PRQA Programming Research

Trinity®

通过静态分析和持续集成保证代码的质量

Jason Masters

October 2013

持续集成技术可确保在每次有增量变化的时候，开发团队中每个成员的代码都会被整合、构建到源代码中，并且保证这些代码都经过了测试。进行持续交付的目的是保证代码一直处于经过充分测试的状态，进而保证代码的质量，让产品随时可以发布。静态分析就是要保证代码符合编码规范，并且没有程序错误，从而自动提高代码的质量。将这些方法和技术同时使用，就能够提高代码的质量，降低开发成本，并增强项目的可预测性。

前言

现代软件开发团队面临着很多挑战，这些挑战包括：产品交付期限越来越紧，团队的分布越来越广，软件的复杂度越来越高，而且对软件的质量要求越来越高。

本文分为两个章节。第一章讨论持续集成的原理，持续集成如何简化软件开发生命周期，以及静态分析如何配合持续集成。第二章研究特定的工具套件，以及在实际使用中如何获得该套件。主要研究 Jenkins 持续集成和 PRQA 的静态分析工具(QA•C, QA•C++ 和 QA•Verify)，并说明他们是怎样结合在一起使用，以及他们所生成的信息。

第一章

第一部分：持续集成和持续交付

持续集成（CI）和持续交付（CD）这两个术语常常一起使用。持续交付的最终目标是保证版本控制系统存储库中的代码始终是经过检测和验证的。代码要通过所有必要的测试（编码规范合规检查，单元测试，系统测试，代码覆盖率测试，验收测试，部署测试等）及所有的验证环节。这样，产品的发布时间就不是由该项目的开发方或工程方决定了，而是由产品经理决定。

在嵌入式领域，软件只是制成品的一个组件，所以保持软件始终处于可发布的状态好像并不是十分紧要，但是通过本文可以看出，这样做的确会带来很大帮助。另外，现在的消费品中常常带有需要用户购买之后自行更新的软件（笔者最近为了解决电视机的声音问题，对自己购买的 LCD 电视进行了更新），在这种情况下，保持软件始终处于可发布的状态就显得更加紧要了。对于很多产品而言，软件是设备的附加价值的体现，而且为产品增加新功能也是一种销售策略：可以经常性地为用户增加一些新功能，而不一定非要在传统的发布周期中才发布新功能。

开发过程中的集成阶段往往比较困难/麻烦/伤脑筋，持续集成技术可使这一过程实现自动化，从而帮助实现持续交付。

问题

传统的开发模式（比如：瀑布式开发、V 模式开发、轻量的敏捷开发）由不同阶段组成。最先对持续集成感兴趣的阶段是编码阶段，在该阶段，开发人员根据分派给自己的任务来编写代码。随着开发的不断进行，开发人员很可能要对存储库进行修改，但是他们只是更新自己开发的代码，不会同时更新其它开发人员的修改。所以，在所有的代码都开发完成之后，会有一个特别的集成阶段，将所有开发人员对代码所作的修改进行整合，以形成一个可测试的软件构建。

我们可以估算出该阶段的开始时间（根据对每项任务的判断进行估算），但是很难预测出截止日期，也就是说很难知道完成集成工作需要花费多长时间。开发人员编写的代码可能会相互冲突：每个开发人员编写的代码在独立运行时，也许能够运行得非常好，但是当该代码和其他开发人员的代码结合在一起时，可能就会发现它不能与其它开发人员对代码所作的修改

相兼容。这种冲突必须要解决——通常情况下，这都需要手动解决（虽然也有工具可以帮忙），然后还必须重构代码。但是，代码重构过程中也可能会引进更多的矛盾，这样集成工作就会陷入不断重复的状态。

集成是一项非常重要的工作，在极端情况下，解决冲突所耗的精力要远远超乎一开始编写代码时所花费的精力。

这些可能出现的冲突和由此产生的工作量是开发人员不愿意更新存储库中的局部代码的原因之一：更新可能会破坏现在可以正常运行的代码构建。有些问题可能是其他开发人员的代码引起的，所以开发人员就不想花时间去解决这些由其他人的代码引进的问题。

有好多种解决代码冲突的方法。其中一种方法就是改善开发步骤：每次只允许一个开发人员对存储库进行修改。对存储库中的代码进行更新之后，要立刻解决所有的新问题，这样开发人员要做的就不仅仅只是提交代码那么简单了。这种方法可以避免冲突——但是代价很高。开发人员在提交代码的时候，必须保证只有他一个人可以访问这个存储库：在该开发人员解决好所有冲突之前，不能够有任何其它代码提交进来。（否则，所有的提交工作都要重新开始。）这种排外的访问限制会造成很大的瓶颈，而且很难实行：有些公司甚至利用实体对象来表示是否允许提交：如果桌子上没有该实物就不可以提交。如果团队比较大，不是只有几个开发人员，那么该方法就很难实施，如果团队分布在不同地区，那么这种方法就更不可行了。

完成集成之后就会生成供测试阶段使用的可执行程序——可能会有很多测试阶段——比如，编码规范合规检查，单元测试，集成测试，系统测试，覆盖率测试，验收测试，部署测试等。在各个测试阶段，如果测试失败，表示代码需要重新编写、重新集成，并且重新测试。这样会有一个潜在的问题，就是开发人员在编写代码时所花费的时间与重写代码时所花费的时间不同：这两者之间的时间差越大，开发人员重新熟悉这些代码的难度越大，因而修改阶段的工作量也会大大增加。

问题的根源在于对开发阶段的划分。软件的编写不是一个单向过程：不是集成工作一开始，编码工作就结束了。虽然也有比较轻松的做法，就是对开发任务进行划分，再将他们分配给开发人员，让他们平行地开展工作。但是集成往往是系列性的，所以要将集成工作同时分配给多个开发人员就比较困难了。

解决方案

正如持续集成所推崇的那样，这个解决方案的部分内容就是用一系列小阶段（如图 1）取代传统的编码测试、单元测试、集成测试、系统测试和验收环节。每个小阶段都包含编码测试、单元测试、集成测试、系统测试和验收环节——开发人员每次对存储库进行修改时都要执行这些步骤。每次更改之后都要对代码进行集成、测试和验证，代码每通过一项测试，其质量就越接近于可发布版本。

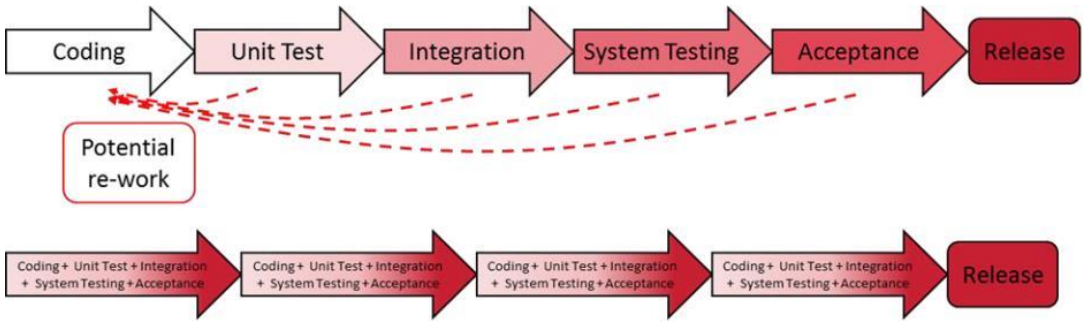

图 1：将大阶段转化成小步骤

将集成工作扩展到整个开发阶段意味着繁重的集成工作被分割成了一个个更小、更易操作的任务。当问题能够尽早被发现并独立解决时，就不会得到恶化，因而就不可能形成严重的集成问题——所谓的“集成地狱”。这就是持续集成。可以将这个过程视为从瀑布式向 V 模型开发模式发展，从敏捷开发向持续集成发展。

这种方法只有在大多数测试过程和任务都可进行实现自动化操作的情况下才可行。因为如果每次存储库中有增量提交时，都要手动完成那些任务，很明显是不可行。

提交之前要进行测试

持续集成模式存在的一个问题就是，它无法防止不能运行的、坏的或不兼容的代码进入存储库中：这种模式只能做到在这类代码提交之后，立刻发现它们。开发人员在处理及修复这类代码造成的问题时，必须严格遵守规则。

在提交之前对代码进行测试的做法是为了保证只有正常的、可运行的、经过测试的代码才被提交到核心存储库中（存储待发布代码的库），因此，这一做法也是实现持续交付的关键所在。这一概念就是说，只有能够发布的代码才能进入存储库中——所以，在提交代码之前，必须对代码进行全面测试。虽然有很多方法可以帮助实现这一目标，但是最常见的做法就是保证未经测试的代码绝不会进入核心存储库。另一个方法就是同时使用临时存储库和核心存储库：在这里只是简单介绍一下这种方法，本文一直会提到这种方法。

当开发人员向存储库中提交代码时，代码不会自动提交到正式存储库，而是会到临时库中。这种做法对于开发人员来说很容易理解：他们提交代码时，和向正式存储库中提交代码一样。持续集成服务器会从临时库中检测到新提交的代码，然后构建并测试这些代码。如果这些代码能够通过所有的测试，那么本次提交就会被转移到正式存储库中，同时也会附上提交者的名字和日志文件。这个做法对于开发人员来说也很好理解：因为就和他们将代码直接提交到了正式存储库中一样（只是时间上有点滞后）。

如果代码未能通过所有的测试，那么本次提交就会保留在临时库中。开发人员会收到通知：他们可以采取必要的补救措施来修复存在的问题，而此时正式存储库中的代码则保持不变。

这一过程要求所有的测试都能够自动执行，而且定义明确。随着项目的成熟，测试也可能会有所改变：比如，beta 测试可能主要着眼于基本功能，而对于待发布的代码会有更严格的测试标准，在维护阶段则会更加注重测试新功能。

持续集成系统

实现持续集成的关键在于测试阶段的自动化水平。自动化分析一旦开始，持续集成系统就能够运行相应的测试，并可以根据测试结果采取适当的措施。所以持续集成系统可以被看作一个精密的调度程序，可启动编译和测试程序。

持续集成系统的核心需求就是能够自动实现以下功能：

- 发现已提交的代码（既可通过监控存储库来发现提交的代码，也可通过接受一些外部刺激物来发现——通常在存储库中）
- 将存储库中的代码检出到配备所有必要构建和测试工具的机器上
- 构建代码
- 对产生的可执行文件进行全面测试
- 报告构建和测试的结果
- 选择性地启动其它测试组合，可能的话最好根据之前所作的测试的结果来选择进行什么样的附加测试。

这一工作可以利用一些脚本撰写工具和命令行工具来完成，实际上，很多公司一开始都是采用的这种方法。然而，管理特定程序中的特定脚本，本身也是一项工作，因此这也会分散开发人员编写代码的精力。

向持续集成演变

很多组织机构都有每日夜间构建，先检出存储库中白天完成的代码库，然后进行构建和测试：这可以被视为持续集成的原始模式。第二天早上会检查夜间构建所产生的结果。

这种方法也有局限性：

- 首先，测试完代码之后不会立刻查看结果。开发人员可能早上才会去检查前一天进行夜间构建时出问题的那些代码。所以，开发人员在第二天才会发现代码存在的问题。而这时，他们可能已经开始进行另外一项工作了。
- 其次，也许这一点更为严重，就是构建和测试基本上都是由定制的脚本来控制的。这些脚本需要定期维护，而且他们本身也会存在问题。增加新测试会比较耗时间，而且这些脚本很难用于不同的项目中。

与大公司联系比较密切的一个问题是：基本只有开发人员，可能还有一些团队领导能看到这些夜间构建产生的结果。对于更高层次的管理人员而言，他们可能会希望知道哪些代码通过的测试，哪些代码没有通过测试——但是，要获取合适的概要总结信息可能会比较困难：因为这些信息主要集中在开发人员这边。紧接着，管理员可能还会想知道和上周相比，现在的代码是否通过了更多的测试，即：趋势分析信息。只有保存了之前的构建所获得的结果，才可能会有这些数据——这就意味着应该将一段时间内的结果都保存起来。

使用现成的持续集成系统是解决这些问题最简单的方法。图 2 展示了一个典型的持续集成方案是如何工作的。

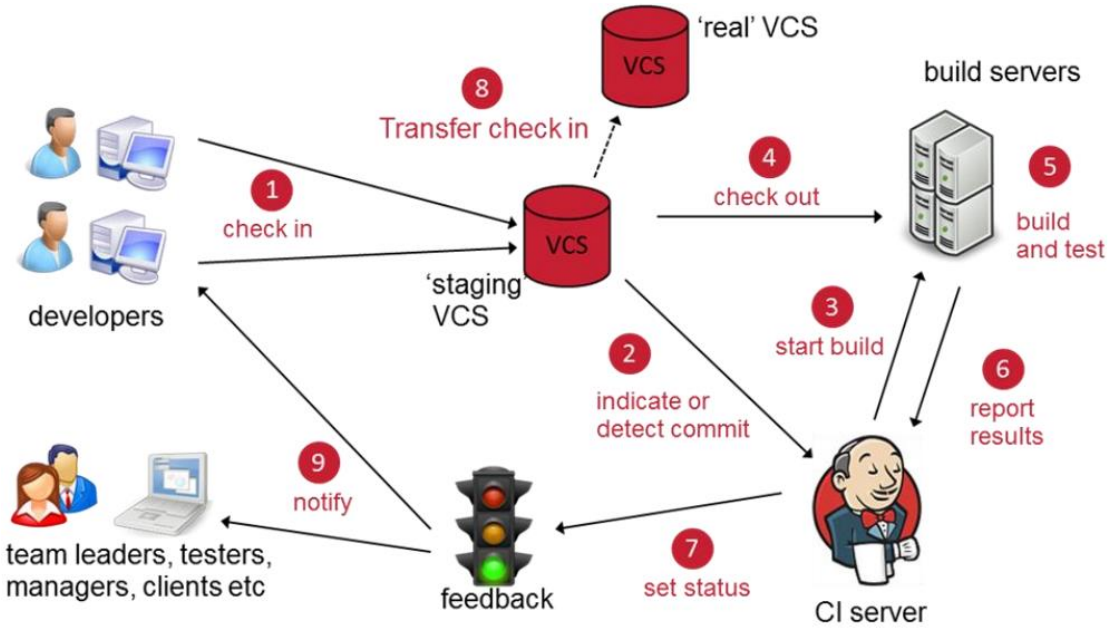

图 2 持续集成概观

这个典型的系统流程是这样工作的：

1. 开发人员将代码存放到版本控制系统（临时）库中。
2. 版本控制系统（VCS）通知持续集成服务器有新交的代码，或者持续集成服务器定期检查存储库，查看是否有新提交的代码。
3. 持续集成服务器开始在构建服务器上进行构建。
4. 将存储库中最新提交的代码检出到构建服务器的本机工作区中。
5. 建立代码并根据预先设定好的的测试标准对代码进行测试。
6. 重要的结果以及其它需要保存的重要文件会一起回传到持续集成服务器。
7. 持续集成服务器会给出构建的最终结果——通过/未通过/不稳定。“不稳定”意味着所有的功能性测试都通过了，但是代码覆盖率可能还没有达到目标要求。
8. 如果构建达到了成功的标准，那么就需要将临时存储库中的代码转移到正式存储库中。如果构建失败，就不能将代码检出到正式存储库中（正式存储库中只存放有用的、经过测试和验证的代码）。
9. 持续集成服务器会通知所有在服务器中注册过并表示对构建感兴趣的人：他们可以登录持续集成服务器查看构建的状态以及更多其它的附加信息。

各个击破

的确可以将把测试代码时所需的所有步骤集成为一个大的整体任务，然后在有代码提交到存储库中的时候，只要执行这一任务就可以了。但是，通常情况下，将这些任务拆分成一个个小任务或事件，然后再按顺序执行这些事件会更好。我们可以对事件的执行顺序进行配置，后面的事件如何执行，取决于前面的事件执行之后所取得的结果。这样，就可以将一系列执行不同测试的事件链接在一起，通过将很多简单的步骤结合在一起，来形成一系列复杂的任

务。

分阶段测试的基本原理在于：测试失败时，能够及时发现。如果单元测试早就失败了，就没有必要将软件下载到硬件上，并执行代码覆盖率测试。所以，在持续集成过程中，使用各个击破的测试方法是非常有效的。软件只有通过当前阶段的测试，才能进入到下一测试阶段。这样可以确保开发人员能够尽快得到反馈，从而尽快提出修复方案。

管理大量细小的事件要比管理一个庞大的整体事件容易一些，比如，添加一个新测试或改变测试的顺序。图 3 展示了如何运行一系列事件。如果代码通过，就可以进入下一阶段。如果代码通过所有阶段的测试，那么这个代码就能成为一个候选发布版本。

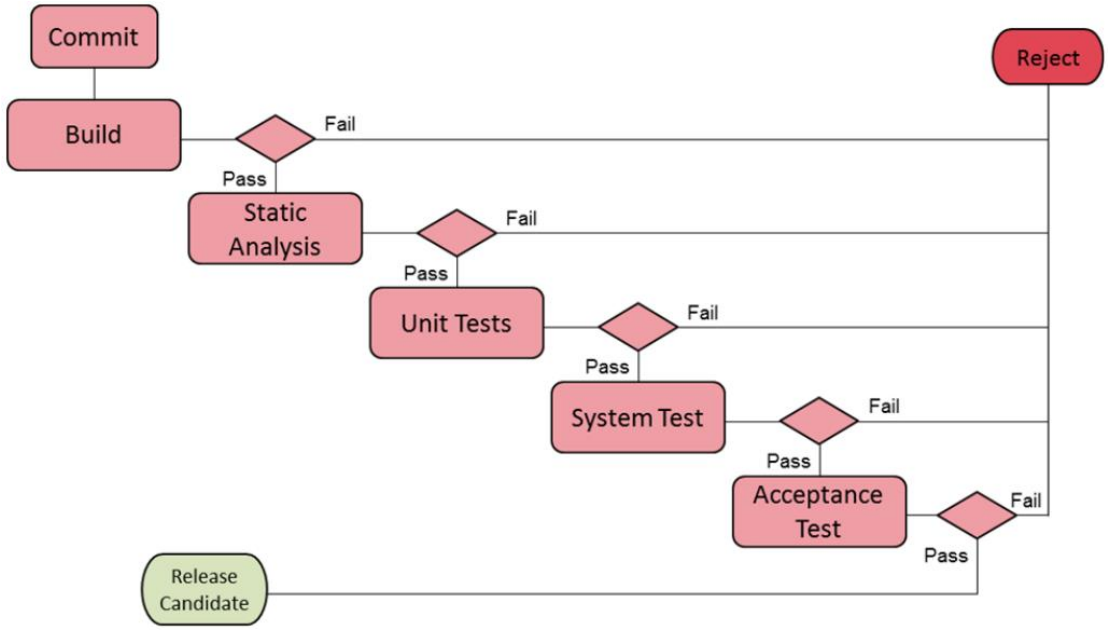

图 3 典型的事件运行序列

第二部分 通过持续集成进行静态分析代码检测

静态分析是一个广义的概念，它包含了很多种对源代码进行分析的方法，进行静态分析时，不需要真正构建和运行代码（虽然好工具也可以显示出该项目在运行过程中的情况）。静态分析可以被看作代码质量谱的结构端的测试，动态测试则主要负责功能方面的测试。静态分析可保证代码遵循最佳编码惯例，还能够发现很多问题，包括未定义行为、可移植性问题、由语言误用产生的危险但是合法的构建、代码清晰度问题、代码的实际布局问题等。静态分析也可以发现动态测试不能发现的一些问题，还能够帮助简化脚本，使脚本更易于维护，更加高效。虽然自动化的静态分析不能完全取代代码评审，但是对于代码评审来说，它是非常重要的一部分。

静态分析一个非常重要的方面就在于：在完成第一行代码之后，就可以立即进行静态分析，这就将测试代码的重点放到了开发初期。

关于对代码进行静态分析带来的优势，我们已经在之前的 PRQA 白皮书中讨论过了【1】。

静态分析的关键在于，能够在早期和持续的应用程序代码评审中大量减少代码中存在的问题，确保在项目的编码阶段，能够遵循最佳编码惯例。为了提高效率，需要大部分代码评审工作都能够实现自动化：代码还是需要开发人员来评审，但是传统代码评审中找出的问题，大多数都能够自动检测到。“尽早、经常性”的代码评审策略与持续集成的概念非常吻合——因此，将自动代码检查与持续集成过程结合起来，是大有裨益的。

代码集成中存在的问题同样存在于执行编码规范的过程中：如果将问题一直留到代码编写完成，那这个做法一定是有问题的。

如果将新编码规范应用于之前未遵循该编码规范的代码，常常会发现很多违规行为。要解决所发现的问题不仅会耗费很多时间，而且在修改功能的过程中，还可能会使代码更加糟糕，甚至还可能引进更多严重的违规行为。

所以，在编写代码时就遵循编码规范会更加有效率、更加安全；开发人员在编码代码的时候，就应该及时解决所发现的编码问题。

因此，进行集成和执行编码规范时，都应该“尽早开始，并经常进行”。

增量执行编码规范最主要的优点就是能够保证代码的质量，但同时也会带来其它好处。可以将很多度量都收集起来，作为静态分析的一部分，而且增量测试可以形成一个趋势图，可以通过观察所形成的趋势图来判断项目是否健康。如果度量开始与期望偏离（可能是通过与“优质标准的项目”进行对比），就可以有针对性地采取措施，使项目回归正常。

第二章 部署示例：Jenkins and PRQA 工具

第一节 Jenkins 作为持续集成系统

现在有很多持续集成工具，既有免费的，也有商业的。最近的研究显示【2】【3】，Jenkins 正发展成为最受欢迎的持续集成工具。Jenkins 是 Hudson 持续集成系统的一个分支，但是 Jenkins 拥有最活跃的生态系统。

在 MIT 许可下，Jenkins 是免费的，但是和一些开源软件一样，Jenkins 也有付费版本，并会提供技术支持。虽然 Jenkins 是一个开源项目，但是对核心代码所作的所有修改，都由核心提交者严格把控，其中一位核心提交者就是项目架构师 Koshuke Kawaguchi 【5】。

系统概述

Jenkins 是用 Java 写的，由一个很小的内核和一个内置网站服务器组成。大多数用户与 Jenkins 的互动都是通过常规网页浏览器来实现的，不需要安装特别的客户端软件。

灵活性是优秀的持续集成系统的主要特性之一，它能够满足所有公司的业务要求。Jenkins 有两个非常重要的结构特点，使得它既可自定义又可扩展：主/从分布式构建结构和插件。

主/从结构

虽然 Jenkins 经常以主/从安装形式配置，但其实 Jenkins 也可以部署在一台独立的电脑上。由一个主机控制何时进行代码的构建和测试，并为用户提供接口。Jenkins 上的所有任务都叫做事件。由一个或多个从属机器执行实际的构建和测试工作。构建（交付物，如最终的二进制文件、程序包、测试报告，等等）产生的所有重要结果，都会从从属机器转移到并保存到主机上。中间文件，如目标代码，还是保留在从属机器上（会定期移除）。主机会保留所有的历史构建结果，因而可以判断出项目的趋势。

从属机器没有什么特别的——只要是装有所需的开发和测试工具的现成电脑就可以了。从属机器可以运行与主机不同的操作系统——这对于为多平台编写代码有重要意义。从属机器也可以连接到硬件设备，以便可以将嵌入式代码下载到实际设备中进行测试。从属机器也不一定要是实际机器，现在虚拟机正在越来越广泛地被用做从属机器，采用基于云的解决方案，因为这种方法成本较低。

插件

Jenkins 完成的大多数自定义任务都是通过插件来实现的。Jenkins 内核中定义好的应用程序界面（API），可以通过安装插件来丰富图形用户界面（GUI）的配置选项，这些功能在使用的时候就好像是内核的一部分一样。

现在有超过 800（正在增多）种插件，插件涉及很多方面，包括：版本控制系统的交互作用、代码构建、测试、报告、交付物管理。通过安装相关的插件，可以定制 Jenkins，以适应项目的构建和测试需求。

仪表盘

Jenkins 可通过仪表盘模式显示所有事件的整体状态。每个事件的状态都以点状形式显示，颜色表示上次执行的状态（在默认情况下，红色表示失败，黄色表示不稳定，蓝色表示成功——但是和 Jenkins 中的大多数功能一样，颜色也是可以自定义的）。闪烁的点表示该事件正在构建中。用“天气”的标志，通过晴天（太阳）、阴天（雷云）等天气状态，以比较直观的形式总结出最近的构建情况。

仪表盘会显示事件队列——待构建事件的列表。同时还会显示从属机器的状态，指出哪些事件是不运行的，哪些正在执行。图 4 显示的是 Apache Software Foundation【5】的 Jenkins 仪表盘。

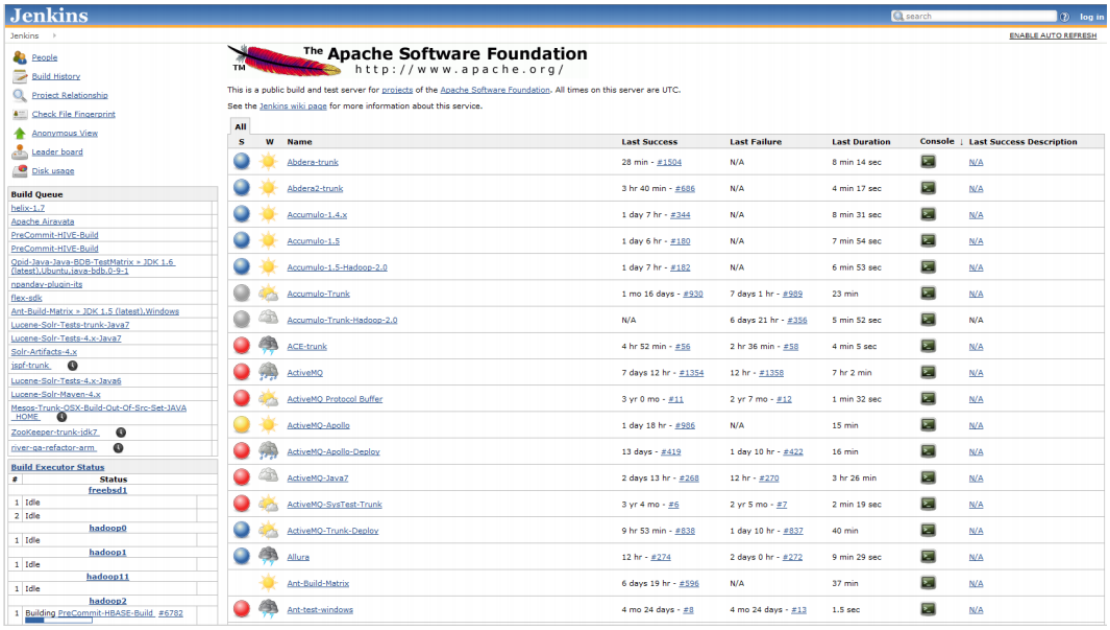

图 4 Jenkins 仪表盘

图 4 显示的是 Jenkins 仪表盘。右侧显示的是事件列表。左下方显示的是从属机器列表，从属机器列表的上方显示的是构建队列。

事件

每个事件都有一个页面记录构建的历史情况，通常情况下，是一些图形化输出，显示测试状态以及与最近的构建版本、代码修改及其它重要或相关的数据。

一个事件可被拆分成 3 个主要阶段：

1. 构建前。在这个阶段，会从存储库中检出最新的代码。通常是先检查存储库，看是否有新代码提交进来，如果有就检出。也可能是根据时间安排或在完成了其它事件之后，对存储库中的代码进行检出。
2. 构建中。这一步在大多数实际工作完成之后开始，包括：从编译和耦合到对代码进行测试。Jenkins 会记录所有命令的控制台输出，以便排除故障。
3. 构建后。在这一过程中，需要对构建好的代码进行进一步测试。同时还要收集并解析测试结果，最后将结果显示出来。并将所需的构建版本存档到主机上。

构建阶段和构建后的阶段需不需要包含多个步骤，取决于构建/测试的需求，以及所安装的插件。

一个简单的事件可能要进行以下配置：

1. 构建前：版本控制系统应用程序，存储库的路径，模块名称
2. 构建中：包含简单的“make”命令的 shell 执行步骤
3. 构建后：对交付物进行存档时要使用构建阶段生成的可执行文件的名称。

在执行事件的时候，Jenkins 会从存储库中检出最新版本的代码，然后在 shell 中执行“make”

命令，并将生成的可执行文件复制到主机上。如果其中某一步执行失败了，那么后面的步骤就都无法执行了，所以事件的状态就会显示为“失败”。只有当所有的步骤能准确无误地执行完成之后，该事件的状态才可能显示为“成功”。如果能达到这个最基本的水平，就可以用 Jenkins 安排夜间构建。

入门指导

Jenkins 的入门很简单——它可以以一个完整、可运行的网络应用程序的存档文件形式存在。该文件可以下载【7】并执行——但是，为了正确安装，要将它以服务/后台程序的形式安装，并按权限以用户的身份执行。

第二部分：利用 PRQA Jenkins 插件进行静态分析

PRQA 的 Jenkins 插件【8】，可与我们的静态分析工具 QA C 或 QA C++一起使用，来对源代码进行分析。在构建完成之后的阶段，该插件自动执行以下主要任务：

1. 分析项目
2. 生成合规报告
3. 将项目中的信息总数与可配置的临界值进行对比，如果超过了该临界值，就将构建状态设置为“不稳定”：可以将其当作大门，阻止后续任务的执行。
4. 有选择性地将分析结果上传到 QA Verify 中，这是 PRQA 的网络质量管理系统。

分析阶段的工作可以通过“依赖模式”来开展——所以只需要对在上次分析之后有所改变的文件进行分析，从而尽量减少项目的分析时间。也可以在构建阶段进行分析（比如 Makefile 集成），并向插件提供列表，列出所有经过分析的文件，让插件可以识别到已分析过的文件。

如图 5，插件将概要信息显示在事件网页上。

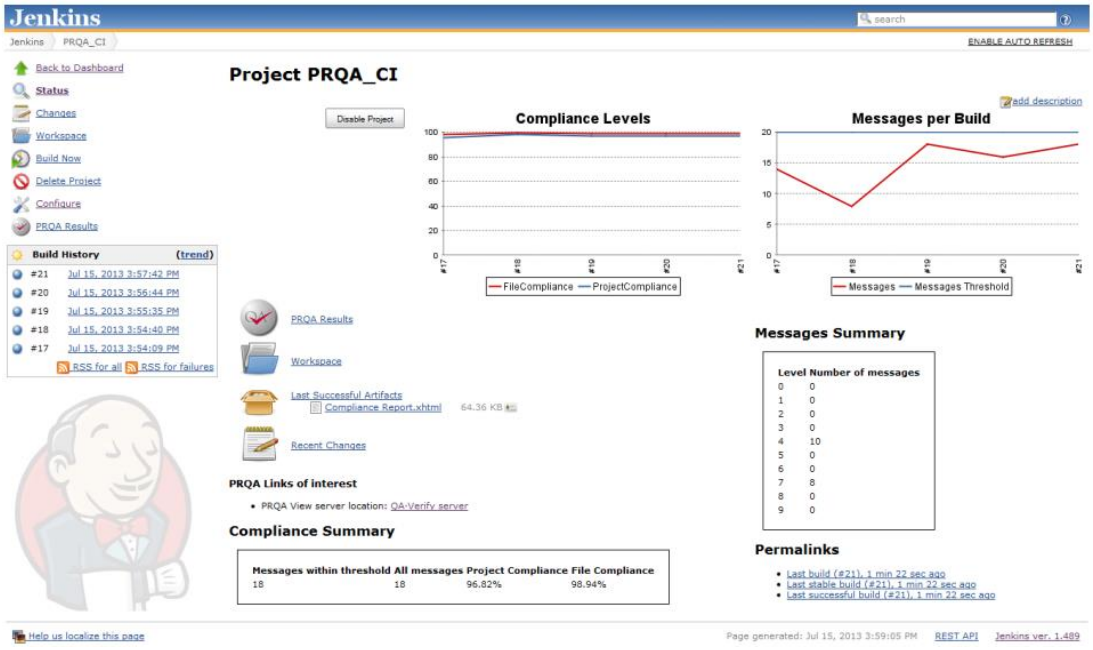

图 5 使用 PRQA 插件的事件（为适应页面大小改变了一下格式）

遵从度和每次构建时产生的信息总数都能以曲线图形式显示出来，这样就很容易看到编码规范的违规趋势。合规汇总表显示项目和文件的合规百分比（这些数据可从 QA C/QA C++合规报告中获取，报告会详细指出不符合编码规范的代码）以及所产生的信息总数。第二个表显示的是各个级别的分解信息。插件会自动存档所有报告，所以在项目合规状态方面能够提供更详细的信息。

违规数量的临界值可设置为绝对值，比如 0，表示不允许有任何违规行为。此外，还可以设置执行临界值的信息级，这样就可以将信息的严重性等级考虑在内。如果代码需要的维护工作不是特别复杂，样式方面的问题也不是很严重，就可以视为通过；但是如果代码有严重的问题，就不能通过。

能够将分析结果上传到 QA·Verify 是该插件最强大的功能。将结果上传之后，开发人员就可以看到深入的分析结果，包括带注解的源代码，这些结果中会显示违反编码规范的具体位置。该插件还允许添加、管理和报告违反编码规范的代码，并可自动校对代码是否遵循编码规范，如 MISRA C。QA·Verify 也会向其它项目相关者（比如，管理员，甚至是客户）公开这些质量信息。图 6 展示了 PRQA 工具是如何与 Jenkins 配合使用的。

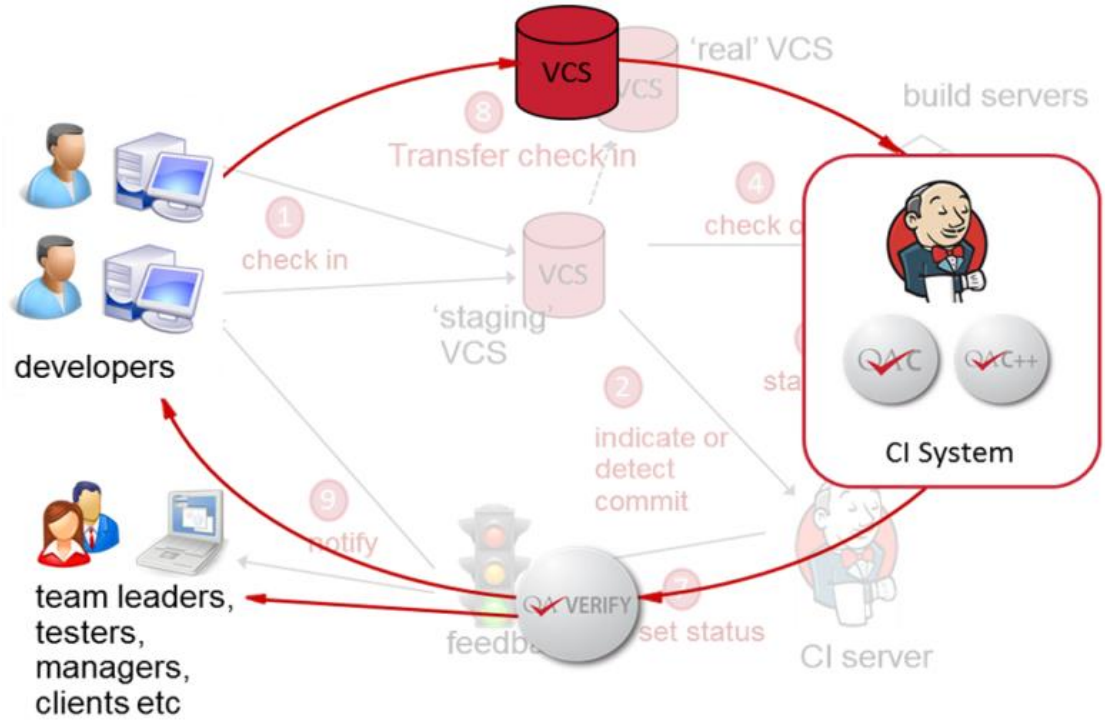

图 6：Jenkins 与 QA C，QA C++，QA·Verify 配合使用的情况

开发人员像平常一样提交代码（代码先提交到临时库中，当代码通过所有测试时，再转入正式库）。要配置持续集成服务器，以便可以对原代码进行 QA C 或 QA C++分析。概要信息，如：通过集成服务器可以了解代码是否遵循了最佳编码惯例，而且，分析结果还可以自动上传到 QA·Verify。这样开发人员就可以通过网页浏览器看到带注释的源代码，从而准确找到不符合编码规范的地方，以便在重新提交代码之前修复这些问题。

其它项目相关人员可以在 QA·Verify 中看到带注释的源代码，但是管理层和质检人员等，会希望了解更多其它更具概括性的数据。管理层可能不太想看源代码，他们感兴趣的应该是：代码中有多少问题——问题的数量是呈上升还是下降趋势。因为可以看到项目中不同级别的信息，管理层在做决定时，就可以根据必要的数据来判断应该如何重新分配资源。

结论

通过持续集成，很容易就能够“循序渐进地”将最佳编码实践方法应用于静态分析中。持续集成能够让集成和测试环节贯穿整个开发阶段，这将为开发带来很多好处。因为这样有利于在问题变得严重之前尽早解决，还能保证存储库中的代码一直处于高质量状态。

使用插件，很容易能够将 PRQA 静态分析工具 QA C 和 QA C++与 Jenkins 持续集成结合起来，从而迅速给出反馈，指出代码是否遵循了编码规范和最佳实践方法。该插件还能够将结果上传到 QA·Verify，以便进行深入的调查研究和趋势分析。

将持续集成和 PRQA 的静态分析工具结合起来能够帮助提高代码的交付质量，并且能保证在规定时间和预算范围内实现交付。

参考

1. PRQA 白皮书：《持续的代码检查白皮书》（Continuous Code Inspection WP）129A/01/13, Clayton Weimer 和 Fergus Bolger 作
   http://www.programmingresearch.com/resources/white-papers/

2. 《持续集成的缩放世界》（The Shrinking Expanding World of CI），Dr Dobbs Journal, 2012.2，
http://www.drdobbs.com/architecture-and-design/the-shrinking-expanding-world-of-ci/232601132

3. 《Cloud Bees Jenkins 调查 2012》（Cloud Bees Jenkins Survey 2012）
   http://www.cloudbees.com/jenkins/jenkins-ci/2012-survey/state-jenkins-ci.cb

4. Jenkins 持续集成网站（Jenkins CI Website），http://www.jenkins-ci.org

5. Apache 软件基金会的 Jenkins 仪表盘（Apache Software Foundation Jenkins Dashboard），https://builds.apache.org

6. Kohsuke Kawaguchi, http://kohsuke.org

7. Jenkins 的最新版本 WAR 文件,
   http://www.jenkins-ci.org/war/latest/jenkins.war

8. PRQA Jenkins 插件, https://wiki.jenkins-ci.org/display/JENKINS/PRQA+Plugin

创提信息科技（上海）有限公司 – Trinity Technologies

专注于嵌入式软件研发质量和自动化测试的方案和咨询服务，提供覆盖软件测试整个流程的完整的解决方案，包括从研发前期的代码级测试到后期的系统级测试，从静态分析到动态测试，从编码检查，单元测试、集成测试到性能测试和测试覆盖率分析等。

公司通过专业的自动化工具（如 DT10, VectorCAST, PRQA, SQUORE 等）和服务满足不同客户对软件质量和测试的需求，持续协助客户改进软件研发质量和效率。客户主要集中在高安全和高可靠性领域，如国防和航空航天、轨道交通、汽车电子、医疗器械、工业控制、通讯和电力电子等行业。公司提供的领先的解决方案不仅为数以百计的客户提高产品质量，还协助客户遵循高安全和高可靠性行业的合规性要求，如 DO-178B/C，IEC61508, EN50128, ISO26262, IEC62304 和 MISRA 等行业标准，并获得相关机构认可和认证。