EE 目录结构图模拟

对于该产品的构建，通常采用从命令行构建，方式如下：

```
D:\project>mkdir build\product
D:\project>javac -classpath lib\mysql-connector-java-3.1.10-bin.jar;lib\junit.jar -d build\product src\com\pms\*.java
```

这条命令使用了 Java 语言中自带的 javac 编译器来构建产品。其中-classpath 编译器选项列出项目中应用到的所有第三方 JAR 文件，在 Windows 平台下，各 JAR 文件之间用分号隔开，但在 Linux/Unix 系列平台将不相同，因为在这些平台下，分隔符号是冒号，在后面还会提到这种区别。-d 选项告诉编译器，将生成的结果类文件放到 build\product 目录，这个选项后面是编译器需要编译的源文件集合，上面指定的是所有 src 目录下面的源代码。运行该条命令后，将会在 build\product 目录下生成很多 class 文件，如图 3-1 所示。

3.2.2 将命令行脚本化

在项目的开发过程中，可能要运行无数次构建。如果在每次需要构建时都不得不键入这些命令，那么效率会是非常低下，而且很有可能因打错一些字符而不得不检查错误之处，浪费很多时间。可能也有一些项目组，有具体配置人员来做这样的事情，但程序员还是得熟悉命令中这些选项的具体作用。

另一方面，命令行构建过程无法跨平台。正如在上面的构建命令，就无法应用于 Unix 平台。上面只是一个非常简单的构建命令，问题的复杂性不是很明显，但如果构建过程相当复杂时，这个问题将很严重。

为了解决上述两个问题，应当将构建命令放入 bat 批处理文件或 shell 脚本

（分别对应于 Windows 和 Unix 用户）。将多项命令放入一个可执行文件，意味着在键盘敲回车键时，这些命令就可以被固定重复的运行。在 Windows 下面，可以双击该 bat 文件；在 Unix 平台下，也只要输入一条宏命令 sh 即可，而不用重复输入那些复杂且需要记忆的命令用法和选项。这样，任何人就能通过运行其操作系统下的相应文件来完成构建。通常称这种操作过程为“命令自动化”[24]。

对于小项目而言，通过这些脚本文件就可以实现自动化。但是在实际开发过程，项目往往比较复杂，很可能需要将构建过程扩展到包括编译源文件在内的许多步骤，通常需要选用构建工具，对于 Java 项目，Ant 就是可选的构建工具。

3.2.3 利用构建工具

Ant 是一个开放源代码的构建工具，专门用于构建 Java 项目。相比命令行或脚本构建过程，使用构建工具 Ant 有如下好处[25]：

(1) Ant 构建文件是可移植的。当你运行一个构建文件，Ant 会依据当前的操作系统正确格式化 Java 类路径之类的平台依赖性问题，正如前面提到的不能跨平台问题。

(2) Ant 可以跟踪文件之间的关联。Ant 可以只针对有改变的源文件才调用 javac 编译器。这样在运行编译步骤时，可以不必等着所有文件都被重新编译一趟。这对于大型的项目，将节省大量的构建时间。

(3) Ant 包含可以做各种各样事情的任务（task）集。Ant 不仅知道如何编译 Java 源文件，还可以做很多事情，比如运行 JUnit 测试的任务。也可以通过用 Java 编写自定义任务来扩充 Ant 的功能。

对于上面的构建过程，如果采用 Ant，只需要编写一个构建文件，以包含在命令行键入的各个构建步骤。默认情况下，当 Ant 运行时，会在当前目录下寻找名为 build.xml 的文件。上述构建命令在 Ant 中对应如下：

```
<target name="compile" depends="prepare">
    <javac srcdir="${src.dir}" destdir="${build.prod.dir}">
    <classpath refid="project.classpath" />
    </javac>
</target>
```

运行构建非常容易，只需要切换到 build.xml 所在的目录，然后在命令行运行 Ant（在运行 Ant 之前，需要先将 Ant 的安装目录设置到系统的环境变量中）。如果要运行测试，可以在 Ant 中添加新的 test 任务即可，在持续集成的测试过程一节将会详细描述该实现细节。

3.2.4 定时构建

利用 Ant 可以完成构建 Java 项目的绝大多数任务，为实现持续集成提供了大部分的手段，但这个过程准确说还是“交互式”的，是半自动化的，即还是需要手工在命令行执行 Ant 的构建文件，参见文献[25]。持续集成更进一步，利用操作系统的“作业调度”功能，将构建过程完全自动化。例如在 Windows 操作系统下，任务计划程序可用于调度日常作业。At 宏命令将作业放入队列，调度任务将按照指定的时间间隔执行这些作业。(在 Unix 系列平台下，可以使用宏命令 cron 将作业置于队列，就相当于调度作业。）可以为构建文件（build.xml）编写一个批处理 bat 文件，定时执行构建。下面以在 Windows 平台作实例，编写的 build.bat 脚本如下：

```
set CLASSPATH=%CLASSPATH%
cd D:\workingspace
call %ANT_HOME%\bin\ant.bat build.xml
```

为了定时执行这个 build.bat 文件，可以使用 at 宏命令调度文件。测试如下：

```
D:\workingspace>at 22:30 /every:M,T,W,Th,F,S,Su "D:\workingspace\build.bat"
新加了一项作业，其作业 ID = 1
```

执行无参数的 at 命令，可以显示加入的作业计划。在本实例中，我们将构建安排在每天晚上的 10：30 半进行（每晚构建的实现原理）。显示如下：

```
D:\workingspace>at
状态 ID       日期              时间            命令行
-------------------------------------------------------------------------------
 1   每月执行日期:...          22:30           D:\workingspace\build.bat
```

这样就可以在每天晚上执行构建，做到构建过程的完全自动化。同样可以利用 at 实现每隔一段时间就去构建一次，一天构建多次。通常称这种过程为“定时自动化”。

3.2.5 持续集成的自动化过程

利用操作系统的作业调度功能是一种快捷的自动化构建方式，但单纯地将构建过程定时自动化也不能解决太多问题。如果将时间定在每天晚上，那么只有到第二天才能看到构建结果，但很多时候，程序员想立刻看到自己的代码运行结果，而且持续集成也提倡开发人员频繁的提交修改过的代码，因而必须有一种方式支持这种高频度的提交过程的自动化构建。解决问题的方式有两种：

(1) 将两次构建之间的间隔时间尽量缩短；

(2) 在源码控制系统中有文件更新或加入新的文件时，才执行构建。

很明显，第一种方式有一些缺点。因为如果两次构建之间时间很短，很多时候在源码控制系统的代码根本没有改变，只是一种重复构建。第二种方式看起来很可行，但实际上运行时存在问题。因为源码控制系统本身不能执行构建，尽管它可以检测到变化，但要想执行自动化，还必须利用操作系统的作业调度功能。

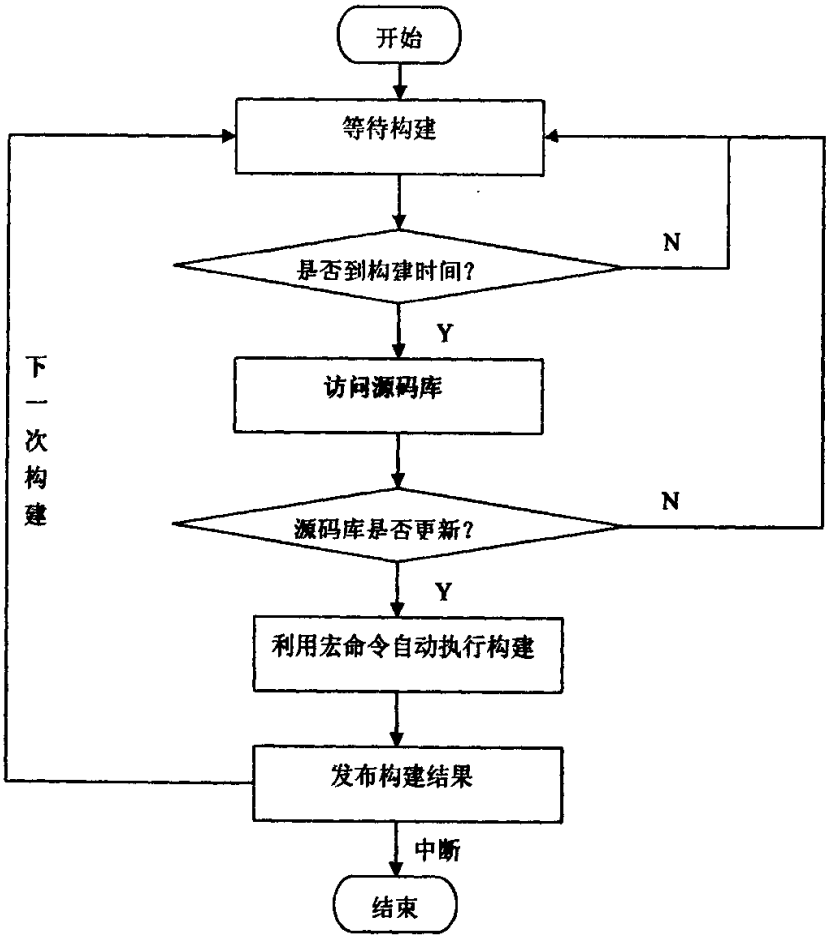


图 3-2 持续集成的自动化过程流程图

持续集成技术综合这两种方式，首先它会尽量使两次构建之间时间间隔短，其次它利用源码控制系统的功能，如果在源码控制系统中的文件没有更新或者没有增加新文件时，不执行构建。因为尽管源码控制系统本身不能执行构建，但它可以检测源码库有没有发生改变，一旦发生改变，将利用操作系统的宏命令执行构建，这样就解决了方式一和方式二中存在的问题。持续集成的自动化构建实现过程如图 3-2 所示。

在持续集成中，自动化过程不仅仅包括编译、连接、测试和部署，还包含反馈。对于编译、连接、测试和部署等构建过程的自动化，可以利用 Ant 工具的扩展，添加 Ant 对应的任务，即可实现（可以参见 3.4 节测试自动化过程，在第五章也会具体演示部署的自动化过程）。反馈的自动化，也是持续集成强调最为重要的一个方面，因为正是基于快速反馈才能及时发现项目中存在的问题。持续集成技术通常会在每次构建完成之后，自动将每次构建结果通过电子邮件的形式发送到团队成员。还有很多支持持续集成的工具在“作业调度”的基础上，进一步完善反馈的功能，把构建的结果以 HTML 格式发布，每次构建结束之后可以在浏览器中以 web 方式查看构建结果，CruiseControl 就是其中优秀的一种，在第四章将详细介绍 CruiseControl，第五章也会具体实践多种自动化反馈方案。

3.3 持续集成的测试过程实现原理

在持续集成中，另外一个重要方面就是 BVT 测试过程。代码通过编译和连接并不能说明集成是成功的，因为编译器只能够检查出程序中存在的语法错误等问题，但对于程序逻辑上的错误却无能为力。因此，一次成功的集成不仅要求代码顺利通过编译和连接，而且还需要通过严格的 BVT 测试。

同样以 Java/J2EE 技术为例，分析持续集成的测试过程。

3.3.1 一般测试过程

Calculator 类实现两 double 型数据的相加，其实现方法如下：

```
public class Calculator{
    public double add (double number1, double number2){
        return number1+number2;
    }
}
```

Calculator 类的 add(double, double)方法的作用为：接受两个 double 值并以 double 类型返回它们的和。实例中没有选择业务逻辑比较复杂的类，是为了专注说明一般的单元测试过程，而不想被实例类业务逻辑本身弄混淆。为了验证该程序的正确性，开发人员经常会采用如 TestCalculator 类中用判断语句的方法来测试 Calculator 类。TestCalculator 类如下：

```
public class TestCalculator{
    public static void main(String[] args){
        Calculator calculator = new Calculator();
        double result = calculator.add(10,20);
```

```
            if(result != 30){
                System.out.println("Error result:" + result);
            }
        }
    }
```

很明显，在 TestCalculator 类中并没有编写命令行提示用户输入数字并计算相加的功能，因为这远远超出了 add 方法要做的事。我们只是选取了一个“工作单元”，10 与 20 的相加是否等于 30，来验证是否 add 方法确实把两个 double 值相加并返回正确的结果，这就是单元测试用例[26]。TestCalculator 类很简单，我们利用了 System.out.println 命令，如果错误，则可以在命令行显示错误信息。但是采用 TestCalculator 这种测试方法存在很多问题：

(1) 如果修改了代码，使得测试失败，我们必须依赖命令行，因为只有在命令行才能找出错误信息。

(2) 如果需要在 Calculator 类中增加新的功能，比如 subtract 或者 multiply 方法，扩展 TestCalculator 类增加测试非常困难。

(3) 由于 Calculator 类不复杂，编写 TestCalculator 类不是很难。但是一旦测试变得复杂，创建、编写和维护测试就变得非常困难。

3.3.2 利用测试工具

为了不过于依赖命令行和让增加和维护测试变得简单，Erich Gamma 和 Kent Beck 在 1997 年为 Java 语言创建了一个简单但非常有效的单元测试框架 JUnit。JUnit 团队在设计框架时，设立了 3 个具体目标[27]：

(1) 框架必须可以帮助编写有用的测试。

(2) 框架必须能够运行和保持过去有用的测试。

(3) 框架必须通过复用代码减少编写测试的成本。

利用 JUnit 框架，编写上面的 Calculator 类的测试如下。

```
import junit.framework.TestCase;                                    ①
public class TestCalculator extends TestCase{
    public void testAdd (){                                         ②
        Calculator calculator = new Calculator ();                  ③
        double result = calculator. add(10,20);                     ④
        assertEquals(30, result, 0);                                ⑤
    }
}
```

①如果要使用 JUnit 框架，必须扩展 junit.framework.TestCase，这个基类中

包含了 JUnit 自动运行测试所需的框架代码。

②在 JUnit 中，命名时应该遵守 testXXX()这样的模式。遵守这样的模式，JUnit 框架就会知道这是个测试方法。

③创建 Calculator 类的实例，开始测试工作。

④和前面的测试一样，调用测试方法并传递 2 个值来执行测试。

⑤JUnit 工作的地方。为了检查结果，调用了 JUnit 的 assertEquals 方法。这个方法由 TestCase 继承而来，用于判断期望的值和实际的值是否相同。assertEquals 的 JavaDoc 为：

```
Static public void assertEquals(double expected,double actual, double delta)
```

在 TestCalculator 单元测试用例中，expected=30，actual=result，delta=0。因为传递给 add 方法的值是 10 和 20，所以期待的结果是 30，后面的 delta 表示误差。（在大多数情况下，delta 参数可以为 0，可以安全忽略，如在本测试用例中。但是在执行不一定会精确的计算时，delta 提供了一个误差范围。只要 actual 值在（expected-delta）和（expected+delta）范围内，测试就算通过了。）

在 TestCalculator 类，只有一个单元测试 TestCase 对象，但当需要一次执行多个 TestCase 对象的时候，就要创建 TestSuite 对象，同样为了执行 TestSuite，需要使用 TestRunner 容器。它们之间的关系，如下图 3-3 所示：

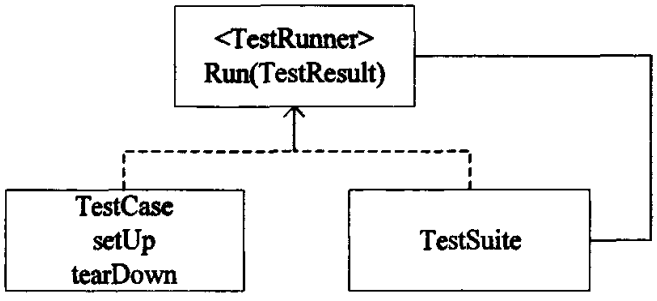


图 3-3 JUnit 工作原理

本文不会讲解关于 JUnit 使用的过多细节，更多关于单元测试及 JUnit 知识可以查看参考文献[26][27]。在第五章项目实践中，测试工具也是选用 JUnit 框架。

3.3.3 将测试自动化

从 3.3.3 节可以看出，基于 Ant 构建工具可以很好地实现对构建任务的扩展。Ant 集成了 JUnit 框架，可以将测试组件作为构建的一部分，如果将构建过程做到自动化，那么测试也就可以完全自动化。在 Ant 中添加单元测试任务如下：

```
<target name="test" depends="compile">
```

```
        <junit printsummary="yes" haltonfailure="no" failureproperty="tests.failed">
            <fileset dir="${src.test}">
                <include name="**/*Test*.java" />
            </fileset>
        </junit>
    </target>
```

上述脚本中我们配置了 Ant 的 JUnit 工作任务，这个任务可以运行一个或者多个 JUnit 测试用例（实例中为${src.test}目录下面的所有测试文件）。设置好配置文件，就可以用 Ant 来一次执行整个项目的测试用例，而且执行过程也很简单，只需在工程目录下，运行 ant test 即可，其中 test 为该任务的名称。下面是运行 ant 后，输出的结果：

```
D:\workingspace>ant test
... ...
test:
[junit] Running xsc.TestCalculator
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.021 sec
BUILD SUCCESSFUL
Total time: 5 seconds
```

3.3.4 持续集成的测试过程

在持续集成里面构建过程已经不只是传统的编译、连接、组成可执行程序那么简单，还应包括对应用程序的测试。测试的代码是开发人员提交源码的时候同时提交的，是针对源码的单元测试，如前文中 TestCalculator 类。将所有的测试代码整合到一起形成测试集，在所有的最新源码通过编译和连接之后还必须通过这个测试集的测试才算是成功的构建版本。这种测试的主要目的是为了验证构建版本的有效性，McConnell[28]称之为“冒烟测试”。通过一个相对简单的检查，看一看这个程序是否能够在运行时冒烟，也就是是否能够体现出正常运行程序能体现的一些正常表象。

图 3-4 展示了持续集成过程中测试的流程。如果一次构建通过了冒烟测试，则可认为这次构建的结果是足够稳定的，那么便可以进行更进一步彻底的测试，通常是由质量保证人员 QA 来处理。如果没有通过冒烟测试，则应该丢弃当前的构建版本，根据反馈的结果迅速修改源码中存在的问题。在不稳定的版本上进行测试，只会浪费更多的时间。冒烟测试应该随着项目的进展而进展，在最开始，可能只进行一些简单测试，如：系统是否可以表示一些简单的字符输出，但随着系统的开发，冒烟测试会变得更加周详，第一个测试可能只需要几秒钟的运行，

但到项目快结束时，冒烟测试的时间可能增加到 30 分钟，一个小时甚至更多。

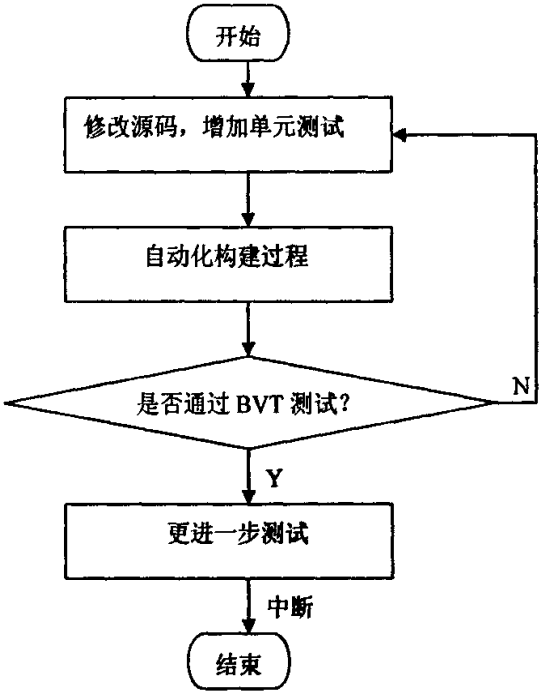


图 3-4 持续集成的测试过程流程图

可以这么认为：没有冒烟测试的持续集成没有一点价值，持续集成正是利用冒烟测试来保证软件产品的质量，同时避免集成问题。在第五章，还会以通过具体项目实践冒烟测试的过程。

3.4 持续集成的价值

从上面对持续集成基本理论的了解，以及第二章对持续集成思想来源的深入分析，根据实际应用经验，笔者总结持续集成的应用价值如下：

1．最小化了集成过程的风险

一个项目团队面对的一个最大的风险就是，当项目中不同成员将他们分别开发的代码结合或集成在一起，得到的结果经常是合并在一起的代码不能很好的工作。为了解决代码中不兼容的情况，调试这些程序需要花费大量的时间。由于很多 bug 在项目的早期就存在，到最后集成的时候才发现问题，开发人员需要在集成阶段花费大量的时间来寻找 bug 的根源，加上软件的复杂性，问题的根源很难定位，可能会花费很多时间来开会讨论 bug 的问题所在（bug meeting），而往往是得不到答案。持续集成采用频繁的集成和及时的反馈就促使项目小组人员积极的面对问题，在 bug 出现的那一天或者那个时刻不久就能发现，并且让 bug 远离系统，而不是将问题推到最后来解决，最终造成隐患。

2. 提高了软件项目质量

与不成功的或者有问题的集成风险有关的是软件开发的低质量风险。通过每天对所有代码进行最小冒烟集成测试，可以防止项目控制中的质量问题。

3. 加强了团队之间的沟通

软件开发本身就是一项强调沟通和协作的活动，持续集成以制度的形式加强团队成员之间沟通和协作，因为每一次的构建都会涉及到团队中的所有成员。每一次构建的成功，是整个团队的成功；而每一次构建的失败，也将是整个团队的失败。因而所有成员的最终目标都是朝着每次的成功构建努力。

4. 明确了项目开发进展的状态

正如在传统开发过程中，项目的开发进度非常难以控制，而持续集成的一个重要作用是在项目开发过程中引入了持续集成制度，使项目随时具有一个明确的“当前状态”。项目团队中的成员的所有工作都是建立在该状态之上的：程序员基于该状态编写代码；测试人员针对该状态进行软件测试。更为重要的是项目管理人员可以根据项目的最新状态对项目的进度、风险、资源使用情况等进行有效的评估，为最终的项目成功打下坚实的基础。

5. 增强了项目开发人员的信心

由于项目的开发人员拿到的代码都是经过测试的，最新的代码，能够保证至少目前没有发现 bug。而且持续集成对项目从一开始，就要求很好的规划，项目的进展是可控制的。项目开发人员能够感受得到实实在在的进步，可以大大增强小组开发人员对于项目开发的信心。

6. 可以改善客户关系

如果说持续集成对项目开发人员是正面的影响，那么他对客户的影响也是正面的。客户同样关心项目的进展情况，如果每天都能让他们感受到项目的进展，他们在项目后面的进展中，也将会变得非常乐于参与。

持续集成的价值远非上面所列举的这些，例如持续集成的过程还可以是一个很好的整合平台。在软件项目开发过程中，很多优秀的实践都可以结合到持续集成的过程中，如测试驱动开发、重构等。尽管实现持续集成可能需要一定的代价，但是对于持续集成所产生的效果，相信这些花费的代价也是值得的。

3.5 本章小结

本章详细介绍了持续集成的基础理论，从中得出持续集成包含三大基础实践，统一代码源、自动化过程及冒烟测试。基于 Java/J2EE 技术，深入分析了持续集成中关键技术自动化过程与测试过程的实现原理，并且总结了在软件开发过程中应用持续集成的价值。

第四章 持续集成插件开发实现及其在项目管理中应用

持续集成的应用可以给软件开发过程带来巨大的价值，但目前还缺乏支持持续集成整个过程的解决方案。本章将根据持续集成的实现原理，开发实现持续集成插件，并将其应用到项目管理中，提供该问题的一种解决方案。

4.1 持续集成插件设计目标

在设计开发持续集成插件时，最主要考虑以下几个方面：

1．有机整合持续集成基础实践

持续集成的过程包含过程自动化、BVT 测试以及统一代码源等多个基础实践，尽管对于每一个实践都有工具支持，但怎样将这些实践有机地组合成一个整体，还存在很多问题。统一整合这些实践是持续集成插件需要解决的主要问题。

2．过程自动化

持续集成能够支持高频率提交、高频率构建、高频率测试及高频率反馈，核心基础就是过程的自动化。保证这些过程的自动化执行也是持续集成插件需要解决的一个问题。

3．持续反馈

自动化的目的是为了获取快速的反馈，进而快速改进软件开发过程中出现的错误，可以认为持续集成是一种基于反馈驱动的过程。持续集成插件应该能够提供多种手段来反馈项目中出现的问题。

4．提供统一的 Web 展示

任何一个项目的成功都有一个共同的特点，那就是能够使整个团队之间有效的沟通与交流。插件应该为项目信息提供统一的 Web 展示，使整个项目团队都知道项目构建状态，明确当前项目的进展情况。

4.2 持续集成插件基础支撑工具

持续集成的过程包含多个基础实践，而对于这些实践的实现需要工具的支持，在设计持续集成插件时选用了一组开源工具。整个插件建立在问题跟踪和项目管理工具 JIRA 之上，构建过程中应用开源工具 CruiseControl，对于统一代码源的实现选择了版本控制系统 Subversion。

4.2.1 插件基础框架 JIRA

JIRA[29]是一个面向问题跟踪和项目管理的应用软件，它的初衷就是为了解

决项目管理中出现的问题。JIRA 从基础上构建，非常注重可用性和灵活性，通过其简洁易用的 Web 交互式充分满足了技术用户和非技术用户的需求。JIRA 支持工作流定制，方便与其他系统集成（包括 Email、RSS、Excel、XML 和源码控制工具）；并且可以在几乎所有硬件、操作系统和数据库平台下运行。JIRA 现已成为很多开源产品的任务及缺陷跟踪工具，其中包括大型开源组织 Apache 等，并且获得了第 17 界 Jolt 生产力大奖。开发持续集成插件基于 JIRA，主要是由于以下几点：

1．Web 的插件机制

JIRA 的出现开创了 web 应用程序开发的一种新的模式，它提出了 Web 应用程序也能基于插件开发的理念[30]。JIRA 提供 Web 的插件开发机制，拥有良好的扩展性，它提供了完整的应用程序接口 API，可以通过编写代码直接与 JIRA 基础平台连接，从而无限制的扩展 JIRA。JIRA 的创造者 Atlassian 公司为插件开发者提供了丰富的资源，包括开发文档、教程、插件开发工具箱以及 JIRA API 使用说明等。JIRA 的插件机制是持续集成插件实现的基础。

2．支持几乎所有的源码控制系统

持续集成中一个很重要的实践就是统一代码源，插件的一个核心任务是要能够跟踪源代码的变更情况，从而快速分析导致构建失败的问题所在。在开发持续集成插件中应用了 Subversion 工具，JIRA 非常完善支持几乎所有的源码控制系统，而且可以工作相当好。

3．出色的管理流程

JIRA 能够管理任务、测试用例、缺陷、改进或者其他任何项目开发过程中问题，这一点非常吻合持续集成的过程。在项目经理安排每次迭代内容及项目成员任务，项目成员执行任务，在每一个任务完成之后，迅速提交到源码仓库中，执行 BVT 测试，获得项目缺陷反馈，从而进行改进。这种“修改—反馈—再修改—再反馈”的思想正是持续集成中所强调那种“小步快进”，而 JIRA 能够对这一过程进行管理，非常容易将持续集成插件应用于项目管理中。下图 4-1 为 JIRA 记录项目任务完成情况的分布图。

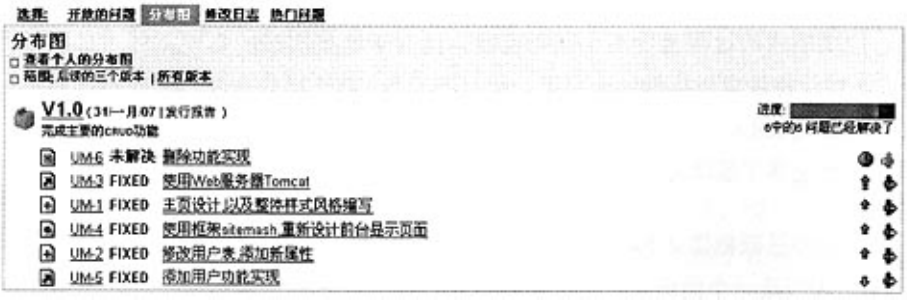


图 4-1 JIRA 任务管理

4.2.2 构建引擎 CruiseControl

持续集成过程中自动化的实现，在插件中是通过使用构建工具 CruiseControl 来完成。CruiseControl[31]是一个支持自动化构建的工具，它是由 Matrin Fowler 所在的公司 ThoughtWorks 开发的用于支持持续集成过程开放源码框架。通过第三章对持续集成的自动化过程的了解，要想支持自动化构建，都是通过调用系统宏命令来实现的，CruiseControl 的工作原理也一样。

CruiseControl 由两部分组成：一部分是可独立运行的 Java 应用程序，负责运行构建；另一部分是 Web 应用程序，负责报告构建状态与结果。独立运行的 Java 应用程序内部实现了对系统宏命令的调用，负责整个构建过程，循环执行各构建周期，包含：引导初始化、检测源码变化、集成构建、自动化冒烟测试、发布构建和测试结果等步骤。Web 应用程序是为完善反馈的功能，把构建的结果以在线 Web 形式发布。CruiseControl 的体系结构如下图 4-2 所示：

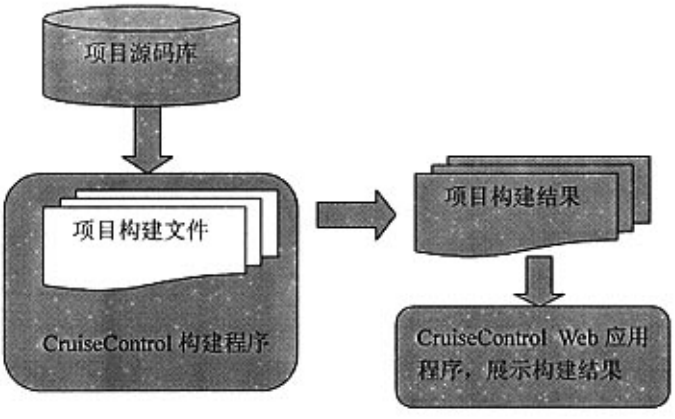


图 4-2 CruiseControl 体系结构

每当项目开发人员在个人开发终端上将改变的代码提交到源码控制系统（如 Subversion），CruiseControl 在随后的构建周期循环中，将会通过源码控制工具的插件检测到这一变化，CruiseControl 会等待一定的间隔时间，以防止新的源码变化出现。一旦创建时间到，CruiseControl 开始调用创建配置文件（如果采用 Ant，一般情况下为 Build.xml）执行构建，它首先更新目标源码目录下的所有内容，以同步变化的源码，再进行编译、连接、形成可执行文件、完成预定的冒烟测试，并将构建结果记录到相应的日志中；CruiseControl 在构建完成后，通过 e-mail 将成功或失败的结果通知提交源码变更的开发人员或者指定的其他人员，并生成构建报告网页，相关人员通过 e-mail 接受通知的同时，也可以通过 Web 网页来浏览构建报告详细信息。使用 CruiseControl 主要是基于以下两点：

1．开源，扩展性强

CruiseControl 改进了作业调度命令的不足，源代码开放且实现简单，可以根据项目需要改进源代码，并且 CruiseControl 支持几乎所有的源码控制系统，Subversion 可以很好的集成到 CruiseControl 框架中来。

2．出色的反馈功能

持续集成的过程非常强调反馈，这点也是我们在设计插件时需要考虑的问题。CruiseControl 拥有强大的反馈功能，首先 CruiseControl 有基于 Web 界面的构建结果，项目相关人员可以在浏览器中直接查看项目构建情况；其次它提供了自动化的邮件发送功能，可以在每次构建完成后，给相关人员发送构建邮件。如果开发人员不在现场，还可以提供 SMS 短消息服务，在第五章将会具体实践持续集成中多种自动反馈方式。

4.2.3 源码控制 Subversion

持续集成插件对于源代码的控制是通过 Subversion 来实现。Subversion[32]是一个开源的版本控制系统，它将文件存放在中心版本库里，这个版本库很像一个普通的文件服务器，但不同的是它可以记录每一次文件和目录的修改情况。因而可以将数据回复到以前的版本，并可以查看数据的更改细节。正因为如此，许多人将版本控制系统当作一种神奇的“时间机器”，即在 Subversion 管理下，文件和目录可以跨越时空。

当前在企业级开发中，应用版本控制系统是一项最基本的实践，而且有很多工具支持，不仅有商业的也有开源的。对于开源的版本控制系统，应用最为广泛的就是 CVS 和 Subversion。在持续集成插件中选用 Subversion 主要是基于以下几点：

1．Subversion 更先进

作为 CVS 的替代品，Subversion 拥有很多 CVS 所不具有的特性，改进了很多 CVS 的不足[33]。CVS 只能跟踪单个文件的变更历史，但 Subversion 实现的“虚拟”版本化文件系统可以跟踪整个目录树的变更。Subversion 能够实现真正意义上的原子提交，也即一系列相关的更改，要么全部提交到版本库，要么一个也不提交。这样用户就可以将相关的更改组成一个逻辑整体，防止出现部分修改而另一部分未为修改的情况提交到版本库中。

2．提供网络访问机制

Subversion 在版本库访问的实现上提供了网络访问机制，可以将 Subversion 作为一个扩展模块嵌入到 Apache 服务器之中。这种方式在稳定性和交互性方面有很大的优势，因为这样可以直接使用服务器的成熟技术，比如认证、授权和传输压缩等。此外，Subversion 自身也实现了一个轻型的，可独立运行的服务器软

件，该软件可以轻松的用 SSH 封装以提高安全性。从 4.4.1 中可以看到，将持续集成插件应用于项目管理平台 PMS 中选用的 Web 服务器正是 Apache，Subversion 源码控制系统和 Apache 服务器之间的无缝集成是插件中选用 Subversion 的一个主要原因。

4.3 持续集成插件开发实现

4.3.1 JIRA 插件体系结构

JIRA 的 web 插件机制是持续集成插件实现的基础，JIRA 框架最成功的地方就是它在设计之初就是基于插件的体系结构，如图 4-3 所示。这个体系中重要的概念是扩展点（extension points），也就是 JIRA 底层应用为插件提供的接口。每一个插件都是在现有的扩展点上开发，并可能还留有自己的扩展点，以便在这个插件上继续开发。针对 JIRA 的开发，可以分为两类人员：插件开发人员和基础部分应用程序开发人员。JIRA 提供了良好的插件开发机制，在主页上也提供了完整的应用开发工具包，其中不仅包括各类的使用，而且包括整个插件开发工程的建立。JIRA 的插件开发非常活跃，并且在 2006 年举办了 Codegeist 插件开发比赛[34]。作为插件开发人员开发插件时首先需要清楚基础部分应用程序提供的功能以及怎样让 JIRA 底层应用发现插件，通常是以 XML 文件描述。这样 JIRA 系统的核心部分在启动的时候要完成的工作十分简单：启动平台的基础部分和查找系统的插件。

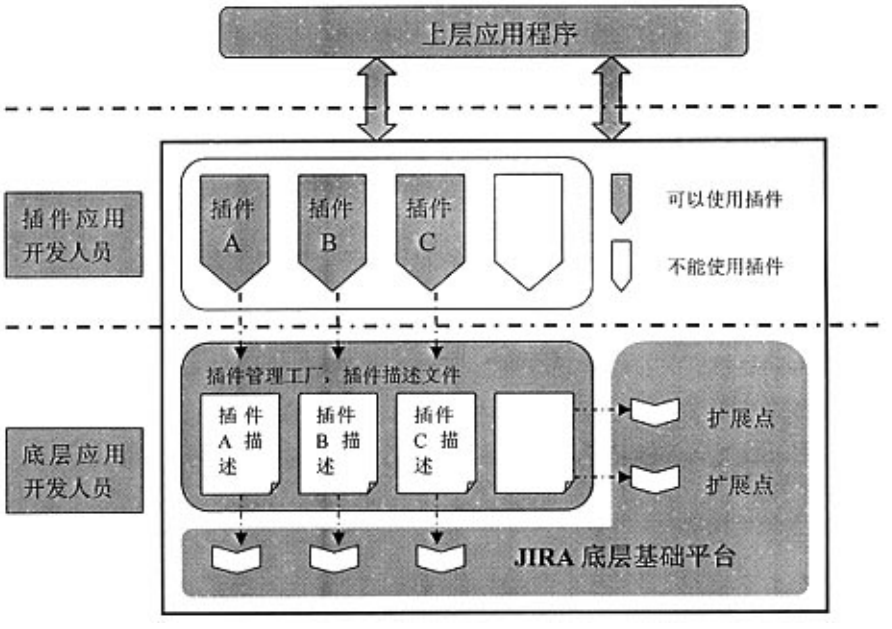


图 4-3 JIRA 插件体系结构

在 JIRA 插件开发中有一个 XML 文件（atlassian-plugin.xml）专门用于管理插件，任何在上层应用中提供功能的插件，即上图中可使用插件，都需要在这里注册。该文件的作用就是用于说明插件的信息，以及插件模块的实现的位置。一个典型的插件描述 XML 文件如下：

```
<atlassian-plugin name="Test Plugin" key="test.atlassian.plugin">
    <plugin-info>
        <description>A test plugin!</description>
        <version>1.0</version>
        <application-version min="3.0" max="3.1"/>
        <vendor name="Atlassian" url="http://www.atlassian.com"/>
    </plugin-info>
     ... individual modules ...
</atlassian-plugin>
```

其中<plugin-info>的作用就是说明插件的一些基本信息，而最核心的部分是各模块（individual modules）具体逻辑的实现。

4.3.2 持续集成插件的开发实现

JIRA 中持续集成插件的开发实现主要包括以下三步骤：扩展点的选择、持续集成业务逻辑实现及插件描述文件编写。

1．扩展点的选择

如图 4-3，在 JIRA 中有很多插件的扩展点。针对持续集成插件设计目标，选择的扩展点为继承 ProjectTabPanel 类。ProjectTabPanel 为每个项目信息展示的面板，通过继承 ProjectTabPanel 类来实现项目的构建测试信息以及源码变更的反馈。ProjectTabPanel 为一接口类，实现如下：

```
public interface ProjectTabPanel {
    void init(ProjectTabPanelModuleDescriptor descriptor);
    String getHtml(Browser browser);
    boolean showPanel(ProjectActionSupport action, GenericValue project);
}
```

持续集成插件中核心业务逻辑类 ContinuousIntegrationProjectTabPanel 实现 ProjectTabPanel 接口，其层次类结构如下图 4-4 所示。

构建引擎工具 CruiseControl 每次构建的信息都是保存在日志 log 文件中，因而 ContinuousIntegrationProjectTabPanel 类需要解析这些 log 文件，获取其中构建结果信息。

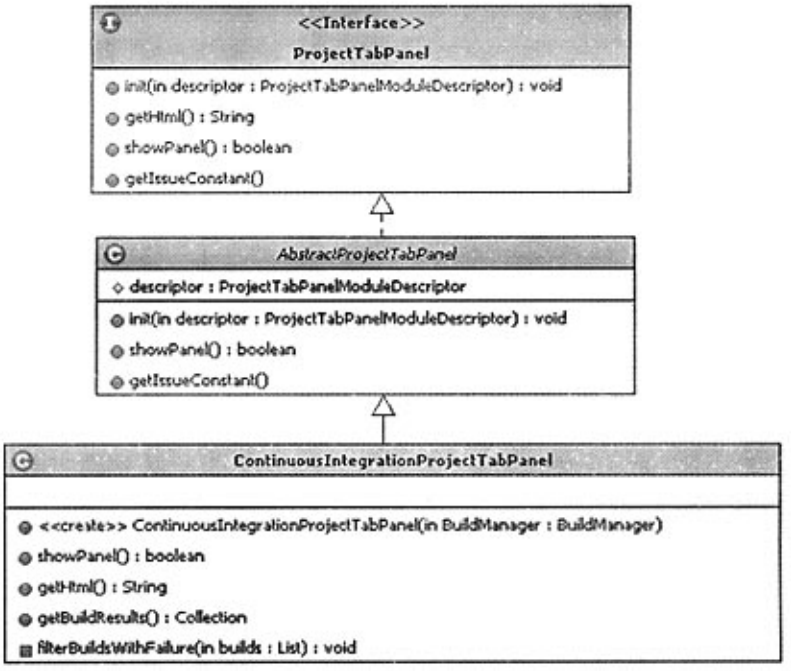


图 4-4 持续集成插件扩展点

2. 插件业务逻辑开发流程

持续集成插件业务逻辑的实现过程如下图 4-5 所示，其中最为核心的过程就是对构建日志 log 文件的解析以及提出的信息发布。

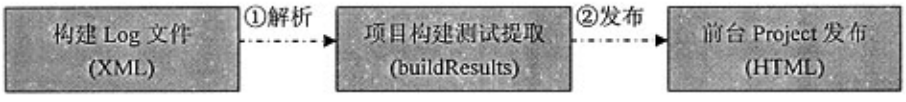


图 4-5 持续集成插件业务逻辑实现过程

对于构件日志 log 文件的解析，我们采用了 W3C 推荐标准的文档对象模型（Document Object Model，DOM[35]）技术来实现。DOM 文档是以层次结构组织的节点或信息片断的集合，这个层次结构允许开发人员在树中导航寻找特定信息；另一方面，DOM 还提供了一个 API，允许开发人员添加、编辑、移动或删除树中任意位置的节点，从而创建一个引用程序。更多关于 DOM 技术的使用，可以参看文献[36]。

图 4-6 为实际 CruiseControl 的构建日志 log 文件一片段代码，下面将介绍怎样基于 DOM 技术解析其中的构建信息。

```
<modifications>
    <modification type="svn">
      <file action="modified">
        <revision>6</revision>
        <filename>/src/org/appfuse/dao/hibernate/UserDAOHibernate.java</filename>
      </file>
      <user>Raymond</user>
      <comment><![CDATA[Correct the error of UserDaoHibernate.java, implement the function of saveUser.]]></comment>
……
    </modification>
</modifications>
<info>
……
<property name="projectname" value="UserManager" />
<property name="builddate" value="01/23/2007 17:32:47" />
<property name="lastbuildsuccessful" value="true" />
……
</info>
```

图 4-6 CruiseControl 构建 log 文件片段

以获取图 4-6 中的构建时间“builddate”信息作为事例，编写解析代码如下：

```
public String parseXmlFile(String xmlFile){
    DocumentBuilderFactory domfac= DocumentBuilderFactory.newInstance();          ①
    … …
    DocumentBuilder dombuilder=domfac.newDocumentBuilder();                       ②
    InputStream is=new FileInputStream(xmlFile);
    Document doc=dombuilder.parse(is);                                            ③
    Element root=doc.getDocumentElement();                                        ④
    NodeList nodes=root.getElementsByTagName("property");                         ⑤
    … …
    if(node.getAttributes().getNamedItem("name").getNodeValue().equals("builddate")) {
            return node.getAttributes().getNamedItem("value").getNodeValue();     ⑥
    }
    … …
}
```

①创建 DocumentBuilderFactory，通过 DocumentBuilderFactory 对象创建 DocumentBuilder。

②创建 DocumentBuilder，通过 DocumentBuilder 执行实际的解析以创建 Document 对象。

③解析文件创建 Document 对象。①②③步为典型的通过工厂模式[37]创建对象的应用。

④获取解析 XML 文件根元素。

⑤获取 XML 文件根元素下面 property 标志。

⑥获取 XML 文件中节点为“builddate”的值。

同样为了获得日志 Log 文件其他信息，比如发生改变文件“filename”节点、成员节点“user”及最后一次构建成功与否“lastbuildsuccessful”节点等都可以采用类似的方式，通过这六步来完成。解析出来的信息以 HashMap 的形式存储在 buildResults 中，通过实现 ProjectTabPanel 接口类中 getHtml 方法完成发布，即图 4-5 中第二步，方法实现如下：

```
public class ContinuousIntegrationProjectTabPanel implements ProjectTabPanel  {
    ... ...
    public String getHtml(Browser browser)  {
        final Map velocityCtx = UtilMisc.toMap("action", browser);
        velocityCtx.put("buildResults", buildManager.getBuildresults());
        return descriptor.getHtml("view", velocityCtx);
    }
    ... ...
}
```

3．插件描述文件编写

要完成一个完整插件的编写，使插件能够应用，还需要最后一步，即编写该插件的描述文件。在 atlassian-plugin.xml 文件编写持续集成插件的描述文件如下：

```
<atlassian-plugin name="ContinuousIntegration Plugin" key="CI.Worksoft.Plugin">
    <plugin-info>
        <description>ContinuousIntegration Plugin</description>
                <version>1.0</version>
                <application-version min="3.0" max="3.1"/>
                <vendor name="worksoft" url="http://www.worksoft.com.cn"/>
     </plugin-info>
     <project-tabpanel key="buildresults" name="ContinuousIntegrationPanel"
class="com.atlassian...ContinuousIntegrationProjectTabPanel">
        <description key="projectpanels.buildresults.description"/>
        <label key="common.concepts.buildresults" />
        <order>50</order>
        <resource type="velocity" name="view" location="templates... buildresults vm" />
        <resource type="i18n" name="i18n" location="com.atlassian... buildresults" />
    </project-tabpanel>
```

</atlassian-plugin>

其中最为重要的是指明业务逻辑实现 ContinuousIntegrationProjectTabPanel 类的相对位置。描述文件中 resource 属性为展示样式，应用了 velocity 模板框架，i18n 为对国际化的支持。

4.3.3 持续集成插件运行流程

持续集成插件一次构建循环过程如下图 4-7 所示：

(1) 项目团队成员提交了新的代码或者更新了源代码，Subversion 源码仓库检测到变化；

(2) 通过利用构建引擎工具 CruiseControl 的定时构建功能，在过一段时间后（通常是以小时为单位），Ant 执行构建；

(3) 在 Ant 构建工具中扩展任务，包括对代码规范进行检测、单元测试以及自动化部署等等；

(4) 将 3 中的结果通过解析快速反馈报告到前台展现中，在构建失败时，提供多种反馈形式。

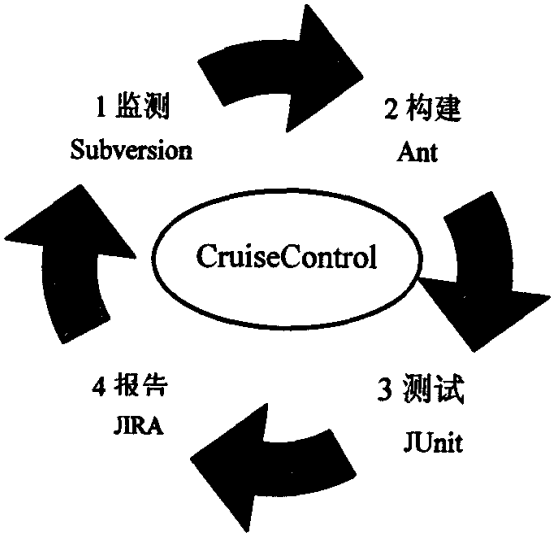


图 4-7 插件一次构建循环过程

可以看出持续集成是一个循环构建的过程，在持续集成插件中，整个构建流程以开源工具 CruiseControl 作为构建引擎，完成对于源码的自动更新、自动构建、自动测试和自动反馈。

4.4 持续集成插件在项目管理中应用

跟踪和管理项目中出现的问题和缺陷是一项至关重要的任务，但是几乎没有

哪一个团队可以有效的做好。持续集成思想的应用对于改进软件开发过程管理，提高软件开发效率有着重要意义。下面给出一种基于上述持续集成插件的项目管理平台 PMS，从而有效管理软件项目的持续集成开发过程。

4.4.1 项目管理平台 PMS 总体架构

项目管理系统 PMS 总体架构如下图 4-8 所示。

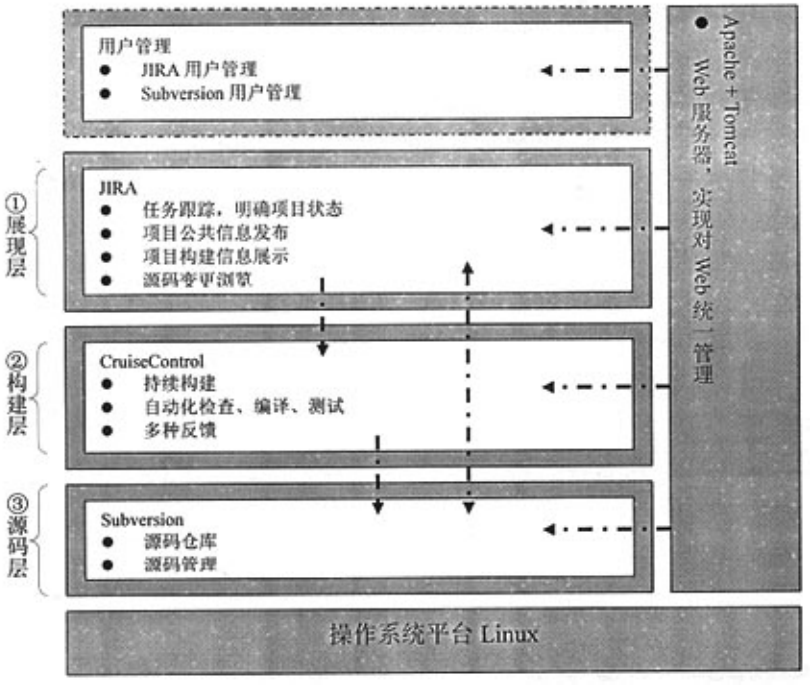


图 4-8 PMS 总体架构

从图 4-8 不难看出，整个 PMS 系统在逻辑上分为三层。其中使用 JIRA 作为前台展现层，完成对任务跟踪管理、公共信息发布、构建状态情况以及源码变化情况的展示；核心构建层使用 CruiseControl 作为每个项目的构建引擎工具，来完成持续集成中自动化过程的实现，关于 CruiseControl 的工作原理可以参看前面 4.2.2 小节；源码管理层使用版本控制工具 Subversion，来完成持续集成中另一项关键实践统一源代码的实现。整个 PMS 平台基于 J2EE web 技术，选用的 web 服务器为 Apache 与 Tomcat 的组合。

将 PMS 平台应用于实际项目应用时，其一种部署方案可以采用如下图 4-9 模式。

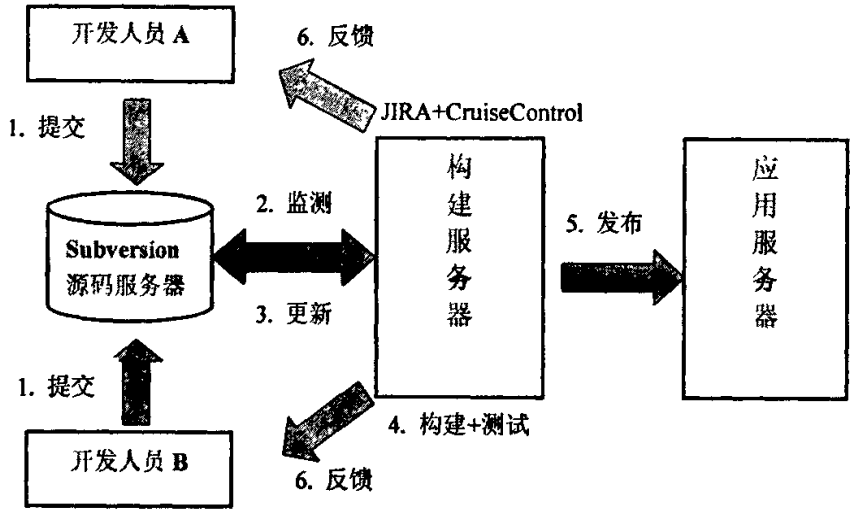


图 4-9 PMS 运行部署

4.4.2 持续集成插件的应用

图 4-9 为持续集成插件在 PMS 平台中运行界面。

选择: 开放的问题 分布图 修改日志 热门问题 持续集成

Time of Build:	Build Result:	Member:	Tasks:	Modifications:	Comments:
01/23/2007 17:32:47	✔	Raymond	UM-5	UserDAOHibernate.java UserDAOTest.java	Correct the error of UserDaoHibernate.java, implement the function of saveUser.
01/23/2007 16:56:17	✔	Raymond	UM-5	UserDAOTest.java	Test the save function of UserDao.
01/23/2007 16:30:28	✔	Eric	UM-4	struts-config.xml	The struts config file is error.
01/23/2007 16:11:03	✔	Eric	UM-4	sitemesh.xml	Use the sitemesh web framework.
01/23/2007 15:44:40	✔	David	UM-3	web.xml	Correct the error of web.xml.
01/23/2007 15:34:14	✔	David	UM-3	web.xml	Change the web server, user Tomcat.
01/23/2007 15:04:46	✔	Raymond	UM-2	User.java	Correct the model, add the attribute.
01/23/2007 14:50:02	✔	Raymond	UM-2	User.hbm.xml	Change the model xml, add a new row.
01/23/2007 14:24:30	✔	David	UM-1	global.css	Change the total style.
01/23/2007 14:04:46	✔	David	UM-1	index.jsp	Add some new pictures, let it looks better.

图 4-10 持续集成插件运行界面

从图中可以非常明显看到当前项目的进展状态，其中最近一次构建发生在“01/23/2007 17:32:47”、构建“成功”、提交成员为“Raymond”、对应任务为“UM-5”（JIRA 中对于每个任务都分配了标识符，参见图 4-1）、改动文件

“UserDAOHibernate.java”以及改动的注释说明是“为了修改上一次构建的错误，实现了 saveUser 功能”。整个信息的来源可以参看图 4-6 日志 log 文件，当然也可以根据实际项目需求，改变插件代码，添加相应的信息。

4.4.3 PMS 总体评价

PMS 平台最大的特点就是基于持续集成思想把 JIRA 的任务管理、CruiseControl 的自动构建以及 Subversion 的源码管理以插件的形式非常良好地整合到一起。PMS 平台封装了持续集成思想的实现细节，对于终端用户来说，只需要从前台 JIRA 展现层获取信息和反馈，根本不用去处理后台 CruiseControl 与 Subversion 的服务。在图 4-10 中，可以看到对于“任务 Tasks”及“源码改动 Modifications”都提供了超链接，点击 Tasks 中属性值，如“UM-5”，即可看到 JIRA 中对该任务的详细描述及状态跟踪；同样，点击 Modifications 中属性值，如“UserDAOHibernate.java”，基于 Subversion 的版本控制功能，可以检查当前代码改动的细节。这样在发生构建失败时，可以非常方便地查找出项目是在哪个功能任务上出现问题以及是由于对代码作哪些改动导致构建失败。跟踪任务进展、获取快速反馈，从而进行快速修改，这即是持续集成思想在项目管理平台 PMS 中应用。

PMS 平台基于开源项目，以插件的形式开发，因而可扩展性强，非常容易定制，可以根据实际项目需求改变前台展现，这也是 PMS 平台另一个特点。

4.5 本章小结

本章详细介绍了持续集成插件开发实现过程，该插件基于 JIRA 框架，并且应用了 CruiseControl 和 Subversion 开源工具。更进一步，本章后面部分分析了该插件在项目管理中的应用，设计了项目管理平台 PMS，提出了一种支持整个持续集成过程的解决方案。该平台是在实际环境中运行稳定，具有良好的可扩展性，可以作为实践持续集成的一种整体解决方案，也可以根据实际企业情况在该方案基础上作适量扩展。

第五章 用户管理系统的持续集成实践

本章将通过用户管理系统 UserManager 这一具体项目来实践持续集成的过程，给出在实际项目中实施持续集成的指导，其中重点为测试过程及反馈过程，实施的整个流程基于 PMS 平台。

5.1 用户管理系统 UserManager 简介

UserManager 项目是一个典型的三层 MVC 模式架构的 Web 应用程序，功能是实现对用户 User 的 CRUD（创建、检索、更新和删除）操作。它集成了目前业内最为主流的轻量级开源框架 Struts、Spring 和 Hibernate 的组合[38]，基于 J2EE 技术实现。UserManager 项目来源于开源项目 Appfuse[39]，其架构如图 5-1 所示：

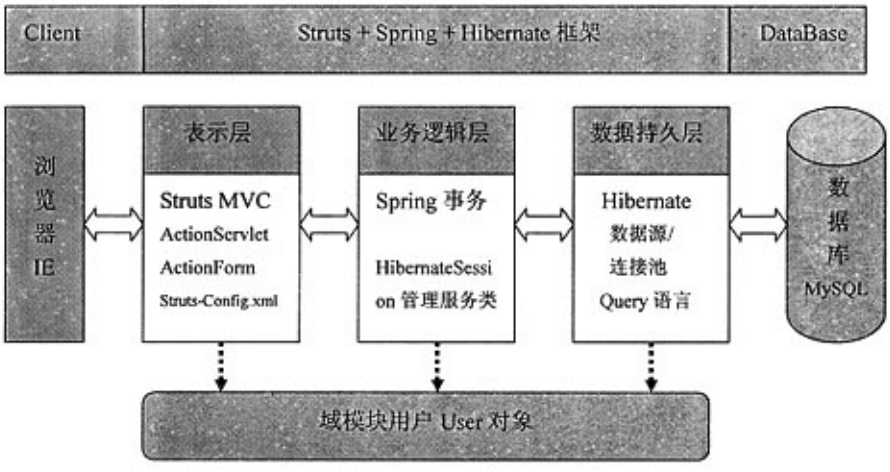


图 5-1 UserManager 项目架构

表示层：由 Struts 实现，以 struts-config.xml 为核心，通用的控制组件 ActionServlet 承担 MVC 中 Controller 的角色，ActionForm 类封装与用户界面的数据元素，用 Action 类实现业务逻辑、动作处理、链接转向。

业务层：由 Spring 实现业务组件的组装关联，通过依赖注入 IoC、AOP 应用以及面向接口编程等技术，来降低业务组件之间的耦合度，增强系统兼容性和可扩展性。

持久层：借助 O/R Mapping 工具 Hibernate 实现数据库访问性能优化和与数据库交互的常用操作（添加、修改、删除、浏览），并将数据库表与对象进行关

联，把利用 SQL 对数据库表的交互转化为直接针对对象的数据库交互，如此大大提高编码效率。

域对象层：域对象是与数据库表关联的对象的集合，是各层之间数据通信的载体，业务的对象化主要是基于业务逻辑复用的考虑。在 UserManager 项目中，操作域对象为用户 User 类。

选用 UserManager 项目，主要是由于以下几点：首先，项目本身不能过于复杂，业务逻辑操作应该非常简单明确；其次，基于 Struts、Spring 和 Hibernate 组合的 MVC 模式架构项目结构清晰[40]；再次，项目应具有代表性，能够涉及到当前企业级开发中出现的各种问题，而基于 Appfuse 框架的项目已广泛应用于各种企业级开发之中。

作为实例项目，本文不会说明 UserManager 具体每一个类的作用，而重在说明持续集成的实施过程。

5.2 UserManager 项目实施持续集成整体模型与实验平台模拟

在对 UserManager 项目实施持续集成之前，首先要说明实施流程整体模型，以及实验平台对于该模型的模拟。

5.2.1 实施整体模型

UserManager 项目实施时，整体过程拟采用如下图 5-2 所示。

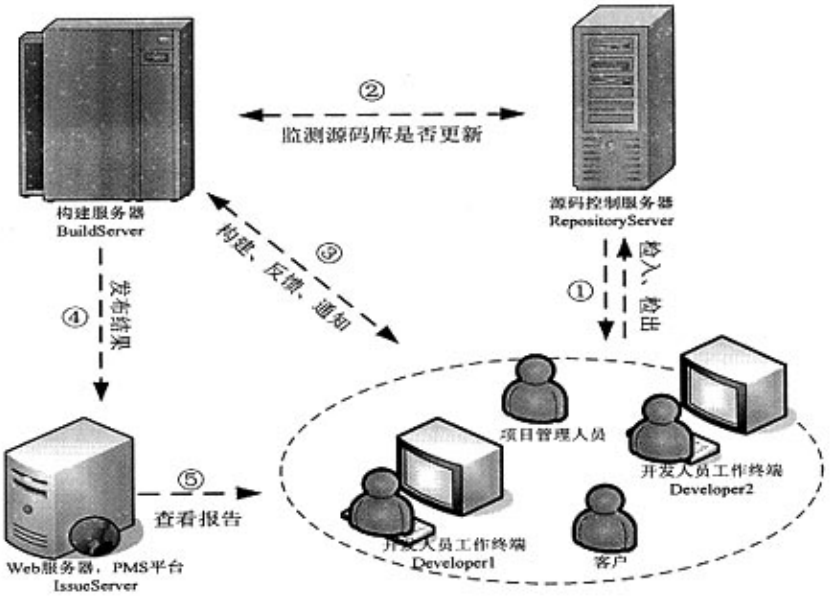


图 5-2 UserManager 项目实施整体模型

从图 5-2 不难看出，该项目的实施是基于 PMS 架构平台，其运行流程借鉴了 PMS 中持续集成思想。

5.2.2 实施平台模拟

图 5-2 给出了真正在项目中实施持续集成的概览，但那样做需要多台服务器、主机和开发人员，笔者在单机上对上述开发平台进行了模拟，单机模拟目录如下图 5-3。本文中 UserManager 项目持续集成的实践过程就是在该模拟平台下实施。

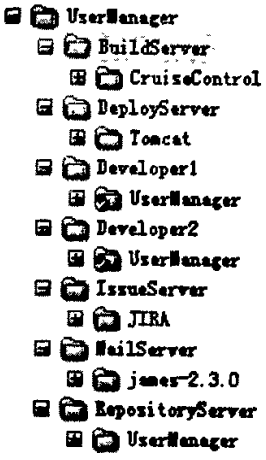


图 5-3 UserManager 实施平台模拟

可以看到，笔者以“虚拟主机”的形式模拟了各种服务器和终端的实现，每个文件夹在实际中都代表一台独立的服务器或主机终端。在 UserManager 项目实施持续集成过程中，这些“虚拟主机”中配置如下：

(1) 源代码管理服务器（RepositoryServer）：持续集成中基础实践统一源代码源的保证。UserManager 项目中选用的源码控制工具为 Subversion，其上面应安装 Subversion 服务器端，负责对项目代码进行统一管理。

(2) 构建服务器（BuildServer）：构建服务器上应安装构建工具 Ant、构建引擎工具 Cruisecontrol 及 Subversion 客户端。

(3) 发布服务器（DeployServer）：由于 UserManager 项目为 J2EE Web 项目，安装选用的 Web 服务器为 Tomcat。Tomcat 能够很好的支持应用程序“热部署”功能，为后面自动化部署过程的实现带来很多方便。

(4) 任务跟踪及缺陷管理服务器（IssueServer）：安装 PMS 平台，用于项目相关信息显示，以及项目人员查看每次构建报告及项目进展。

(5) 邮件服务器（MailServer）：安装邮件服务器，UserManager 项目选用的工具为 Apache James。邮件服务器的模拟是为了实践持续集成中自动化邮件反馈功能的实现，具体见 5.3.6 节。

(6) 开发人员 Developer1 和 Developer2：Developer1 和 Developer2 是模拟现实中的开发人员，其上面应安装 Subversion 客户端，而且应有针对 UserManager 项目独立的工作区间。

持续集成实践最初就是由一组开源工具实现的，UserManager 项目的实践同样涉及到很多开源工具，包括 Subverion、Ant、JUnit 等，在后文反馈手段中还应用了 RSS 阅读器以及 Yahoo Widget 工具，对于这些工具的了解有助于更好地理解持续集成过程。

5.3 UserManager 项目持续集成实施过程

5.3.1 UserManager 源码管理

对于 UserManager 源码管理包括两部分：服务端设置和客户端设置。

1．服务端设置

首先需要服务器端建立源码仓库（Repository），来实现对于源代码变更的存储。在源码控制服务器 RepositoryServer 上安装完 Subversion 之后，将安装文件下 bin 目录拷贝系统变量 path 中（目的是为了方便使用 Subversion 中命令）。在本应用项目中，源码库的路径为：D:\UserManager\RepositoryServer\UserManager。在命令行输入以下代码，即可实现建立源码库，其中 svnadmin 是在 Subversion 中建立仓库使用的命令。

```
D:\UserManager>svnadmin create D:\UserManager\RepositoryServer\UserManager
```

建立源码仓库后，需要将 UserManager 项目的实例代码导入到刚才建立的仓库中进行管理，采用 import 命令，如下：

```
D:\UserManager>svn import . file:///D:/UserManager/RepositoryServer/UserManager -m "initial import"
```

其中“import .”表示导入当前目录，即 D:\UserManager 下面的所有文件；“-m”选项表示注释，即描述当前所作操作完成的功能。在项目的开发过程中，每次对于源码仓库的操作，都应该作注释，表明操作的目的。导入完毕后，可以发现此时“提交后的修订版为 1”，对于 UserManager 源码仓库的操作即完成。上面采用的过程是使用命令行的方式，在现实开发中，也可以选用基于 Subversion 的客户端工具，其中使用最为广泛的是 TortoiseSVN。

2．项目成员客户端同步

创建完 UserManager 源码仓库后，后面就需要项目成员进行同步，每个项目成员在自己的工作机器上都有对应的工作区间（workingspace）。导出源码仓库中 UserManager 的信息，可以使用 Checkout 命令。以其中一开发人员 Developer1 为例，操作如下：

```
D:\UserManager\Developer1\UserManager>svn checkout file:///D:/CIProjectServer/UserMa
nager
.....
A    UserManager\test\org\appfuse\service\UserManagerTest.java
A    UserManager\test\org\appfuse\dao
A    UserManager\test\org\appfuse\dao\UserDAOTest.java
A    UserManager\test\org\appfuse\dao\BaseDAOTestCase.java
A    UserManager\test\org\appfuse\web
A    UserManager\test\org\appfuse\web\UserActionTest.java
.....
取出修订版 1。
```

可以看出，Developer1 取出的版本也为 1，与源码仓库中是同步的。

项目团队中的其他人员也应当作同样的操作，使用 Checkout 命令。需要注意的是，构建服务器 BuildServer 也应作为一客户端，因为每次的构建的操作都是在构建服务器上完成。在构建服务器上，也应该有 UserManager 项目对应的工作目录，需要作同样的操作。与项目团队中工作区间不同的是，构建服务器需要执行的操作只是更新（Update），每次构建前要与源码仓库同步，保证构建的文件是最新的，从后面 CruiseControl 的脚本文件也可以看出这一点。

5.3.2 配置 CruiseControl 管理 UserManager 项目

为了实现对 UserManager 项目的持续集成管理，从图 5-2 中可以看出，使用了持续集成工具 CruiseControl，需要对它作一定的配置。修改 CruiseControl 的配置文件 config.xml，对 UserManager 项目作的配置文件如下：

```
<cruisecontrol>
    <project name="UserManager">
    ... ...
    <modificationset quietperiod="60">
        <svn localWorkingCopy="projects/${project.name}"/>
    </modificationset>
    <schedule interval="300">
    <ant anthome="apache-ant-1.6.5" buildfile="projects/${project.name}/build-cc.xml"/>
     </schedule>
    ... ...
    </project>
</cruisecontrol>
```

其中：<cruisecontrol>为整个配置文件的根目录。它可以包含有一个或者多个<project>属性，上面仅列出了对 UserManager 项目管理的配置文件。

<modificationset>子元素包括了 Subversion 插件的配置信息，用于检测源码仓库中是否有文件更新或新文件的提交。从配置文件可以看出，构建服务器也可以看着是一客户端，其工作区间为 projects/${project.name}，即当前目录 projects

下面。Quietperiod（单位为秒）定义了一个时间值，如果 CruiseControl 检测到变化，会自动等待一段时间后才触发构建过程，等待的时间就是由 Quietperiod 属性值决定，在 UserManager 项目中为 60 秒。设置 Quietperiod 的原因是为了防止在一项目成员提交还没有结束时，另一个项目成员也去提交。CruiseControl 只会在源码仓库“平静”后，才执行创建过程，这样可以有效的避免冲突。

<schedule>指定了创建的时间间隔(UserManager 项目为 300 秒)，即每隔 300 秒检测一次源码库，如果检测到变化，就执行所指定的构建过程。其实现原理可以参看第三章中持续集成的自动化过程。

<ant>属性中 anthome 元素指明了构建工具 Ant 安装目录以及 Ant 构建委托脚本文件，在此项目中为 build-cc.xml。作为委托脚本文件，主要完成两个任务：

(1)在构建之前保证构建服务器工作目录中的文件最新，与源码仓库同步；

(2)指定真正的 Ant 构建文件。

build-cc.xml 实现如下：

```
<target name="update">
        <exec executable="svn"><arg line="update"/></exec>
</target>
<target name="build" depends="update">
        <ant antfile="build.xml" target="all" />
</target>
```

由此可见，CruiseControl 的工作过程，非常好的阐述了在第三章中持续集成自动化原理的工作过程。

前面两部分可以看作是基础设置，其为实施持续集成的过程提供保障。后面将重点说明 UserManager 中持续集成的自动化检查、测试与反馈的过程。

5.3.3 自动化代码规范检查

由于一些个性或者爱好上的问题，程序员之间的编码风格相差很多。因而很多项目在开始时，都制定了编码规范，但效果却不是很好，因为很少有成员真正去执行这些规范。没有代码规范或者应用不够会造成开发过程中的很多问题，必须通过具体措施促使项目成员按照规范编码。一些大型企业使用非常正式的同级评审（peer review），在该评审过程中，开发人员要为代码作出同级评价，并提供改进意见；还有一些企业使用 XP 中倡导的“结对编程”实践，同时经常交换对象，使项目成员彼此影响。但企业采用的这些措施，都给开发人员增加了很大的工作量，在原本时间就很紧的企业开发中效果通常都不是很明显，又由于这些方法几乎都是通过手工的方式，不仅效率很低，而且很容易出错。实际上，对于代码规范的检查不必总是用手工完成，有很多工具可以很方便地对代码进行静态分析，CheckStyle 就是其中非常优秀的一种，参见文献[41]。

在 UserManager 项目中，就使用了 CheckStyle 工具。项目不仅制定了代码编写规范，而且通过利用源代码检查工具 CheckStyle 将这一过程自动化，并且将其溶入到持续集成的过程当中。可以将 CheckStyle 看着是代码编写规范的规则集，它能够检查编写的代码是否符合这一规则集，通常这种规则集是以 xml 语言描述，如默认提供的 Sun 公司代码编写规范 sun_checks.xml。在本文前面分析中可以得知，构建工具 Ant 具有良好的扩展性，可以将 CheckStyle 代码检查这一任务集成到构建脚本中，且 Ant 本身也提供了对 CheckStyle 的支持。扩展任务如下：

```
<target name="checkstyle" depends="init" description="对 UserManager 源代码进行检查并产生检查报告. ">
    <checkstyle config="sun_checks.xml" classpath="%checkstyle_home%\checkstyle.jar">
        <fileset dir="${project.src.dir}" includes="**/*.java"/>
    </checkstyle>
</target>
```

其中：config 是指要使用的编码规范，在 UserManager 项目中，我们采用的编码规范是使用 Sun 公司的规范。当然也可以根据项目情况定制其他合适的编码规范。

classpath 是指要用到的 jar 文件，需要将 checkstyle 的 jar 包文件添加到系统环境变量中。

fileset 是指需要检查的文件集合，我们这里是指 UserManager 项目中源目录 src 下所有 Java 文件。

这样不仅将代码规范检查自动化，而且通过能否构建成功这一措施来要求项目成员按照编码规范来编写代码。如果有项目成员提交了不符合规范的代码到源码仓库中，项目构建（build）就不能成功，而且通过 CheckStyle 可以很容易发现那些代码不符合规范，从而迅速改进。

在持续集成中整合代码规范检查具有很多优点，但是需要注意一点：规范检查不能过于严格，否则可能出现经常因代码规范导致构建不成功，这样很挫伤团队成员的积极性。

5.3.4 自动化测试

实施持续集成的一个难点就是要实现自动化的 BVT 验收测试，这是整个过程中的一个关键部分。在 UserManager 项目中，使用了用户故事 User Story 来协助设计与开发。User Story 可以看着是系统的概要，每个 User Story 都代表系统的一个功能，参看文献[42]。在 UserManager 第一次迭代中，要实现多个用户故事 User Story，下图 5-4 是其中的一个。

名称：保存用户信息 SaveUser
事件：
1. 系统可以让用户填写个人信息，包括 FirstName、LastName；
2. 系统会记录这些信息；
3. 系统保存后，这些信息存储在用户信息库中，并且可以显示用户的信息。

图 5-4 保存用户 User Story

通常使用 CRC 故事卡来描述用户故事 User Story，每个 CRC 故事卡都有一个对应的验收测试，将其记录在 CRC 故事卡的反面，记录方式如下图 5-5：

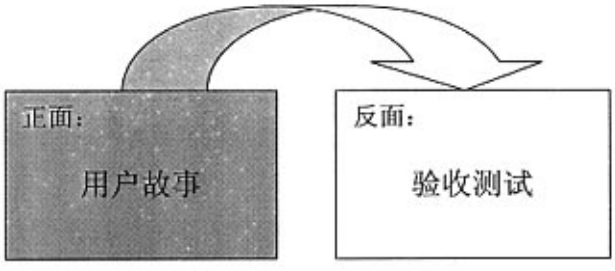


图 5-5 故事卡的正面和反面

上述的 SaveUser 用户故事 User Story 对应的 CRC 故事卡，如下图 5-6 所示：

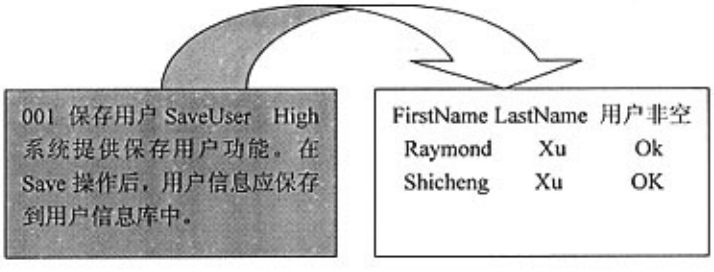


图 5-6 UserManager 中 SaveUser 的故事卡

SaveUser 故事卡的反面记录着这个 User Story 的验收测试，也即在添加了个人信息后（其中 FirstName 为 Raymond，LastName 为 Xu），此时检查用户信息应该是非空，说明“保存”save 功能正确，这就是 BVT 验收测试。正如 3.4.4 节说明的持续集成冒烟测试一样，不需要很详细，但能说明问题。

UserManager 项目使用了数据访问对象 DAO 模式[43]，这是 J2EE 企业级开发

中常用的设计模式。UserDAO 类是 UserManager 项目中最核心业务逻辑类，对用户 User 的 CRUD 操作管理就是由它来实现。UserDAO 类及其对应验收测试 UserDaoTest 类 DAO 模式层次结构如下图 5-7 所示：

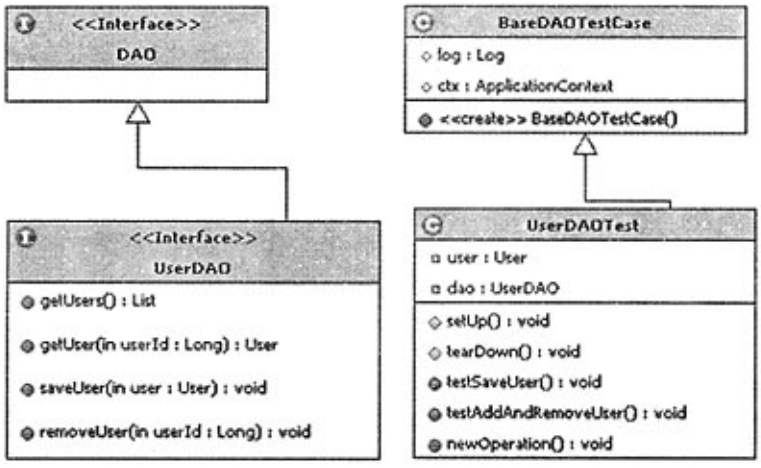


图 5-7 UserDAO 类层次结构图

UserDAO 类实现如下：

```
public interface UserDAO extends DAO {
    … …
    public List getUsers();
    public User getUser (Long userId);
    public void saveUser (User user);
    public void removeUser(Long userId);
    … …
}
```

按照测试驱动开发思想，应首先编写该用例故事的验收测试。从图 5-5 中可以看出，UserDAOTest 类实现对用户 CRUD 操作实现的测试。UserDAOTest 类对本用例故事测试代码片段如下：

```
public class UserDAOTest extends BaseDAOTestCase {
        … …
    public void testSaveUser() throws Exception {
        user = new User();
        user.setFirstName("Raymond");
        user.setLastName("xu");
        dao.saveUser(user);
        assertNotNull("primary key assigned", user.getId());
        log.info(user);
        assertNotNull(user.getFirstName());
    }
```

```
    ... ...
}
```

从上面不难看出，验收测试是由一组单元测试组成，如本用例故事中的testSaveUser。在此单元测试中使用了 assertNotNull 断言，即“保存”用户后，用户信息就不为空。关于 JUnit 框架的使用，可以参见第三章。

由于“保存”的业务逻辑实现代码并没有编写，在模拟开发人员 Developer1 提交代码后，PMS 中反馈信息如下图 5-8，验收测试没能通过，项目构建失败，显示如下图 5-9 所示。

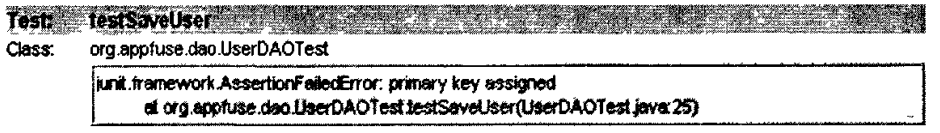


图 5-8 用例验收测试失败

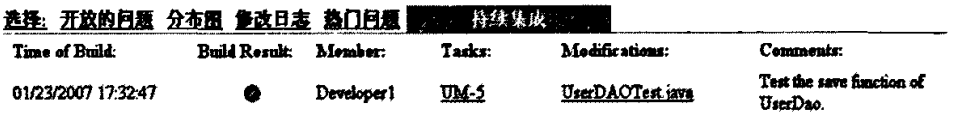


图 5-9 PMS 中持续集成反馈构建失败

UserManager 通过 UserDAOHibernate 类来完成“保存”用户的功能，实现代码如下：

```
public class UserDAOHibernate extends HibernateDaoSupport implements UserDAO {
    ... ...
    public void saveUser(User user) {
        getHibernateTemplate().saveOrUpdate(user);
    }
    ... ...
}
```

另一个开发人员 Developer2 将其迅速提交到源码仓库中，用来解决构建错误，同时来验证刚才编写代码的是否正确（持续集成提倡快速的提交）。在构建服务器 CruiseControl 中后台运行脚本如下：

```
... ...
[cc]一月-22 16:32:06 odificationSet- 1 modification has been detected.

[cc]一月-22 16:32:06 Project          - Project UserManager:   now building
Buildfile: projects\UserManager\build-cc.xml
update:
    [exec] U     src\org\appfuse\dao\hibernate\UserDAOHibernate.java
    [exec] 更新至修订版 2。
```

```
build:
 [junit] DEBUG - UserDAOHibernate.saveUser(33) | userId set to: 1
... ...
BUILD SUCCESSFUL
[cc]一月-22 16:32:22 Project      - Project UserManager:  next build in 5 minutes
[cc]一月-22 16:32:22 Project      - Project UserManager:  waiting for next time to build
... ...
```

从中可以看出，CruiseControl 能够自动检测到了源码仓库中的代码改变，马上进行构建。正如前面分析的那样，在构建之前首先需要将构建服务器中本地工作区间的代码更新至最新版本，然后进行编译、连接和自动化的验收测试。验收测试以及构建结果如下图 5-10、5-11 所示。

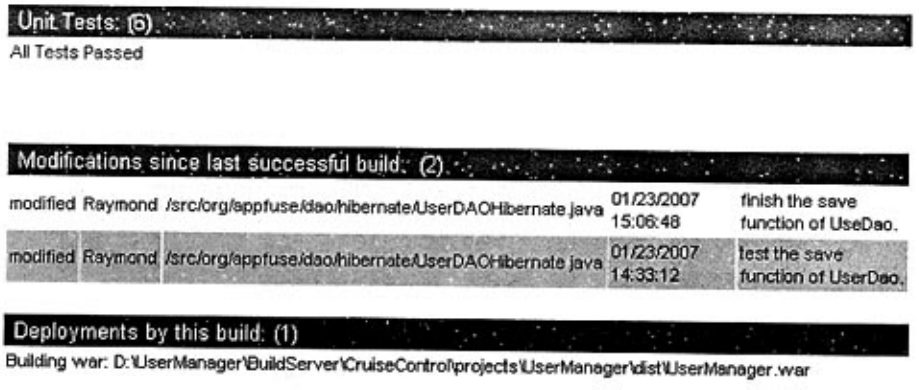


图 5-10 用例验收测试通过

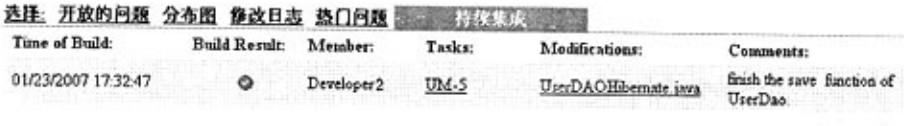


图 5-11 PMS 中持续集成反馈构建成功

5.3.5 持续反馈

一个构建发生故障时，项目组希望能够立即知道这一情况，以便就能够采取适当的行动来解决这一问题——不论是编译错误、测试故障、或是整体复杂性的增加。对问题了解得越迅速，获取的信息就越相关，解决问题的时机也就越有利，这也就是持续集成的价值。

然而，反馈的目的是为了采取行动。而在持续集成中，这个行动必须迅速，因为构建中断会影响到每一个人，因而由构建服务器部署的反馈机制也必须及时。这些反馈机制能够及时协助有关人员迅速采取行动。在 UserManager 项目中，除了上面显示的基于 Web 形式的反馈，还模拟实现了以下三种方式。

1．使用电子邮件

尽管以 Web 形式的反馈可以工作的很好，但这还是一个被动的过程，即需要项目成员经常在浏览器中查看构建信息。由于电子邮件使用非常普遍，可以让它成为了一种简单实用的反馈手段，而且 CruiseControl 也提供这一电子邮件反馈机制。UserManager 采用的策略是在构建失败时，发送邮件给项目所有人员，而不只是项目经理与刚提交代码的成员。如下：

```
<publishers>
    <currentbuildstatuspublisher file=" logs/${project.name}/status.txt " />
    <artifactspublisher dest="artifacts/${project.name}"
file="projects/${project.name}/dist/${project.name}.war"/>
    <email mailhost="localhost" returnaddress="root@localhost" skipusers="true"
reportsuccess="fixes" subjectprefix="[UserManager]"
      buildresultsurl="http://localhost:8080/cruisecontrol/buildresults">
        <failure address="raymond@localhost" />
    </email>
</publishers>
```

其中，mailhost 为邮件服务器，在运行 UserManager 项目之前，笔者在本机已经部署好了邮件服务器，参见图 5-2 模拟平台。在每次构建失败时，都会发送邮件给团队成员，如下图 5-12 所示。

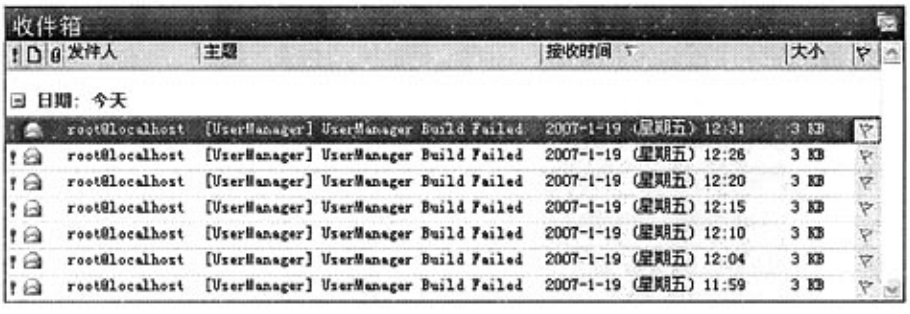


图 5-12 自动化邮件反馈机制

2．使用 RSS

对于使用持续集成的项目，电子邮件是事实上反馈机制的标准，其应用最为普遍，但是如果一次错误的构建修改的时间比较长，采用上面的反馈策略可能会收到不少邮件。如果团队成员收到太多的电子邮件，他们就会开始忽略这些邮件，这就会破坏反馈机制的初衷。为了不被大量的电子邮件所困扰，笔者模拟了使用 RSS 这种方式来协助反馈。RSS（Really Simple Syndication）[44]是一种描述和同步网站内容的格式，它搭建了信息迅速传播的一个技术平台，是目前使用最广泛的 XML 应用。发布一个 RSS 文件后，通常是符合标准的的 XML 文件，该文件

基于某些事件进行更新，利用 RSS 阅读器可以立即获取此更新并创建一条指示新内容的消息。因此如果是由于同一错误导致的构建失败，RSS 阅读器将不会收到重复的信息，只有发生新的构建事件，RSS 文件更新时，才会得到相应的通知。由于这种反馈机制减少了接收到的电子邮件消息的数量，因而可以作为另一种可选的反馈机制。图 5-13（使用的 RSS 阅读器为 Briz）展示了 UserManager 项目中的 RSS 反馈机制，相比使用电子邮件反馈，可以看到其接受到的重复信息要少很多。

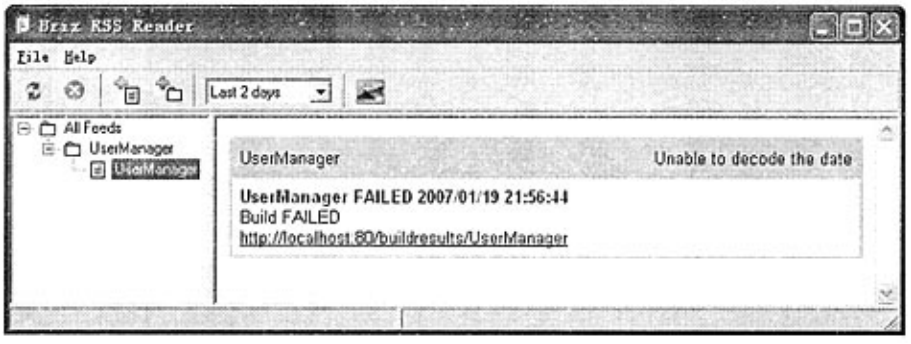


图 5-13 RSS 的反馈机制

3．使用可视化熔岩灯

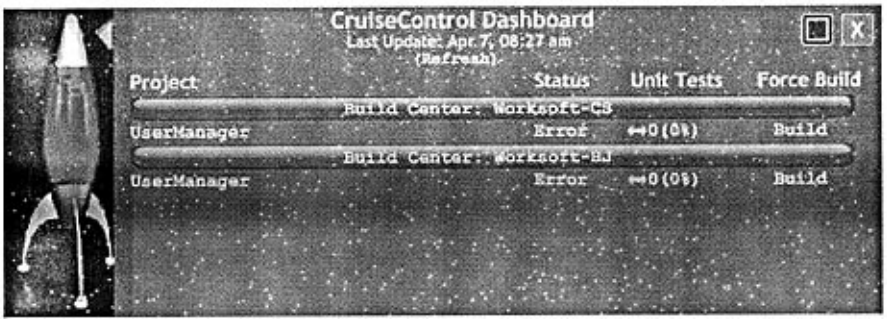
在持续集成的各种反馈方式中，还有有一种反馈机制经常应用于实际企业项目开发中，即基于可视化的熔岩灯（Lava Lamp）。当构建失败时，熔岩灯显示红色；构建成功时，熔岩灯显示绿灯。在实践中采用这种方法可以很好活跃项目组工作气氛，真正把握项目的当前状况，效果非常显著[45]。基于 Yahoo Widget 工具，笔者在 UserManager 项目中同样对上述反馈方式进行了模拟。

首先在反馈设备（本文中为模拟平台）上安装 Yahoo Widget 引擎，然后下载基于持续集成的 Widget 熔岩灯工具，修改配置文件如下：

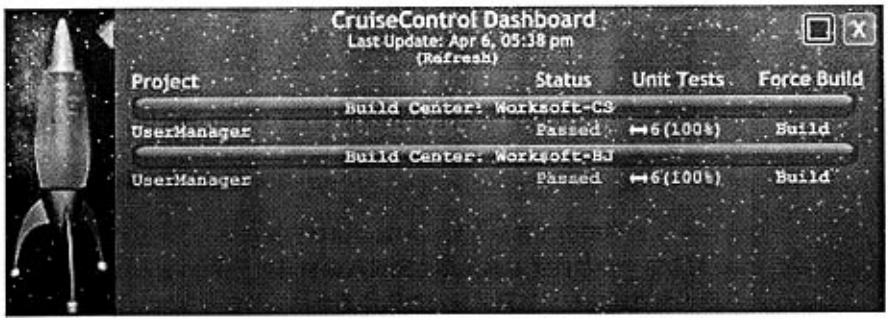
```
<preference name="ccURLs">
    ……
    <defaultValue> UserManager,http://127.0.0.1:8080/cruisecontrol </defaultValue>
</preference>
<preference name="jmxURL">
    ……
    <defaultValue>http://127.0.0.1:8000,http://127.0.0.1:8000</defaultValue>
</preference>
<preference name="projectNames">
    ……
    <defaultValue>UserManager |Project2</defaultValue>
</preference>
```

其中 ccURLs 为 CruiseControl 运行的 web 首页；jmxURL 为 CruiseControl

中 JMS 工作路径；projectNames 为需要监控的项目。UserManager 构建失败与成功时反馈于下图 5-14 所示。



（a）构建失败，熔岩灯显示红色



（b）构建成功，熔岩灯显示绿色

图 5-14 构建失败与成功时的熔岩灯对比

作为持续集成的反馈，还有两种非常好的方式。一种是基于 SMS，另一种是使用 X10 电子装备。如果团队有成员不在计算机前而构建又失败了，此时保持与构建状态同步的一种方法是通过短消息服务（SMS），即向手机发送文本构建短消息，其发送机制与发送邮件反馈机制一样。X10 电子装备是对可视化桌面部件的一种扩展，它可以将构建结果以某种方式（如无线技术）映射到电子设备上，文献[46]介绍了这两种方式的反馈。

5.3.6 自动化部署

如果部署比较复杂，人工工作量和潜在错误就会随之增加，因而需要自动化部署，它能够帮助迅速而准确地部署新应用。部署托管 J2EE 应用，可以采用标准化的、自包含的部署模块（war 文件或者 ear 文件）而变得容易。基于 J2EE 的 UserManager 项目部署时采用的就是将部署模块 UserManager.war 放到 Web 服务器 Tomcat 的相应目录。

创建 war 文件可以包含在构建过程中，同样利用 Ant 工具的扩展功能，在

Ant 构建文件添加<war>任务，即可以自动化生成 UserManager.war，如图 5-15。<war>构建任务如下：

```
<target name="war" depends="compile" description="Packages app as WAR">
    <mkdir dir="${dist.dir}"/>
        <war destfile="${dist.dir}/${webapp.name}.war"webxml="${web.dir}/WEB-INF
    /web.xml">
            <classes dir="${build.dir}/classes"/>
            <fileset dir="${web.dir}">
                <include name="**/*.*"/>
                <exclude name="**/web.xml"/>
                <exclude name="**/junit.jar"/>
                <exclude name="**/*mock.jar"/>
                <exclude name="**/strutstestcase*.jar"/>
            </fileset>
        </war>
</target>
```

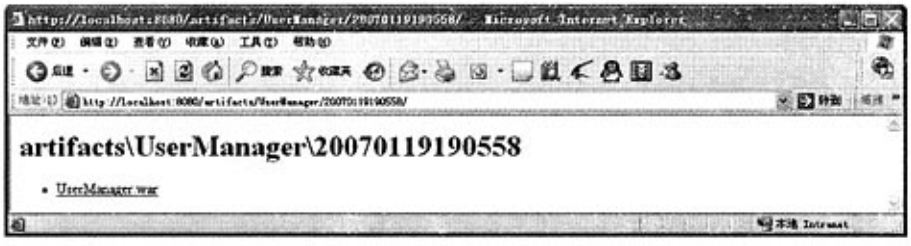


图 5-15 自动化生成的 UserManager.war 文件

为了达到自动化部署的目的，还需要把生成的.war 文件拷贝到 Web 服务器相应目录。同样在 Ant 脚本中增加一个任务（target），在这里命名为“deploywar”。“deploywar”任务配置如下：

```
<target name="deploywar" depends="war" description="Deploy application as a WAR file">
    <copy todir="${tomcat.home}/webapps" preservelastmodified="true"
        file="${dist.dir}/${webapp.name}.war"/>
</target>
```

在运行 UserManager 项目之前，需要在部署服务器（DeployServer）上先安装好 Web 服务器 Tomcat。其中“${tomcat.home}”为 Tomcat 的安装目录，“deploywar”任务的目的就是将我们自动化生成的 UserManager.war 文件拷贝到服务器的 webapps 目录下。利用 Tomcat 服务器的自动解压热部署功能，就可以运行 UserManager 项目。在浏览器中输入：http://localhost:8090/UserManager，其中 8090 是我们配置的 Tomcat 端口，运行界面如下图 5-16。

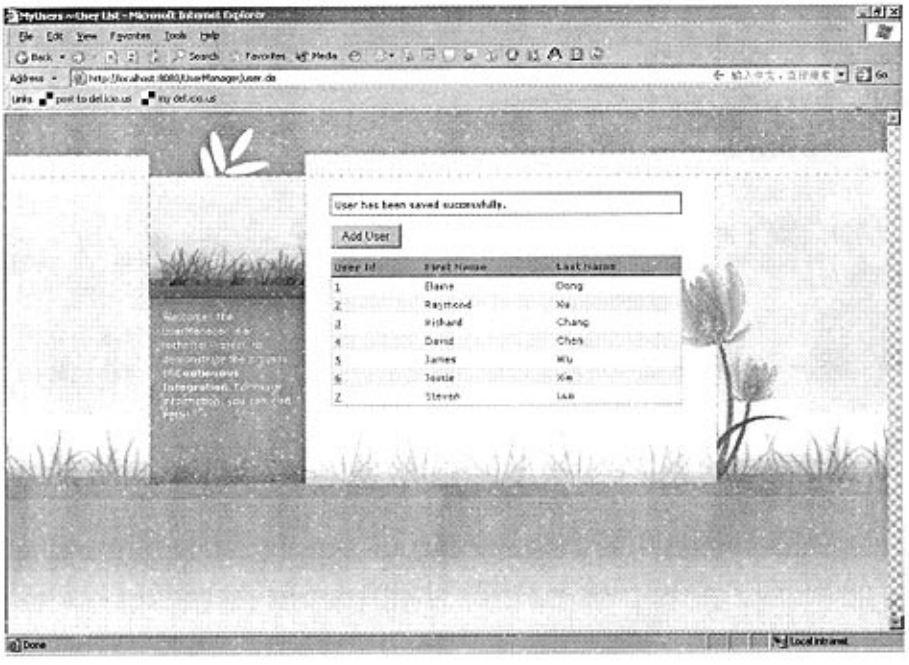


图 5-16 UserManager 自动化部署后运行界面

5.4 实施过程建议

由于在软件开发过程中采用持续集成需要完成很多前置工作，目前也有一些实现持续集成的工具，但是即使这样，在一个新团队中引入持续集成，还是有很多注意事项。现根据笔者的实施经验，总结如下：

1. 适当培训

持续集成的过程是一个强调自动化的过程，如果没有工具的支持，那么很难完成整个过程。在一个团队中实施持续集成的过程可能涉及到很多工具，比如构建工具 Ant，源码管理工具 Subversion，持续集成工具等。“工欲善其事，必先利其器”，对这些工具的概念及使用作适当的培训是值得的（但并不需要成为这方面的专家，一个公司只要一两个这样的配置管理员即可）。在笔者的持续集成实践中，就发现这样的问题，因为团队成员对 Subversion 不熟悉，导致实施过程不顺畅。

2. 保留过渡阶段

实施持续集成需要一定的代价，尤其是对刚接触持续集成的团队，很难改变传统软件开发过程思维，因此可以适当保留一定的过渡阶段。在笔者实践过程中，通常采用两种方式来完成。第一种方式是从构建时间间隔人手，刚开始的时候，

集成的频率放小一些，每天集成一到两次，或采用定时构建；第二种方式是从过渡项目入手，可以先选用一个小项目，比如本章中用户管理系统，项目本身不要太大，主要目的是为团队的成员熟悉持续集成的过程。

3．重视测试

在持续集成中，自动化的测试始终处于最核心的位置。因为持续集成的排错能力同样取决于测试技术，对于集成成功，但有非常多 Bug 的构建，就失去了持续集成的意义。尽管持续集成的过程不要求一定要采用 XP 中的测试驱动开发方法（XP 中另一项优秀实践，非常容易与持续集成结合在一起），但要求单元测试要进行的足够多，要能够对系统进行端到端的检验。那种认为“只要代码能够运行就可以了，即使单元测试不能通过也无所谓”的观点，对于实施持续集成的项目相当危险，因为不知道错误会在什么时候出现。

4．重视反馈

重视持续集成提供的反馈信息，对其报告的错误要及时进行修改。保证在源码控制系统中的构建是可稳定的，在持续集成中是具有最高优先级的，因为后面的每一次构建都是依赖前面的构建为基础的，因此对于反馈错误的修改也应具有最高优先级。应该是在修改完成当前出现的错误之后，才进行新的功能点的增加工作。在笔者实施的过程中，经常发现团队中有成员忽略当前的错误，而一味强调新功能的实现，这很容易出现问题。

在软件开发过程采用持续集成的开发、管理方法，会为确保软件产品质量，发布零缺陷软件产品，起到重要作用。但软件开发是一个持续改进的过程，经典软件工程书籍《人月神话》[47]中一再强调“没有银弹”，提高软件开发效率，改进软件开发过程需要我们不停地在项目开发中实践，总结经验。

5.5 本章小结

本章通过 UserManager 项目模拟实践了企业的持续集成实施过程，说明了整体实施模型及平台模拟，具体演示了持续集成中三大关键实践的实现过程，包括使用 Subversion 控制源码仓库以及 CruiseControl 配置项目过程。本章最后对于持续集成中自动化过程都给出了可行性方案，包括自动化代码规范检查、自动化验收测试、多种反馈方式以及自动化部署，并且总结了在实际项目中实施持续集成建议。

第六章 工作总结与课题展望

6.1 工作总结

在项目开发过程中，尤其在需求不是很确定的软件项目中，经常因集成问题导致项目出现拖延或者崩溃。为了达到对软件项目更好的管理，提高软件项目的交付质量，针对传统集成模式的不足，现代敏捷软件开发提出了采用持续集成的集成方式。针对当前软件开发人员对于持续集成实践存在的一些问题，本文就其展开研究。论文所涉及的主要工作和所取得的进展包括如下几个方面：

(1) 以集成过程为视角，分析了集成方式在软件开发中的重要意义，总结比较了传统软件开发模型中的集成方式的缺点或者不足之处，从理论根源分析了从持续集成的思想来源。持续集成思想是建立在“迭代递增”集成模式之上，并且借鉴了微软 MSF 中“每日构建”实践。

(2) 为了使开发人员更清楚理解持续集成的过程，本文具体分析了持续集成理论基础，对于其关键实践自动化过程和 BVT 测试过程的实现原理通过 Java/J2EE 技术进行了深入研究。

(3) 针对当前缺乏支持整个持续集成过程的工具解决方案，设计开发了持续集成插件，并将其应用于项目管理中，提出了一种基于该插件的项目管理平台 PMS。本文中所开发的持续集成插件及其应用项目管理平台 PMS 在实际项目中运行效果良好，其可以直接作为实践持续集成的一种方案应用于企业项目管理中，也可以基于该方案作进一步扩展。

(4) 针对当前持续集成应用不够，缺乏实际项目实践指导，本文根据持续集成理论，给出了用户管理系统这一具体项目持续集成实践的整个过程，对于持续集成中源码配置、构建脚本编写、代码规范检查、验收测试、持续反馈以及自动化部署等都提供了解决方案。

(5) 在实践的基础上，给出了实施持续集成的建议，并且对于进一步研究方向作了说明。

6.2 进一步研究工作

本文在撰写的过程中，查阅了大量的文献资料，研究了很多现代软件开发过程中的集成方式，比如 Rational 统一过程、微软过程以及敏捷开发过程，也参照了这些过程中许多优秀实践。尽管本文对持续集成的思想来源、基础理论以及其关键技术的实现原理作了详细分析，并且也在具体项目中实践了持续集成的实施

过程，但是由于时间的原因，本课题还有许多方面值得进一步的完善和深入研究。

(1) 改进测试过程，整合测试驱动开发实践。持续集成的过程是一个强调测试的过程，测试驱动开发作为 XP 的另一项最佳实践，可以而且应该与持续集成结合在一起使用。Matrin Fowler 认为无论在什么样的软件开发过程中，都应该尝试的持续集成与测试驱动开发。尽管本文中对于持续集成中 BVT 冒烟测试过程的实现原理作了深入说明，也在 UserManager 项目实践过程也作了一些尝试，但在实际企业项目中，测试代码的编写远比本文中讨论的复杂的多，还可能涉及到很多其他技术，比如 Mock 对象、重构等。怎样将测试作为开发活动的一部分，在持续集成的有机平台中最佳整合测试驱动开发实践是课题的下一步要研究的地方。

(2) 改进持续集成插件。本文中持续集成的插件开发是基于 JIRA 框架，尽管 JIRA 框架提供了良好的插件机制，但开发的插件是不能脱离 JIRA 框架基础运行的 web 容器（本文中为 Tomcat），如果改变容器，插件将不能运行；例外，以 web 插件的形式在 PMS 平台中展示项目构建信息，可能对于多个项目之间的信息交互带来困难，造成很多冗余信息。解决这两个问题，可以采用 Portal/Portlet 技术，开发基于 Portlet 标准的持续集成插件。JBoss 实验室的 Kosmos[48]项目对这方面作了一些研究，但也刚刚开始，处于发展中。作为持续集成插件的进一步改进，可以在 PMS 平台中引入 Portal/Portlet，但怎样将这些技术与持续集成思想更好地结合在一起，更好地应用于企业软件项目开发中值得研究。

(3) 持续集成 IDE 工具开发。现代软件的开发都离不开高效的集成开发工具 IDE。在 Java 开发环境中，通常使用的 IDE 有 Eclipse，NetBeans，IDEA 等等，目前使用最为广泛的为 Eclipse。这些工具基本上都集成了对 Ant、JUnit 和 Subversion 的支持，但到目前为止，还没有 IDE 工具全面完善地支持持续集成这一过程。随着持续集成被越来越多地团队所采用，这一需求已经越来越突现。Eclipse 基金会组织已经考虑在 Eclipse 加入对持续集成的支持，但这些都刚刚开始，基于持续集成的 IDE 工具开发，可以作为进一步研究方向。

参考文献

[1] Robert C. Martin 著，邓辉译．敏捷软件开发：原则、模式与实践．北京：清华大学出版社，2003

[13] Steve McConnell. Rapid Development: Taming wild Software Schedules. Redmond, WA: Microsoft Press, 1995

[3] Sridhar Raghavan, Donald Chand. Diffusing Software Engineering Methods. IEEE Software. 1990, 6(4):81-90

[4] 陈宏刚，熊明华，林斌等编著．软件开发过程与案例.北京：清华大学出版社，2003

[5] 谭力文，田毕飞．美日欧跨国公司离岸服务外包模式的比较研究及启示．中国软科学，2006(5)：128-134

[6] Ron Jeffries. Integration Hell[EB/OL]. http://c2.com/xp/IntegrationHell.html,2000

[7] Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, Pearson Education, 2000

[8] Amr Elssamadisy, Gregory Schalliol. Recognizing and responding to bad smells in extreme programming. Proceedings of the 24th International Conference on Software Engineering, 2002

[9] Matrin Fowler, Matthew Foemmel. Continuous Integration[EB/OL]. http://www.martinfowler.com/articles/ContinuousIntegration.html, 2000

[10] Continuous Integration Testing Conference. http://www.citconf.com./index.php, 2006

[11] Jesper Holck, Niels Jorgensen. Continuous integration and quality assurance:a case study of two open source projects. Free/open Source Software Development. Idea Group Publishing, 2005

[12] 张湘辉等编著．软件开发的过程与管理．北京：清华大学出版社，2005

[13] Steve McConnell. Code Complete. Redmond, WA: Microsoft Press, 1993

[14] Winston Royce. Managing the development of large software systems. Proceedings, IEEE WESCON. 1970

[15] Conradi R, Fuggetta A. Improving software process improvement. IEEE Software, 2002, 12(4):92-99

[16] Stephen R. Schach 著，韩松，邓迎春，李萍等译．Object-Oriented and Classical Software Engineering．北京：机械工业出版社，2003

[17] Ivar Jacobson, Grady Booch, James Rumbaugh 著，周伯生等译．统一软件开发过程．北京：机械工业出版社，2002

[18] Craig Larman 著，张晓坤等译．敏捷迭代开发管理者指南．北京：中国电力出版社，2004

[19] 金敏，周翔编著．高级软件开发过程——Rational 统一过程、敏捷过程与微软过程．北京：清华大学出版社，2005

[20] Jim McCarthy. Dynamics of Software Development. Redmond, WA: Microsoft Press, 1995

[21] Michael A. Cusumano, Richard W. Selby. How Microsoft builds software. Communications of the ACM. 1997, 40(6):53-61

[22] 雷剑文，陈振冲，李明树著．超越传统的软件开发——极限编程的幻想与真实．北京：电子工业出版社，2005

[23] Grady Booch 著，冯博琴等译．Object-Oriented Analysis and Design with Applications．北京：机械工业出版社，2004

[24] Mike Clark. Pragmatic Project Automation——how to build, deploy, and monitor Java application. The Pragmatic Programmers Publication, 2004

[25] Erik Hatcher, Steve Loughran 著，刘永丹，陈洋译．使用 Ant 进行 Java 开发．北京：电子工业出版社，2005

[26] Andrew Hunt，David Thomas 著，陈伟柱，陶文．单元测试之道——使用 JUnit．北京：电子工业出版社，2005

[27] Panagiotis Louridas. JUnit: Unit Testing and Coding in Tandem. IEEE Software. 2005, 22(4):12-15

[28] Steve McConnell. Daily Build and Smoke Test. IEEE Software. 1996, 13(4):9-13

[29] Ten reasons to use JIRA for bug tracking, issue tracking, and project management. http://www.atlassian.com/software/jira/learn/10reasons.jsp

[30] Mike Cannon-Brookes. Pluggable Web Applications. TheServerSide Java Symposium, 2005

[31] Lasse Koskela. Driving on CruiseControl. JavaRach Journal, 2004, 3(7):1-12

[32] Ben Collins-Sussmann, Brian W. Fitzpatrick, C. Michael Pilato. Version Control with Subversion. O'Reilly Press, 2004

[33] Ben Collins-Sussman. The subversion project: building a better CVS. Linux Journal, 2002(94):33-35

[34] JIRA Codegeist. http://confluence.atlassian.com/display/CODEGEIST, 2006

[35] W3C Recommendation. Document Object Model Technical Reports[EB/OL].

http://www.w3.org/DOM/DOMTR, 2004

[36] Nicholas Chase. 理解 DOM[EB/OL]. http://www.ibm.com/developerworks/cn/views/xml/tutorials.jsp?cv_doc_id=84890, 2004

[37] Erich Gamma, Richard Helm, Ralph Johnson,et al. Design Patterns:Elements of Reusable Object-Oriented Software. 北京：机械工业出版社，2002

[38] Rod Johnson. J2EE Development Frameworks. IEEE Computer. 2005, 38(1):107-110

[39] Matt Raible. Seven simple reasons to use AppFuse[EB/OL]. http://www-128.ibm.com/developerworks/java/library/j-appfuse/index.html, 2006

[40] 牟军，吕立．使用轻量级框架进行 J2EE 应用开发．小型微型计算机系统．2006(6)：1149-1152

[41] Panagiotis Louridas. Static Code Analysis. IEEE Software. 2006, 23(4):58-61

[42] James Newkirk, Robert C. Martin 著，王钧译. Extreme programming in practice. 北京：人民邮电出版社，2002

[43] Deepak Alur，John Crupi，Dan Malks 著，刘天北，熊节译．Core J2EE Patterns: Best Practices and Design Strategies．北京：机械工业出版社，2005

[44] Gabe Beged-Dov, Dan Brickley, Rael Dornfest, et al. RDF Technology Site Summary .http://web.resource.org/rss/1.0/spec, 2000

[45] Mike Swanson. Automated Continuous Integration and the Ambient Orb. http://blogs.msdn.com/mswanson/articles/169058.aspx, 2004

[46] Paul Duvall.Automation for the people:Continuous feedback. http://www-128.ibm.com/developerworks/java/library/j-ap11146/, 2006

[47] Frederick P. Brooks 著，汪颖译．The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition. 北京：清华大学出版社，2002

[48] The Kosmos Project of JBoss Labs. http://labs.jboss.com/portal/kosmos/, 2005

致 谢

时光飞逝，三年的研究生生活即将结束。三年的校园生活经历，不仅仅是文化知识的增加，更重要的是使我有机会重新认识自我，它将是我人生中最宝贵的一笔财富。当然，这里有很多人值得我去感谢和铭记。

饮其流者怀其源，学有所成念吾师。首先要感谢我的导师杨邦荣教授在三年的研究生学习、生活中，对我的悉心培养和大力帮助。论文从选题、构思、撰写及修改整个过程中都得到了杨老师的精心指导，杨老师严谨的治学态度和宽厚仁爱的学者风范，使我在耳濡目染中受益终生。

感谢信息院的各位老师，正是你们的辛勤劳动为我打开了一扇扇通往科学知识殿堂的大门，正是你们的循循善诱让我感受到了学习的乐趣，而你们的谆谆教导我也必将永远铭记在心。

感谢陈浩博士给我耐心而细致地讲解 Mozilla 项目的构建模式，本文中的很多理念都是借鉴 Mozilla 项目构建的思想。

感谢文思创新 PMS 项目组全体成员，正是由于 PMS 项目的实施才使我真正体验到持续集成在企业级项目中的价值。

感谢我的室友，实验室师兄弟，以及 04 级许多同学，很多问题的解决都是由于你们的帮助，你们给我带来了许多的快乐。

感谢我的父母和哥哥，给我无微不至的关爱。谢谢你们在任何时候都关心我、支持我，你们的关心和支持是我不竭的动力。

最后还要感谢各位评委老师在百忙之中抽出宝贵时间认真审阅我的论文。

徐仕成

2007 年 5 月

于中南大学

攻读硕士学位期间主要的科研成果

发表论文情况：

[1] 徐仕成，杨邦荣. 基于 CruiseControl 的持续集成实现方案. 计算机数字与工程，已录用.

参加项目情况：

[1] 参与 Biznavi 株式会社企业商务应用系统的设计与开发；

[2] 参与 Worksoft 项目管理系统 PMS 的设计与开发，负责持续集成插件的实现；

[3] 参与全国银行业监督委员会 CBRC 门户 Portal 系统的整合与开发。