The Paper Research of Continuous Integration System for Cross-Platform

Wendy Y.C. Yang a,1, Amy J.C. Trappey a

aDepartment of Industrial Engineering and Engineering Management, National Tsing Hua University, Taiwan

Abstract. With the rapid evolution of computer and programming languages, supporting multiple hardware and software platforms became necessary for software systems. Cross-platform software may be divided into two types. One requires individual building or compilation for each platform that it supports, and the other one can be directly run on any platform without special preparation, e.g., software written in an interpreted language or pre-compiled portable bytecode for which the interpreters or run-time packages are common or standard components of all platforms. For example, a cross-platform application may run on Microsoft Windows on the x86 architecture, Linux on the x86 architecture and Mac OS X on either the PowerPC or x86-based Apple Macintosh systems. Cross-platform programs may run on as many as all existing platforms, or on as few as two platforms. Therefore, the continuous integration system was developed. Continuous integration is used for merging all developer working copies to a shared mainline several times a day. Continuously building and integrating your application to reduce risk and improve quality. It is an important part of modern professional software development. The main aim of continuous integration is to prevent integration problems but there still have several problems in the existing continuous integration systems. Thus, this research do the market survey to find the problems on the existing continuous integration systems. After investigating, author tried to figure out the solutions to solve the problems.

Keywords. Cross platform, Continuous Integration (CI), Integration System

Number of words. 2799 words

1. Introduction

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build including test to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage. Integration plays a crucial role in the success of iterative and incremental software development. It serves as an information radiator for the team to monitor the current state of the product under development. Consistently practiced, continuous integration can help guaranteeing that

1 Corresponding Author: wendy940245@yahoo.com.tw
working software is delivered at the end of an iteration/increment. As such, continuous integration needs to be made a sustainable practice in incremental and iterative software development. For a piece of software to be considered cross-platform, it must be able to function on more than one computer architecture or operating system. Developing such a program can be a time consuming task because different operating systems have different application programming interfaces (API). For example, Linux uses a different application programming interfaces for application software than Windows does. Just because a particular operating system may run on different computer architectures. It does not mean that the software written for that operating system will automatically work on all architectures that the operating system supports. One example as of August 2006 was OpenOffice.org, which did not natively run on the AMD64 or Intel 64 lines of processors implementing the x86-64 standards for computers[1]. This has since been changed, and the OpenOffice.org suite of software is “mostly” ported to these 64-bit systems. This also means that just because a program is written in a popular programming language such as C or C++, it does not mean it will run on all operating systems that support that programming language or even on the same operating system on a different architecture. The main aim of continuous integration is to prevent integration problems, referred to as “integration hell” in early descriptions of XP. Continuous integration isn't universally accepted as an improvement over frequent integration, so it is important to distinguish between the two as there is disagreement about the virtues of each.

As shown in the Figure 1[2], the build server checks out new code from source control, compiles it and tests the code. Once the code is tested, the build server deploys it to QA. At this point, the build server can also launch scripts to perform integration testing, user interface testing, advanced security testing more on this soon and other tests requiring a running version of the software. Consistent with Agile requirements that emphasize a continually working version of the software, our CI server automatically reverts to the last successful version of the software, keeping a working QA system available even if integration tests failed.

![Figure 1. Process chart of the continuous integration system.](image-url)
Cross-platform software means software products that simultaneously support multiple operating systems, e.g., Microsoft Windows and various Linux distributions. Further, limiting the scope to cross-platform software that contains both platform-dependent and platform independent components. While cross-platform software harbors the potential for software developing organizations to expand markets and earn more profit, it is also more costly to develop. In our experience, most of the extra cost has come from the increased integration and testing efforts. Therefore, the ability to harness automatic integration and testing technologies is essential for cutting the cost of developing quality cross-platform software. To this end, within continuous integration, a widely used software practice in the agile community lays the promise for harnessing automatic integration and testing technologies. However, to development teams that have little or no prior experience in practicing continuous integration, cross-platform issues present even greater challenges. Figure 2 is Continuous Integration Process the infrastructure [3]. To effectively take up the challenges, the development team will need a common language of cross platform continuous integration to analyze, design, discuss, implement, and improve its continuous integration practices. This paper presents the first results of our attempt to establish such a common language.

2. Literature review

This section introduces the pre-integration procedure and the status communication and test separation.

2.1. Pre-integration procedure

The pre-integration procedure of continuous integration refers to which actions are prescribed prior to performing integration, e.g. by checking in source code. In some cases, these procedures can be practically non-existent, with one source arguing that the benefit of continuous integration can be measured as the time saved by developers not compiling and testing before checking in Miller (2008). Others offer the option without prescribing any mandatory process, with developers running small subsets of tests rather than waiting for the centrally executed test suite (Holmes and Kellogg, 2006). It is also related how developers “typically run tests before checking in changes” (Brooks, 2008) and that “developers could ensure their check-in both by manually compiling the
code and by executing the set of unit tests, but few did so” (Downs et al., 2010). These examples appear to share the sentiment that “fundamentally, it needs to be acceptable to break the build” (Rogers, 2004).

Other sources provide examples or mandatory pre-integration procedures, where developers are obliged “to integrate their own contributions properly” (Holck and Jørgensen, 2007). This can take various forms, such as reviews before checking in (Downs et al., 2010 and Janus et al., 2012), running light-weight “developer builds” (Rasmusson, 2004), performing “a pre-check before committing” (Woskowski, 2012). One source stresses the importance of testing before integration. The developers must compile the whole system, design and code the needed unit and integration tests, and finally execute the entire unit and integration test suites. Clearly, there are stark contrasts in what procedures are required before integrating. One source remarks on this, stating that it is “a common approach to institute a strict and thorough recommit procedure that will make it as hard as possible for developers to break the build”, but that such procedures also have negative side effects (Rogers, 2004).

![Figure 3. The process of continuous integration system.](image)

2.2. Status communication

There are various approaches to communicating the continuous integration status, e.g. sending notifications of build failures (Dösinger et al., 2012; Matsumoto et al., 2012), in development projects. Dispatching e-mails is common (Sturdevant, 2007, Stolberg, 2009, Kim et al., 2009, Kim et al., 2009, Hoffman et al., 2009, Downs et al., 2010 and Gestwicki, 2012), either notifying the last person to check in Ablett et al. (2007), “relevant developers” (Yuksel et al., 2009) or “the whole development project” (Lacoste, 2009). Other communication methods can be used, such as really simple syndication (Stolberg, 2009), web pages (van der Storm, 2005) or dashboards (Baumeister and Reutelshoefer, 2011). This may then be displayed on “information radiators” (Ablett et al., 2007, Goodman and Elbaz, 2008, Hoffman et al., 2009 and Downs et al., 2010), making the current status visible to all in the vicinity. Other methods include differently colored lava lamps and robotic dogs walking up to the
responsible developers, “displaying to the team that it is not happy with that developer, in a friendly, funny and playful way” (Ablett et al., 2007). Figure 4 is Patterns in action [5].

![Figure 4. Patterns in action.](image)

One source extensively discusses and evaluates differences in notification methods, and concludes that a combination of multiple communication channels can have a great impact on awareness of and responsiveness to broken builds (Downs et al., 2012).

2.3. Test separation

Test separation refers to the practice of segmenting test suites into multiple parallel or sequential activities. Similar to the case of modularization, sources that touch upon this issue tend to be positive examples, and it is difficult to find explicit statements to the effect that testing is not separated, although to our understanding this is the case in a number of the articles in the study. That being said, one source argues that even though it's common to have “a single integration process that compiles the code, runs the unit tests and the acceptance tests, builds deployment packages for QA and the customer, validates code coverage and checks coding standards amongst other things” (Rogers, 2004), this is not necessarily a good thing, as they increase the build duration and thereby delay feedback. Consequently, tests are sometimes separated into multiple activities.

One separation approach is to “segment tests by functional area and to only run those tests thought to be affected by the code change” (Brooks, 2008) or to split test suites by components (Kim et al., 2008). Commonly tests are separated into sequential stages based on the time it takes to execute them and the context in which they run (Sturdevant, 2007, Sunindyo et al., 2010, Brooks, 2008 and Yuksel et al., 2009), e.g. “one an ‘express build’ that just runs unit tests to give a basic idea of the success of an integration; another a longer ‘full’ build that actually runs database processes, acceptance tests, deployments, etc.” (Roberts, 2004), or slower tests are performed on a different schedule altogether (Tingling and Akbar, 2007 and Woskowski, 2012). Another source states in passing that continuous integration “is the automation of...
sequential build process steps” (Dösinger et al., 2012), which could be interpreted as implying that automated steps are linked together in a chain of sequential stages, but is ultimately too ambiguous to be included in the statement cluster.

3. Methodology

Nowadays, continuous integration systems have shown two major problems. The first one is didn’t consider the factor of cross-platform. The second one is the integration of systems cannot be operated automatically. So developer should install an integration system on each platform. It will cause difficulties on the use and management. Furthermore, developer should test each platform manually. It is waste time and error will easily occur. Table 1 is the market research [6]. I searched the four existing integration systems in the market.

<table>
<thead>
<tr>
<th>Function</th>
<th>CruiseControl</th>
<th>Anthill OS</th>
<th>Gump</th>
<th>Continuum</th>
<th>JCIS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open source</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Free</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>Building Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support cross-platform project building</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Program library dependability</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Building multiple project simultaneously</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Distributed architecture process simultaneously</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Building Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross platform project</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constructing system</td>
<td>Ant, Maven, NAnt</td>
<td>Ant</td>
<td>Ant</td>
<td>Ant, Maven</td>
<td>Ant</td>
</tr>
</tbody>
</table>

There are numbers of continuous integration systems in the market, but most of them are unable support multiple platforms in the same time. Those systems need to install a continuous integration system in each platform to achieve the goal of cross-platform. It seems that we have already integrated the system successfully, but the fact is there still some missing details. So, this paper focus on this two function.
3.1. The process of continuous integration system

According to above analysis, I add two function in the existing integration systems. The first one is design a continuous integration system for cross-platform project. The other one is the result of the integrating systems can be shown in one report. Therefore, the developer can know the new software can run in each platform smoothly or not. Figure 5 is the prototype of the process of continuous integration system [7].

![Figure 5. The process of continuous integration system.](image)

Project manager will work on their own computer to develop, test project. In the early of development, project manager will establish a project on the continuous integration system. Setting timer to construction work and integration work to be performed and integrating environment. As development proceeds, program developers and testers will commit code and test code in the version control system. When the integration is finished, the system results will inform the relevant personnel to build bonded. Here will detail introduce the process of integration.

1. Program developers compile and test on the local computer. Testers drive the new version of the source code from the version control system. In the final we will test program.
2. Program code developers to test through a common version control system project. Testers confirmed that all test cases written in a version control system.
3. Integration Server will take out the latest version of the system source code and test codes periodically to the version control system. Construct the project base on the relevant settings.
4. Integration Server based on the set of the entire project and integrate the designated environment. Integration Server will dispatch job to JobPool.
3.2. Expected Result and Future Work

(1) Continuous Integration can reduce manual test time of developer and tester.
(2) Reduce the probability of errors occurred.
(3) Improve the quality if software systems.
(4) Reduce the cost of software development.

Traditionally, software systems are usually used on the PC platform, but nowadays more and more apps are being used on the Android platform instead of PC. Improve the continuous integration system, allowing it to be connected with the Android and IOS systems.

4. Conclusions

I think the greatest and most wide ranging benefit of Continuous Integration is reduced risk. The trouble with deferred integration is that it's very hard to predict how long it will take to do, and worse it's very hard to see how far you are through the process. The result is that you are putting yourself into a complete blind spot right at one of tensest parts of a project - even if you're one of the rare cases where you aren't already late. Continuous Integration completely finesses this problem. There's no long integration, you completely eliminate the blind spot. At all times you know where you are, what works, what doesn't, the outstanding bugs you have in your system. Continuous Integrations doesn't get rid of bugs, but it does make them dramatically easier to find and remove. As a result projects with Continuous Integration tend to have dramatically less bugs, both in production and in process. However, it takes a while before a team really gets to the low level of bugs that they have the potential to reach. Getting there means constantly working on and improving the tests. This helps break down the barriers between customers and development barriers which I believe are the biggest barriers to successful software development.

5. Acknowledgement

This study is the final report of E-Enterprise Integration and supported by Professor Amy J.C. Trappey.
References