JP Morgan Chase & Co.
Continuous Integration Project

Major Qualifying Project Report

Submitted By:

__________________________________________  ________________
Yang Yang                                    Yao Li

Advisors:

__________________________________________  ________________
Arthur Gerstenfeld                           Xinming Huang

Project Center:

Wall St. New York, NY
B term 2013

Sponsor:

JPMorgan Chase & Co.

Jan 18, 2014
Abstract

We collaborated with Global Real Asset team and Alternative Investment team at JPMorgan to implement a continuous integration platform for software development. The process utilized an open source tool to streamline the automated build process accompanied with unit and regression suite testing to mitigate risk. This platform helped reduce time and effort to initiate the regression testing in the development cycle.
Acknowledgements

The MQP team would like to thank their primary liaisons at JPMorgan Chase & Co.: Sri Atluri, Scott Burton and Chris Rice for their constant support. The team is also grateful for the guidance from their mentor Charles Dixon. Additionally, the team would like to thank their advisors: Prof. Arthur Gersenfeld and Prof. Xinming Huang for their help and assistance during the project in New York City. At last but not least, the team would like to thank the entire JPMorgan Asset Management Technology Team for granting them full access to the system, and helping them solve technical issues.
Authorship

The Major Qualifying Project was completed through collaboration between both team members. Each team member worked diligently on the project and the final report. The Introduction, Background and Methodology were completed by both members. Yao Li was mainly responsible for the Result and Analysis section, and Yang Yang was responsible for the Reflection section.

Each team member contributed equally to the report and they both dedicated a significant amount of time of effort.
# Table of Content

Abstract.............................................................................................................................2  
Acknowledgements .............................................................................................................3  
Authorship ..........................................................................................................................4  
Table of Content ................................................................................................................5  
Table of Figures ..................................................................................................................6  
Table of Tables ....................................................................................................................6  
Chapter 1 Introduction .......................................................................................................7  
Chapter 2 Background ......................................................................................................9  
  2.1 Sponsor Description ....................................................................................................9  
  2.2 Software Development Methodology .......................................................................10  
    2.2.1 Waterfall Model .................................................................................................10  
    2.2.2 Agile Methodology .............................................................................................12  
    2.2.3 Continuous Integration ......................................................................................14  
    2.2.4 Refactoring Continuous Integration at JPMorgan .............................................15  
  2.3 Tools and Data Format .............................................................................................17  
    2.3.1 Jenkins ...............................................................................................................17  
    2.3.2 Nexus ...............................................................................................................19  
    2.3.3 Sonar ...............................................................................................................20  
    2.3.4 Subversion and TortoiseSVN ...........................................................................21  
    2.3.5 XML ...............................................................................................................24  
Chapter 3 Methodology ...................................................................................................25  
  3.1 Task List ...................................................................................................................25  
  3.2 Survey .......................................................................................................................26  
  3.3 Automated Release Management ............................................................................26  
  3.4 Connection of Jenkins and ARM ............................................................................29  
  3.5 Connection of Machine and ARM ..........................................................................31  
  3.6 Deployment Plan Creation and Modification .............................................................32  
Chapter 4 Result and analysis ..........................................................................................34  
Chapter 5 Future Plan ......................................................................................................36  
References ..........................................................................................................................37  
Appendix A- Deployment Plan (All actions included) .......................................................38  
Reflection on the Project ....................................................................................................39
Table of Figures

FIGURE 1  Waterfall Model .................................................................................................................. 10
FIGURE 2  Agile Lifecycle ...................................................................................................................... 12
FIGURE 3  Demonstrating Timeline for Agile and Waterfall Model ..................................................... 13
FIGURE 4  Continuous Integration Model .................................................................................................. 15
FIGURE 5  Continuous Integration Model in Industry ............................................................................. 15
FIGURE 6  Refactor Continuous Integration at JPMorgan ........................................................................ 17
FIGURE 7  Jenkins Console .................................................................................................................... 18
FIGURE 8  Configurations of Application .............................................................................................. 19
FIGURE 9  Sonatype Nexus .................................................................................................................... 20
FIGURE 10  Sonar Dashboard ................................................................................................................... 21
FIGURE 11  Subversion and TortoiseSVN ............................................................................................... 22
FIGURE 12  TortoiseSVN ........................................................................................................................ 23
FIGURE 13  Commit Changes (JIRA Ticket) ............................................................................................ 23
FIGURE 14  Update ............................................................................................................................... 24
FIGURE 15  ARM Main Page ................................................................................................................... 27
FIGURE 16  Detailed Continuous Integration ......................................................................................... 28
FIGURE 17  Corresponding Software ..................................................................................................... 29
FIGURE 18  Jenkins Post Build Section .................................................................................................... 29
FIGURE 19  ARM Main Page ................................................................................................................... 30
FIGURE 20  Flow Controller Mechanisms ............................................................................................. 31
FIGURE 21  ARM Main Page ................................................................................................................... 31
FIGURE 22  Deployment Plan .................................................................................................................. 32
FIGURE 23  Refactoring Continuous Integration Process ....................................................................... 34
FIGURE 24  New Continuous Integration Model .................................................................................... 34

Table of Tables

TABLE 1  Task List ................................................................................................................................. 25
TABLE 2  Survey ................................................................................................................................. 26
Chapter 1 Introduction

A software development methodology is a framework that is used to structure, plan and control the process of developing an information system. Some of the most common methodologies are Waterfall and Agile.

Waterfall is a traditional software development methodology. It contains the following steps: requirements, design, implementation, verification and release. It is a linear sequential process, which means it is simple and easy to understand and use. Waterfall especially works well for small and relatively simple projects. However in many cases, it can be a bad practice due to its inflexibility to changes.

Agile software development is a good solution to the drawbacks of Waterfall. It is an iterative and incremental process. Thus Agile is flexible to changes. The technology division of JPMorgan Chase & Co. focuses on Agile methodology for their financial software development. They created a continuous integration platform based on Agile, which allows developers to integrate code changes frequently in order to build up final products more efficiently.

This already existed platform automatically built up software based on requirements. However, quality assurance testing would still have to be done manually. This was not an ideal situation as manual work meant waste of time. In order to include quality assurance in the entire automated lifecycle, the MQP team was assigned to refactor the continuous integration platform.

The team utilized various continuous integration tools including Jenkins, Sonar and Subversion to bridge the automated development lifecycle and quality assurance. The new continuous integration platform helps initiate testing within quality assurance automatically after development.
The team also set the following five objectives to achieve through this project:

- Identify and understand the goal of the Asset Management Technology Team of JP Morgan Chase & Co.
- Conduct research on software development process as well as quality assurance process
- Test the tools and scripts to eliminate errors and solve issues
- Optimize the continuous integration process
- Provide recommendation on future improvement
Chapter 2 Background

2.1 Sponsor Description

JPMorgan Chase & Co. is an American multinational banking and financial services holding company. It is one of the oldest financial institutions in the United States. With total assets of $2.509 trillion, it is the largest bank in the United States. JPMorgan Chase & Co. operates in over 60 countries, serving consumers in the United States and many of the world’s most prominent companies, institutions and government clients. It is a leader in investment banking, financial services for consumers, small business and commercial banking, financial transaction processing, asset management and private equity. JPMorgan Chase & Co. is one of the Big Four banks of the United States. According to a composite ranking by Forbes magazine in 2013, JPMorgan Chase & Co. is the world’s third largest public company.

JPMorgan Chase & Co. is built on the foundation of more than 1,000 predecessor institutions, including Chase Manhattan Bank, JPMorgan Chase & Co., Bank One, Bear Stearns and Washington Mutual. Originally established as the Manhattan Company in 1799, JPMorgan Chase & Co. has over 200 years of banking service experiences. In 2000, Chase Manhattan purchased J.P. Morgan & Co for $34 billion and the company was renamed JPMorgan Chase & Co.. It then acquired Chicago-based Bank one in 2004 and the assets of Washington Mutual and Bear Sterns in 2008. Now JPMorgan Chase & Co. is considered to be a universal bank.

JPMorgan Chase & Co. conducts businesses through two brands, Chase and J.P. Morgan. Chase provides consumer and commercial banking service. Its consumer businesses include but only limited to Credit Cards, Small Business, Education Finance, Retire and Investment and Merchant Service. Its commercial banking
businesses include Middle Market Banking, Business Credit, Equipment Finance, Commercial Term Lending and Community Development Banking. The J.P. Morgan brand conducts businesses including Asset Management, Investment Banking, Private Banking, Treasury and Securities Services, and Commercial banking. J.P. Morgan’s broad global platform and strength enable it to create long-lasting value for clients.

The MQP team worked at the Asset Management division of JPMorgan Chase & Co.. This division is a leading asset manager for individuals, advisors and institutions. It is one of the largest asset and wealth managers in the world, with assets under management of $1.5 trillion (as of September 30, 2013). It provides global market insights and a range of investment capabilities that few other firms can match.

2.2 Software Development Methodology

2.2.1 Waterfall Model

![Waterfall Model Diagram](Figure 1 Waterfall Model)
Waterfall is a classic approach to software development. As seen in Figure 1, it is a linear sequential process, starting from gathering requirements and ending with the release of software. Waterfall model is simple and easy to understand and use. Before one starts to develop the software, he needs to gather all the requirements from the clients. After he has all the requirements, he will start to design the software. Then he will build up the design, test the product and release it.

Waterfall development process is a sequential approach to design and development, assuming the requirement must be identified before starting the process. Each phase must be 100% completed before the next phase starts. The product will be delivered at the end of development process. However, when the requirement changes during the term of the process, it would be difficult to turn back in the process. The entire process would have to be restarted, which is time-consuming and expensive.

**Advantage:**

- Waterfall is easy to understand and implement
- It works well for small and relatively simple projects

**Disadvantage:**

- It is inflexible to changes. When there is any change in requirements, the entire process has to be restarted
- Testing is relatively late, which makes it difficult to fix bugs
- Clients are not often involved. Voice of customer is weak
- It is not a good model for complex project
2.2.2 Agile Methodology

2.2.2.1 Introduction to Agile

Agile software development is a solution to the disadvantages of Waterfall model. It is also the basic idea behind continuous integration. Unlike the linear sequential waterfall model, Agile is based on an iterative and incremental process.

![Agile Lifecycle Diagram]

As seen in Figure 2, it contains the following steps: Development, Build, Testing and Deployment, which means developing the code, building up the product, testing the product and deploying it (an iteration). Agile does not need to gather all the requirements before starting to build the product. It can start from a few requirements, and keep integrating clients’ changes in requirement later. Each iteration in Agile contains a Testing phase, which helps detect bugs easily and early. In each iteration process, the direction of the design can be changed base on the requirements of the clients so that it is more efficient to meet the expectations of clients. Thus Agile is more optimized.
Advantages:

- Agile is flexible to changes
- Bugs are detected and fixed more easily
- Clients are involved
- It works well for complex projects

Agile methodology can respond to change of clients’ requirements efficiently. In many cases, it is a good alternative to traditional waterfall development process.

2.2.2.2 Comparison between Agile and Waterfall

As shown in Figure 3, Agile is less time consuming. Its iterative and incremental characteristic makes it flexible to changes and helps improve risk management. It is a good practice in software development.
2.2.2.3 Development History of Agile

Tracing back to 1957, E. A. Edmonds introduced an adaptive software development process, at the same time New York Telephone Company's Systems Development Center developed same method. In the early 1970s, Tom Gilb published the Evolutionary Project Management (EVO), which is the predecessor of Competitive Engineering. Half century later, Gielan came out with Incremental software development methods. Agile software development methods evolved in the mid-1990s as a reaction against the waterfall-oriented methods, which were characterized by their critics as being heavily regulated, regimented, micromanaged and overly incremental approaches to development. In 2001 software developers published the Manifesto for Agile Software Development to define the approach agile software development. Some of the manifesto's authors formed the Agile Alliance, a nonprofit organization that promotes software development according to the manifesto's principles.

2.2.3 Continuous Integration

Continuous integration is a practice of agile in software development and it makes agile more efficient by automation. It allows developers to integrate code changes frequently, to be more efficient in building up the final products. The entire process of continuous integration is very similar to those in Agile, which contains Development, Build, Test and Deploy. The only difference is continuous integration automates its process.
Whenever there is a change in requirement, it will be integrated immediately through automated Build, Test and Deploy. This saves the time cost by manual intervene, because developers do not have to communicate with those who would trigger the following steps.

2.2.4 Refactoring Continuous Integration at JPMorgan

Figure 5 shows the continuous integration platform JPMorgan created in the past. It contains two major parts: Development lifecycle and Test Environment. Development lifecycle includes Development, Build, Test and Deploy. These four
steps make up the automated development lifecycle. Test Environment, in another word, Quality Assurance, contains regression testing, functional testing and other testing which make sure the final product meets the requirements and is free of defects. Functional testing is used to test whether the code functions as expected. Regression testing is used to test unchanged code and make sure the changed code does not introduce new fault. It determines whether the previous functions works normally with the new change.

Every time when there is a change in requirement, developers will make changes on their own code copy and commit changes back to code library. The automated Build, Unit Test and Deploy will then start. After a product goes through Unit Test, it will be deployed to Test Environment in a non-automated manner, which is time consuming.

In order to optimize this continuous integration platform, the team worked to reconstruct and revamp the platform by merging regression testing (part of Quality Assurance) to the automated software development lifecycle. As shown in Figure 6, Test Environment and Development lifecycle are merged and becomes one automated process.
2.3 Tools and Data Format

2.3.1 Jenkins

Jenkins is the leading open-source continuous integration server. It is the main continuous integration tool in this project. It was created in 2004 in Sun Microsystems and was first released in java.net in 2005. Built with Java, Jenkins provides 824 plugins to support building and testing virtually any project. Jenkins helps build/test software projects continuously and monitoring executions of externally-run jobs. Jenkins has several features including easy installation/configuration, permanent links, after-the-fact tagging, distributed builds, plugin support etc. Current Jenkins focuses on the following two jobs:

- Building/testing software projects continuously. Jenkins provides an easy-to-use so-called continuous integration system, making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. The automated, continuous build increases the productivity.
• Monitoring executions of externally-run jobs. Jenkins keeps outputs of externally-run jobs and makes it easy for you to notice when something is wrong.

Figure 7 shows the Jenkins console, and a list of GRA (Global Real Asset) modules within Jenkins. GRA is one of the applications the team worked on. The other one is the Alternative Investment.

![Jenkins Console](image)

**Figure 7 Jenkins Console**

One plug-in of Jenkins is Maven. It is a software project management and comprehension tool. Maven is based on the concept of a project object model (POM), which contains specifications of the project. It can manage a project's build, reporting and documentation from a central place. Maven lifecycle includes clean, build, unit test, package and deploy.
2.3.2 Nexus

Sonatype Nexus (Figure 9) sets the standard for repository management providing development teams with the ability to proxy remote repositories and share software artifacts. With Nexus, developers can have control over open source consumption and internal collaboration.
2.3.3 Sonar

Sonar (Figure 10) is an open source software quality platform and it uses various static code analysis tools to extract software metrics.

JPMorgan Asset Management Technology Team use Sonar, a separate product, to monitor code quality. They also use Sonar plug-ins to display quality data.
on Greenhopper (which can manage Agile project) dashboard.

![Sonar Dashboard](image)

**Figure 10 Sonar Dashboard**

### 2.3.4 Subversion and TortoiseSVN

Subversion (SVN) is a version control system that manages files and directories over time and in a central repository.

![Subversion Web and Application Structure](image)
TortoiseSVN is a shell extension as well as a client for subversion. While Subversion is the central repository containing the most updated code of development team. TortoiseSVN is able to synchronize with Subversion, so that each developer has his own local copy of the source code on his machines. The relationship between Subversion and TortoiseSVN is shown in Figure 11.

![Figure 11 Subversion and TortoiseSVN](image)

The developers may use web browser to check out code from subversion.
After the developer made changes to the code, they will commit change to subversion by creating a JIRA ticket with special name indicating what changes they have made. The ticket will be used to record and memorize the changes, so in the future the developers are capable of finding the changes they made and even revert them. Figure 13 shows the commit dialog of TortoiseSVN.

Figure 12 TortoiseSVN

Figure 13 Commit Changes (JIRA ticket)
Since every developer updates the file before he makes new changes every day, he will get the latest code before development process proceeds again. Thus the function of TortoiseSVN makes it easier for different developers to make changes to mainline code without having integration problem in the end.

2.3.5 XML

XML (Extensible Markup Language) is the main coding language for this continuous integration project at JPMorgan.

It is a markup language which sets rules for encrypting document in a special format so that it can be read by both users and machine. It uses tags to describe the content of a document. The language can be defined by user so that it is extendable and unlimited. Presently, XML allows data sharing through Internet, computers and applications which make it distinct among all other languages.
Chapter 3 Methodology

3.1 Task List

Continuous Integration On-boarding Task

<table>
<thead>
<tr>
<th>Steps</th>
<th>Tasks</th>
<th>Our Time</th>
<th>Elapsed Time</th>
<th>Completion Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Survey/project characteristics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Review Jenkins project and nexus info</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Request ARM node on target machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set schedule</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Request function ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Test get-package script</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Load groovy and get script on target machine(s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Create Install/deploy script</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Test get &amp; install script</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Test calling get-package and install script from ARM console</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test calling get-package and install script with Scheduler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Test Jenkins build and ARM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reset Scheduler to live run</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Turn over to QA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Post implementation survey</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Task List

At the beginning of the project, the team created a task list (Table 1) in order to plan and track future work progress. It contains twelve steps required for the project as well as elapsed time and completion status.

Based on the task list, the team first designed a survey (Table 2) to gather information from the development team. Then the team would review Jenkins project and Nexus information, request ARM node on target machine, set schedule and request function ID. When these steps were done, the team would create script with specific functionality for the project. Next, the team would link various web applications and machine together. Finally, the team would test the result and turn it over to Quality Assurance. The team also created a post implementation survey at the end of the project.
3.2 Survey

The survey as shown in Table 2, helped request information for the project, such as server address of target machine, artifact type, installation time, recycle frequency and specific configuration from the developers. This information provided the team a detailed understanding of software development environment at JPMorgan. The team interviewed the Global Real Asset and the Alternative Investment development teams based on the survey and summarized the results.

<table>
<thead>
<tr>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which Jenkins environment supports your project?</td>
<td></td>
</tr>
<tr>
<td>What are your target machines? How many machines do you need in your project?</td>
<td></td>
</tr>
<tr>
<td>What is your test environment and how is it like?</td>
<td></td>
</tr>
<tr>
<td>What types of artifacts do you use? Do they have any dependency? How to deploy them?</td>
<td></td>
</tr>
<tr>
<td>Do you have scripts for deployment on the target machine?</td>
<td></td>
</tr>
<tr>
<td>Who to contact if there are problems with script deployment?</td>
<td></td>
</tr>
<tr>
<td>Does it install automatically or need to be installed manually?</td>
<td></td>
</tr>
<tr>
<td>How long does it take to install?</td>
<td></td>
</tr>
<tr>
<td>How often do you recycle your machine?</td>
<td></td>
</tr>
<tr>
<td>How many people can access the machine and have update right?</td>
<td></td>
</tr>
<tr>
<td>Is the machine used for other purposes or by other teams?</td>
<td></td>
</tr>
<tr>
<td>Is there any special setup/configuration/customization for installation?</td>
<td></td>
</tr>
<tr>
<td>What do you expect in the project?</td>
<td></td>
</tr>
<tr>
<td>What do you think of the survey? Does it include what you expect for the project?</td>
<td>Table 2 Survey</td>
</tr>
</tbody>
</table>

3.3 Automated Release Management

JPMorgan utilizes a web application named Automated Release Management (ARM), which served as a communicator to trigger the deployment on machines. It saves time spent on manual deployments and will increase efficiency of the development lifecycle eventually. Automated Release Management (ARM) is a lightweight communication protocol linked to machine and initiates desired functionality on machine automatically. Below is the snapshot of ARM.
Figure 16 shows the detailed reconstructed continuous integration model at JPMorgan. Whenever the developers want to make changes to the code, they check out code from Subversion, which is a code repository containing the most updated code. They then make changes on their own code copy and commit changes back to Subversion using JIRA ticket. The continuous integration tool Jenkins scans the work area every 15 minutes and looks for changes in code. When it detects a change, it will start the development process automatically. This process includes clean, build, test, package and deploy (Maven Lifecycle). After an artifact is built and packaged, it will be stored in Nexus, which functions as an artifact repository. In the post-build section of Jenkins, the ARM will automatically initiate deployment by pulling artifact from Nexus and have it deployed on target machine. Regression testing will also be performed on target machine.
Figure 16 Detailed Continuous Integration

Figure 17 lists development lifecycle and the corresponding tools and web applications. The team focused on Jenkins, Nexus, Target machines and ARM for the platform. After Jenkins builds up artifacts, they are sent to be stored in Nexus. Target machine will bring the artifacts from Nexus and start the deployment triggered by ARM, which is a plugin of Jenkins.
3.4 Connection of Jenkins and ARM

In post-build section of Jenkins, there is a plugin called ARM plugin, which does the link between Jenkins and ARM. This process enables development teams to have their artifacts deployed on to the target machines automatically. In other words, Jenkins builds the project, and ARM triggers deployment on machines. This automation saves a lot time that was normally spent on manual builds and deployments.
Figure 18 and 19 show the ARM plugin in Jenkins post build section and ARM web application respectively.

First, the team needed to enter specific Job Identifier, which is a piece of translated information of a numeric value from ARM web application (Tab out of the Job ID field and the plugin will pull out the Job information from ARM), as well as Job Option. Job ID tells ARM what specific functionality the developers want it to perform. In this project, the Flow Controller is a job from ARM, which primarily controls flow of different actions; Job Option provides the web application with the information needed for this specific job. In this project, it is the URL of deployment plan (http://subversion.ny.jpmorgan.com/svn/repos/IMA-IMA-GRA/GRA-Dispositions/branches/GRA-Dispositions-1.1/GRA-Dispositions-CI-ARM-SIT-Deploymentplan.xml), and the encoded password (sshPassword) for special password protected server. After the SAVE button is clicked, Jenkins will pass Job Description to ARM. Thus Jenkins job is integrated with an ARM job. When Jenkins finishes the build, ARM will trigger the deployment on machine automatically.
Figure 20 Flow Controller Mechanisms

Figure 20 shows the mechanism of ARM’s job in this project, Flow Controller. Flow controller defines the flow of the work. It reads the XML, constructs the syntax and calls other appropriate controllers to do all the work. For instance, it calls the FASTLEGACY controller (version 301) to do different commands such as stop, deploy, undeploy and start. Additionally, it can call shell script to perform regression test.

3.5 Connection of Machine and ARM

```
<table>
<thead>
<tr>
<th>WorkFlow</th>
<th>Options</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABB deployment using ARM node: false</td>
<td>-1 Job Description -&gt; -enablePassword &lt;&gt;</td>
<td>SIT</td>
</tr>
<tr>
<td>Release Label: NA</td>
<td>Log level: INFO</td>
<td></td>
</tr>
<tr>
<td>Keepgoing: false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Matched nodes: 

Thread Count: 1
Keep going on error: false
```

Figure 21 ARM Main Page
After the team connected Jenkins and ARM together, they focused on the connection between target machine and ARM. In ARM, there is a section called Matched Nodes. It tells ARM which specific machine to communicate with. As seen from Figure 21, the piece of string in the Match Nodes is the server address. That is how target machine is connected to ARM, and where ARM can start the deployment.

### 3.6 Deployment Plan Creation and Modification

![Deployment Plan](image)

The team was assigned to create Deploymentplan.xml for over 20 applications of two different teams (Global Real Asset Team and Alternative Investment Team). As mentioned in the previous section, Deploymentplan.xml is an XML file that ARM will call to perform Flow Controller job, and target machines will follow each steps on XML to do all the jobs. The Deploymentplan.xml contains different actions including stop, deploy, undeploy and start. Figure 22 shows a piece of Deploymentplan.xml, containing only the deploy action. The team worked on modifying tag <deploymentPlan> to make sure it points to correct deployment file in Subversion. They also modified tag <war> to get accurate address from Nexus for
target machines to pull out corresponding artifacts. Also, the team located correct machine address, which is shown in tag <hostURL>, and verified some of basic information such as application name, application type with POM file of 20 applications.
Chapter 4 Result and analysis

The team successfully refactored the continuous integration platform by linking up different tools (Figure 23). At the end of the project, ARM was able to deploy artifacts automatically on machine instead of manual work. If the team follow the same steps and change controller (FASTLEGACY) in ARM, they would be capable of performing regression testing (Part of Testing Environment) on machine automatically as well. Instead of the manual work in between the Development Lifecycle and the regression testing, the entire process is fully automated (Figure 24).
This new continuous integration platform helps save time and improve process efficiency. It is currently being in use in the technology division of JPMorgan.
Chapter 5 Future Plan

The Asset Management Technology Team at JPMorgan will keep on merging regression testing in the automated lifecycle. In the future, they will work on changing ARM controller from FASTLEGACY to Shell Script (Figure 20) and send regression test script to this Shell Script controller in order to initiate regression test on machines. Other than merging regression test suite of quality assurance into automated development lifecycle, JPMorgan will continue to include other parts of quality assurance such as functional testing into the automated cycle and try to achieve the target of making the entire software development process more efficient and productive.
References


Appendix A- Deployment Plan (All actions included)
Reflection on the Project

Design component

During the Major Qualifying Project at JPMorgan, we designed a continuous integration platform for the development of financial applications and software. This platform allows software developers to integrate their code changes frequently, to be more efficient in building up the software. It contains two major parts, which are development lifecycle and test environment. Each part is fully automated and they are linked together in an automated manner.

Constraints and alternatives

Time is one major constraint during this project. Our team needed to build up continuous integration platforms for 22 financial applications in 7 weeks. Both of us had little knowledge in continuous integration and XML (the main coding language for the platform) before we started the project. We spent a lot of time learning and discussing with our colleagues. We also made specific work plans for each day. We are glad we finished the project successfully.

Need for life-long learning

This project was a wonderful learning experience and we both enjoyed it very much. We not only gained technical skills in continuous integration and XML coding, but also learnt how to work, talk and behave professionally in a top financial firm. By presenting in front of the management and professors, we gained public speaking skills. Teamwork was very important in this project. We have helped each other learn and grow during the entire project. This project was also a great platform for us to apply critical thinking.

We believe what we learnt and gained from this experience will be very helpful for our future study and work.