Implementing Continuous Integration towards Rapid Application Development

Fazreil Amreen Abdul
Software Development Lab, Application Process
MIMOS Berhad
Kuala Lumpur, Malaysia
fazreil.jalil@mimos.my

Mensely Cheah Siow Fhang
Corporate Quality, Process Quality
MIMOS Berhad
Kuala Lumpur, Malaysia
sf.cheah@mimos.my

***Abstract* - If one is working in isolation, Continuous Integration may not be good for him or her. However, not many of us have the lavishness of working alone in software development. Most software development are done in a team, leveraging on diverse functional groups delivering different modules or subsystem. In an enterprise where development of software involves a collection of developers working on modules, integration management is absolutely a necessity; we need to find ways to work efficiently and effectively to make the long and heavy integration process to a simpler and joyful task. The value of an integrated, streamlined build process is something that any software engineers would immediately recognize, all this needs lead us to the philosophy of Continuous Integration. It is the intent of this paper to illustrate a journey and learning process in setting up a Continuous Integration for a software group.**

Keywords - Continuous Integration; Software Configuration Management; Build Process

I. CONTINUOUS INTEGRATION

It's difficult to tag a definition to Continuous Integration as the concept always evolved, and it should be. Continuous Integration may sound as simple as automated builds with validation, to as complex as covering all the daily tasks of a typical Software Configuration Manager and further extend it to become a measurement of progress, providing the team with invaluable feedback of the health state of the code VOB and objects.

Traditional project management is always a stereotype well known to project managers, not exclusive to software development projects. Previously managing projects mean to keep track which of the module has been completed so that it could be integrated to the product. Continuous Integration in another hand let integration at an early stage.The modules are integrated even they are not even finished. It will undergo development and they will keep integrating rapidly leading to multiple integrations daily [1].

In an article by Martin Fowler in his website, Fowler had written, "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 to detect integration errors as quickly as possible." [1].

Ultimately, the objective of Continuous Integration is to capitalize on developer productivity while reduce integration risk by making integration an easy, natural part of the software engineering cycle.

II. IMPLEMENTING CONTINUOUS INTEGRATION

Implementing Continuous Integration is not easy for a beginner. It requires a lot of thoughtful setup and careful selection of tools. There are two areas that are worth concerning, first are to get the core functionalities in place and second are to get the supporting functionalities running along the system. We need an engine that acts as the central build machine, that collects all the latest check in from the code repository and compile them as a union of integrated code, through its own validation mechanism and reporting system, ensuring the collection of work is accessible anytime and of its best latest build.

First, the integration engineer or SCM will need to define the build strategy, which is best suit the development environment. This process has to be agreed by the project manager. Secondly the engineer must select the best tools to aid the process of developing a build script. New releases of IDE usually include the functionality in developing a build script, the script is used in specifying build procedure, making use of either ANT build script, MAVEN script, batch files or command line terminal, etc. Then, we may extend the feature of Continuous Integration covering static analysis, testing, bug tracking, deployment and etc. Build Server like HUDSON or JENKINS are molded to support multiple plug-ins. This makes them the machine that orchestrates the various tools to complete a build cycle. The availability of plug-ins and automation brought in by the gadgets makes the build server a perfect tool to sit in the center of Continuous

Integration. An organized Continuous Integration cycle will surely make development job more efficient. Developers can now totally engage to development and coding tasks while the rest of the engineering process is being handled automatically.

Automation in building software and deployment need a careful setup and configuration. Each part of the process is preferred to cater for flexible / changeable input and able to link up with each other. For example, we would like to write a deployment script that could be easily set to the desired server by switching a few variables, this would speed up the transition from a configuration to another with minimal change. While writing the script to define the process, one must keep in mind that the script written must be easily plug in and out of the Continuous Integration process with minor changes needed, in which the design of the sub process shall be loosely coupled on one another.

Creating a module to be loosely coupled is advantageous. These modules are like building bricks, it gives us freedom to construct the flow of each process. Coupling some of the process into one is advisable only if the process is dependent to each other. An example of why we do this is like there are times that we must substitute some configuration files (E.g. properties file, input files) to suit the build environment or the goal of the build targeted. For instances, different build type may require different license files, Window and Linux build require different set of files to be included for the licensing of the software to work.

III. PROPOSED CONTINUOUS INTEGRATION BEST PRACTICES

In some organizations, a rapid application development is ensured when Continuous Integration takes place, allowing faster delivery and quality product. Continuous Integration is realized through a set of engineering tools covering bug tracking, version control system, testing tool and deployment web application servers, etc to complete the Continuous Integration effort.

Continuous Integration consists of various engineering processes in nature. Continuous Integration must be made as light as possible to prevent human mistake and not to discourage its usage. In some organizations, automation is implemented in resource gathering, compiling the build and software packaging as shown as below:

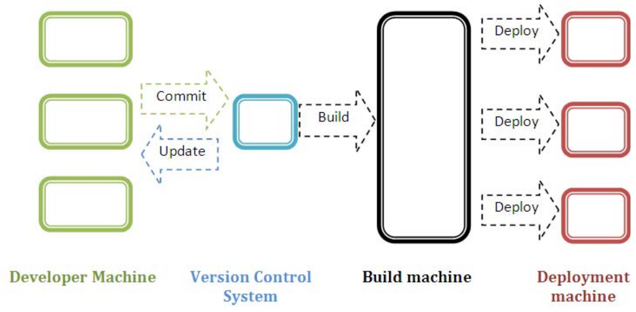


Figure 1. Illustration of Continuous Integration Model

To gather resources is to identify the resources that hold the configuration items of a software project. In most organization these configuration items are stored in repositories. The process is simple if the items are stored at one place. However as the organization grows, the configuration items of a project may be stored in a different repositories. Some projects may need to retrieve files from different version control systems. Good Continuous Integration practice shall collect all source codes into a workspace so that it causes less hassle during the build process.

Compiling build is one of the core activities of Continuous Integration. It is the most crucial part and also the most challenging among all phases. There are various ways of compiling the build. The most used method is to compile a project using the predefined steps in the IDE. This seems to be very convenient since we are doing it on the graphical interface, however, this may not achievable through build server (if IDE is not integrated). The build server normally accepts batch of commands, most often than not, the build manager or SCM will script and automate the build to make the build process dynamic. Dynamic in this context is referring to the ability to place more process throughout the entire compilation. The whole compilation process may include the compilation of the codes by its own compiler, obfuscation, unit testing, and dynamically configuring the build. A build like this could be achieved by using Ant, Maven or MSbuild.

Packaging is usually the final step of the whole process. Packaging involves activities such as making a product which is completely distributable to other parties. The product can be an executable, an archive file, or a report of a build. In most cases, we would want an executable to be presented in the end of a build. The package completes the goal of Continuous Integration because we are able to successfully gather all our resources and compile them. While the packaging concept may seem trivial, packaging software should not be taken lightly.

Compatibility may be one of the parameter constitute to the packaging process. For example, a 32-bit compilation versus a compilation meant for 64-bit; a build configured to deploy on a Linux machine versus a build meant for Window machine. To avoid the compatibility issues, different build should be triggered for separate configuration, the drawback of these individual build is server space are required to fit more builds in the build server and it would require more time to complete all the builds before we could fetch the package. The other alternative is to have a common resource gatherer with switches provided during the build to let the builder know which configuration file or parameters to be referred in order to complete a build. For example, after a build for Linux is done, the item provided for Linux build is taken off from the package in order to make way for the Window build. The second method is more efficient but it requires careful scripting and coordination of build so that the build does not produce a package which is not optimized for what it was built for.

In conventional method of packaging, an executable is usually sufficient as the customer deliverables, but in enterprise level today, a lot of other items are expected to make up the entire software package, which may includes license keys, supporting documents or configuration files. This can be seen in open source projects where weekly build is automated with its required libraries, readme and other supporting items. Level and type of distribution may differ based on its recipient parties, its usage and its target environment. Thus, we shall start considering packaging even during the Continuous Integration effort.

IV. Experience in Applying Continuous Integration

A. *Education*

Educating engineering community in accepting Continuous Integration may not be easy, regardless of whether the direction is coming from top-down (management-driven) or from developers' initiative. In both cases, there are pro's and con's associated as illustrated below:

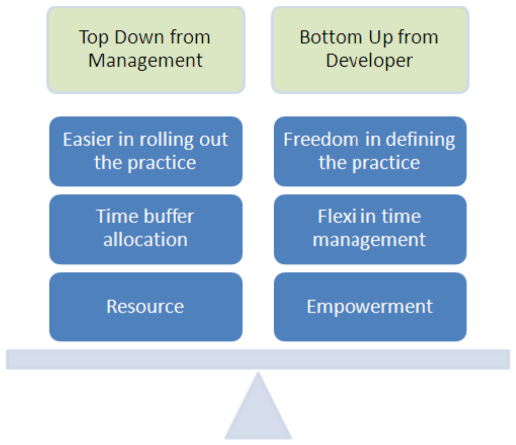


Figure 2. Comparison on the Education Approach

If the Continuous Integration effort comes from top management, the benefit that it brings is all the resource make easy as the top management is supportive of the effort. Project managers have to bear with the effort and servers needed to apply Continuous Integration in the project development. Project managers and engineers are also learning to look at the Continuous Integration seriously and play their part in it. The disadvantage about this scenario is that top management would want the Continuous Integration put in place as soon as possible, resulting less time allocated to implement extra functionalities into the Continuous Integration, it will become an issues when Continuous Integration is being introduced only at the later stage of development.

In the other case, if the Continuous Integration is triggered by developers' initiative, there will be more time buffered for developers to study the tools available in the Continuous Integration setup. The developers could work together in integrating their work in one unified build process put up piece by pieces. In this case, development team has to show case the success stories quantitatively to educate the top management on how Continuous Integration has brought benefit to the project development cycle.

B. *Project Setup*

In some cases, where a project may be rushed into completion in a very short timeframe, build plan may be established only in late development cycle catering only for a specific project needs, which is made less modular, less flexible and not able to cater for reuse later. In this type of short-term setup, it involves less plug-ins, and concentrates mainly on the compilation, excluding good-to-have features such as on testing, reporting or custom notification which takes extra effort and is good for long term setup.

A long term setup is what every configuration manger should aim for. Setting up a build in early stage help to promote flexibility in build setup to cater for long term usage. In this context, flexible means the build is able to accept interchangeable settings with only minimal changes required when it is put in different environment.

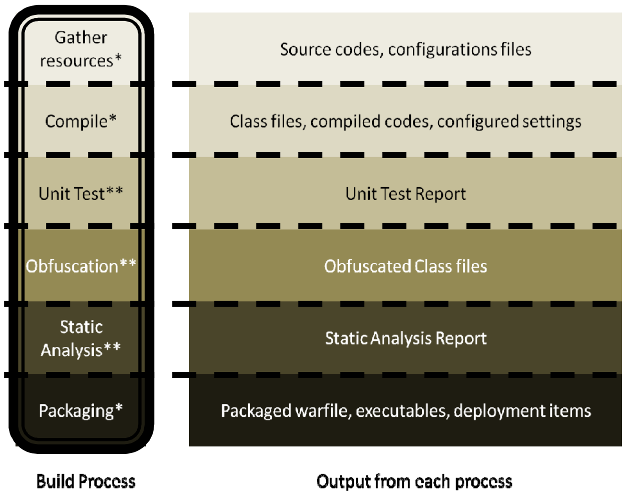
Another benefit brings by long term setup modularization, each step is defined one by one and it is possible to drop a step or rearrange the steps easily. Setup like this makes the setup flexible and adaptable for reuse in the future. Implementing long term setup may increase maintenance as the setup may be affected by change of tools, change of hardware, etc, but the benefits that it brings is worthwhile.

A new project should be setup in Continuous Integration environment in the early stage of the project development. The best time to start the build setup is when the system architecture has been implemented. Preparing build environment in parallel with the software coding could be advantageous. In coding phase, the code will go through changes that would most likely define how the code shall be built. It is not necessary for the build to be finalized at this time but it will help to plan a build that is aligned with the coding practice and fitted both build manager and developers. The developer is not expected to add in extra codes to fit the build configuration but they should add in keywords or annotations that may be needed by the build. For example, regular annotation like 'TODO' or copyright comments may be useful to produce good and complete source code.

C. *Ingredients in Continuous Integration*

Continuous Integration by definition consists of all three processes that are mentioned before; gathering resources, compiling, and packaging. But what makes Continuous Integration essential to be part of the software engineering process is that we could add in more functionality into the implementation. While the original concept is to mimic what an engineer would do in building a product, we could use this opportunity to let the build server perform more actions

This paper separated core functionalities and supporting functionalities in Continuous Integration. Core functionalities are those that perform the gathering of resources, compilation and packaging, whilst the other 'add-in' features is categorized as supporting functionalities, as shown below:



*Core functionalities
**Supporting functionalities

Figure 3. Illustration of Continuous Integration Model

The core functionalities consist of three parts that complete the basic cycle of Continuous Integration. They are version control system to perform the gathering of resources, build system that performs compilation of source codes which execute predefined task included in the build instruction of the project and the deployment management which take care of the packaging and deliverable. An automated issue tracking mechanism may be implemented to serve as the medium for the build system to communicate the build status to the project team.

Supporting functionalities of the Continuous Integration includes code analysis, obfuscation and testing done on the code. Code analysis will help in producing a clean code tree (do not have duplicate and well-written, efficient code [2]). A good code analysis will provide suggestions and highlight which part of the code should be improved. The improvement of the code includes all the following attributes:

- Remove unused codes
- Minimized amount of warnings
- Reduce duplicate codes
- Minimum code size
- Memory efficient programs
- Exception handling
- Software comment

These are example of a few rules. Complete set of rules can be referred at PMD Rulesets available in their website [3]. Testing and coverage could be included to make sure all aspect of the code is individually checked and would expect less failure when integrating with other modules. It is advisable to get the testing done parallel with the code development. Obfuscation is the final step to include into the process, as the code should be proven working before they are sent for obfuscation.

D. *Challenge Faced*

In implementing Continuous Integration, there were challenges came from both software development team and integration requirement.

On the software development team, Software Configuration Manager has to persuade the developers to have software developed not dependent on developers' local machine setup such as external path or external tools. The developer must not code in such a way that the program only works on his machine. For example, the developer would want to include static analysis on his own, the developer should not hard code the path to the static analysis tools path (C:/Document and settings/developer_name/path_to_static_analysis_tool/) in his program. This can be achieved by having a properties file point to the path of the static analysis tool so that the other developers could adopt the same path. Developers IDE (integrated development environment) nowadays can have the code built immediately when the project is opened, and hence making the build script written by the developer to exclude compilation instruction. We are getting used to depend on the IDE to build or compile the project until we tend to forget to have build and compile instruction included in the build scripts. Ideally, a build script should be able to run on a machine with the builder program installed (e.g Ant, Maven etc). This is because when a build server builds the project, it would not run the project from eclipse but it will execute from the builder program.

In some cases, external libraries should be included into the repository and the project directory so that less configuration effort is needed during the build. If the libraries are not included into the repository, they will need manual installation and configuration from the engineers to make it works. Libraries like unit test libraries, build task libraries, third party tools, if they are not too big, should be included in the repositories so that once the developers checks out the project, they will obtain all the dependencies.

On the expectation on integration, the Continuous Integration setup needs to conform to the organization's process and software tools implementation. Tool selection on Continuous Integration depends on the common requirement of the software projects. If the selected tools can suit a common pool of projects, then it will most likely to be acceptable for integration.

Selection of tools is important to ensure that the system is maintainable in the future, we shall go with the tools that is customizable and extension is provided should there be more functionalities added later. For instances, integrating a build system to version control system was easy since a plug-in did the job of bridging the two. Integration of the build system and the issue tracking system is a bit of challenge because the plug-in has limited functionalities and does not offer seamless integration. Having said that, a bridging application may be developed to complete the link, using the internal API of a issue tracker to communicate with the build server and have the build system to initiate it.

Another important factor is the performance observed in the Integration activity. As the numbers and size of the project grows, performance may become an issues to the entire integration cycle, there are three major areas that should be well taken care during build planning:

- Disk space
- Tool hosting
- Accuracy of the build

The above three factors are interrelated to each other. Disk space malfunction may fails the availability of the tool or build server uptime. Uptime failure will affect the accuracy of the build.

Disk space should be monitored at all time, if a certain build used up too much disk space then the build configuration need to be optimized. In disk space monitoring, there are many server monitoring tools in the market, one can use Java Melody to monitor the amount of available disk space regularly.

The hosting of Continuous Integration tools shall not to be taken lightly. Some tools are best not to be hosted in the same machine together with the Continuous Integration Server. For example, the code repository and databases shall be segregated out, as these types of servers have a lot of process working on their own. We would like to avoid other processes run together with the build process. Some build script uses timeout, we should take our build as a critical process and do not want other process interfere before the timeout. Some tools are okay to be installed together in the Continuous Integration server if it does not run on background. However having a parallel build could be an advantage. One build could be building a project that expose running services while another build will use up the exposed services. An example is like building a server and testing it by consuming the services from the server.

Most plug-ins or tools are invoked sequentially according to the build configuration. Tools like these are recommended to be installed together with the Continuous Integration server. This will optimize the execution time while reducing wastage due to network latency. Memory wise, it is recommended to run only a build at one time unless necessary.

The accuracy of the build is another factor to be considered, build server should be collecting source codes from the right sources. Issue may occur when the build server is not picking up the latest code from the source code repository due to some polling issues, especially when the project involve with large amount of source codes. A solution to this is to set a delay before kicking of a build. Another measure is to have some time between polling; build server should not try to collect source code immediately after a check in.

E. Impact

Continuous Integration brings new perspective in software development, it involves process automation throughout the development enhancing the productivity and hence improving the output.

The automation impacts a software project team in many ways. In this paper, we will look into the effect felt by the management, Software Configuration Manager, developers and tester as below:

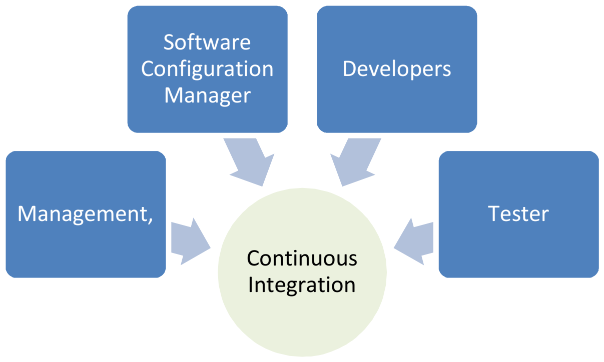


Figure 4. Parties Impacted by Continuous Integration

Continuous Integration produces rapid deployment of the product. Management side would appreciate this feature as they would be able to see the product and the changes it has been through incrementally. Good implementation of Continuous Integration would be able to grab all commit comments from the version control system to present it as part of the build, therefore making all development work visible to management counterpart, on top of that, management are able to see work in progress and defects captured incrementally since early stage.

Developers are able to see their modules in action and how they behave when they interact with other modules. Integration of module can be done easier because the deployed code will have codes contributed by all developers. The amount of work to integrate codes among themselves is lesser because the codes are getting frequently checked into build server for later compilation. A version control tool is used to retrieve all changes made by the developers, any changes to the build would require a build script change, which is maintained by the Software Configuration Manager or build manager.

SCM or build manager is responsible in having the whole Continuous Integration setup in proper. Add-ins selection in Continuous Integration environment is done based on the request from the tech lead, project manager or self initiated. For instances, SCM must be able to configure the project to run in the provided build machine and archive the product of the build later. Extra configurations on the build machine such as setting up emails, various testing environment, etc are also handled by the SCM. SCM plays an important role here in educating developers of what happen during the build so that the knowledge is shared and it would promote two way communication on how the build could be optimized.

Tester benefits greatly in automation, there are various testing tools that perform automation. The automation tools that provide these features usually have an agent that work its way to the system and run predefined scripts to execute system testing and carry the result to be compared with expected results. For instances, Unit tests and UI tests will provide a comprehensive reports with result accompanied. Testing like UI tests are beneficial to test on deployment especially in web environment, the UI test can verify if the web deployed are displaying the intended page. Although the amount of interaction script provided are incredible but sometime new web technologies may not be benefitted from it as yet. For instances, embedded flash or Silverlight may require more research on the test automation.

V. FUTURE WORK

The success of Continuous Integration is highly depends on the tools it used. Having the right set of tools will transform the integration work from time consuming and deadly task to a light and easy work. As such, it is crucial to promote an environment where it is easy for the team to be disciplined.

Sometimes, it's easy to miss-thought that the team is implementing Continuous Integration upon having the infrastructure set up and running. With the power that the tool brought all together, one must not forget that Continuous Integration is a practice that will only be successful if people are given their commitment and shared the same vision.

As a project scales up in complexity and size, together with a substantial of test cases to be triggered, Continuous Integration can become increasingly difficult to practice successfully, especially in relation to performance, reliability and the practice itself. As such, it's good to develop light and effective test cases, to make code base modular and to encourage the use of local integration server.

Also, suggestion to prevent build break by allowing pre-commit mechanism to enable engineers to run an integration build in their local, tested out the build with the new delta before committing the change to the formal build.

VI. CONCLUSION

The success of Continuous Integration is very much depends on the tools selected and discipline of the team. Continuous Integration could help to screen out defect, checking for quality, save time and increase efficiency. But it may not be the silver bullet; project team still need to deliver great code to produce a quality build and had a shared vision in implementing it. The only way to gain full benefit from a Continuous Integration process is if every engineer takes it as a personal responsibility to keep the build from breaking so that other members could keep working and integrating their changes towards a working software.

REFERENCES

[1] http://martinfowler.com/articles/continuousIntegration.html, accessed February 3, 2011.

[2] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftmanship (2009) Chapter 1, Clean Code.

[3] http://pmd.sourceforge.net/rules/index.html#Basic_Rules, accessed January 3, 2011.