Guidelines for Continuous Integration

ITS-GL049, Ver 1.0

Sharda Centre, Off Karve Road, Erandwane,
Pune, Maharashtra, India 411004

Table of Contents

1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to provide relevant guideline to Project Managers, Developers, Testers, Build and Release Managers and Team Members to know what is Continuous Integration, how it is to be applied and its benefits.

Continuous Integration is a software development practice where the members of a team integrate their software on a regular basis (at least once daily) to produce integrated builds. The software is built using build scripts, deployed in an environment (using deployment scripts), and tested (using automated test scripts). The underlying code is inspected for adherence to coding guidelines using tools which are invoked from a Continuous Integration server. The result of the testing and inspection is published as a report using the tools provided by the Continuous Integration server.

The entire process is executed using the functionality provided by the selected Continuous Integration server. There are only four features required for Continuous Integration.

A connection to a version control repository

- A build script
- Feedback mechanism (such as email)
- A process for integrating the source code changes (manual or Continuous Integration server)

For a project to implement Continuous Integration the following are mandatory

- Source code compilation
- Build scripts available and executed using Continuous Integration server.
- Automated Testing Scripts available which is executed by the Continuous Integration server.
- Once an automated build is run with every change to the version control system other features of Continuous Integration can be added to the Continuous Integration system which include
    - Data base integration – enabling continuous database integration in the build process of a Continuous Integration system.
    - Automated code inspection – use the Continuous Integration system to automatically run the code inspection rules.
    - Deployment - continuous deployment enables to deliver working deployable software at any point in time.
    - Documentation and feedback – generating document artifacts. Providing timely feedback to the developers and project stakeholders.

1.2 BENEFITS OF USING THE CONTINUOUS INTEGRATION

Some of the benefits of adopting Continuous Integration are

- Reduce the risk of getting surprises at the end of the project.
- Reduce repetitive process and free up valuable time for higher value work.
- Enable the release of deployable software at any point in time.
- Enable better project visibility.

- Establish better product confidence.
- Improve product quality as frequent testing and inspection leads to early identification and removal of defects.

1.3 When to use Continuous Integration

Continuous Integration process can be applied across projects/technologies and this will add value whether it is maintenance or development projects. The basic idea behind Continuous Integration is to build as frequently as possible. Frequent Building ensures the code is compile-able and testing ensures there are no regressions. Therefore an ideal target for the application of this process is a project which has unit test cases as a culture built into its team members. The process can also do integration and functional testing depending on the project nature.

The Continuous Integration practices should be adopted when

- There are rapid code changes to the project especially at the interface level which can lead to potential Integration issues
- Creation of build is a complex process involving many steps and is prone to human errors.
- The technology is new and hence early identification of issues through testing will identify weak areas.

2. ACRONYMS AND DEFINITIONS

Term/ acronym	Explanation
CI	Continuous Integration
PM	Project Manager
PL	Project Leader

3. CONTINUOUS INTEGRATION IMPLEMENTATION

This section lists the broad steps to be kept in mind while selecting a Continuous Integration server as well as other tools for implementation of Continuous Integration.

3.1 Continuous Integration Server and Tool Selection Process

Build tool's essential functionality includes:

- Code compilation – since compiling source code is the main ingredient in building software
- Component packaging – the build tool should understand how to package the necessary components
- Program execution – the build tool should have good support for invoking programs in its target platform.
- File manipulation – creating , copying , deleting files and directories are the functionalities to be supported by build tools

Build schedulers (Continuous Integration server) should have the following essential functionalities:

- Build execution – execution of the automated build on a periodic basis
- Version control integration – a tool that integrates with the version control system
- Build tool integration
- Feedback – is essential for Continuous Integration to provide instant feedback by instant message or text message
- Build labeling – the tool should mark the artifacts that contributed to a given build

Listed below are the miscellaneous factors that are to be kept in mind for Continuous Integration server and Tool Selection.

- Project Technology: This will be the most important factor. Most of the tools available are support specific technology only. For example from a Continuous Integration server perspective LuntBuild supports only Java whereas CruiseControl.NET supports only .NET, CruiseControl.rb supports Ruby ,Tinderbox supports Perl and BuildBot supports Python. Similarly ANT scripts support Java whereas NAnt support only.NET projects.

- Supported Tools: Every project will normally have a Source Control Repository that's being used across locations. The selection of Continuous Integration server will also be influenced by this.

- Cost of the Tool: Tools like Bamboo, Cruise are commercial while Hudson, LuntBuild, CruiseControl.NET, CruiseControl are freely available.

- Third Party Support: Many Continuous Integration servers depend on Third Party plug-ins for enhancing the features. For example Bamboo supports the ClearCase only after installation of plug-in. Similarly FTP feature in Hudson can be enabled after installation of a plug-in and is not available with the standard product. Depending on the project requirement, the list of features supported by the tool needs to be evaluated.

3.2 CONTINUOUS INTEGRATION ENVIRONMENT SETUP

This section will list the steps to be followed to set up Continuous Integration environment.

- The first step is to identify the Continuous Integration server as listed in the section above.

- The next step is to configure the Continuous Integration server to connect to the source safe repository to ensure that the link is working correctly. This become very important if the Continuous Integration server and SourceSafe repository are at different locations (e.g. One at offshore and another at onsite) as performance could become a critical factor when setting the build frequency.

- The next step is to integrate the Build Scripts with Continuous Integration server and setting the configuration for build creation. This would depend on the project. It is advised that a build is created at every check in. The build scripts will also clean the environment before creating the build.

- The third step is to integrate the testing framework with the Continuous Integration server framework.

- Once the above steps are completed, the source code inspection tool needs to be integrated with the Continuous Integration server.

- The last step is to automate the feedback process. Most of the Continuous Integration servers have features which need to be configured. Examples would be mails to be sent on build failures, SMS to be sent etc. Also the Continuous Integration servers have pages where the status of builds can be seen along with the details of failures.

3.3 DECIDING BUILD FREQUENCY

Continuous Integration is all about “build as frequently as possible”. Practically building after every check in is the most ideal way to go. Some projects are not meant for such frequency and in these cases the team must decide on the most suitable frequency. Most projects can have builds twice a day, one in the afternoon (usually during lunch hours) and the other late at night. This ensures early morning check-ins are tested in the afternoon and late evening check-ins are testing @ night. In all cases, always ensure that there is a Version Control quiet time of at least 5 minutes before the build is triggered. This is to ensure that if a developer forgets to check in all files a build triggered in between goes waste. This quiet time can be configured within the Continuous Integration server.

Some of the factors that needs to be taken into account for deciding on build frequency

- Check-In Stability
- Tools Auditing Frequency (How often do you want the status of say code review or find bugs)
- Testing Frequency.

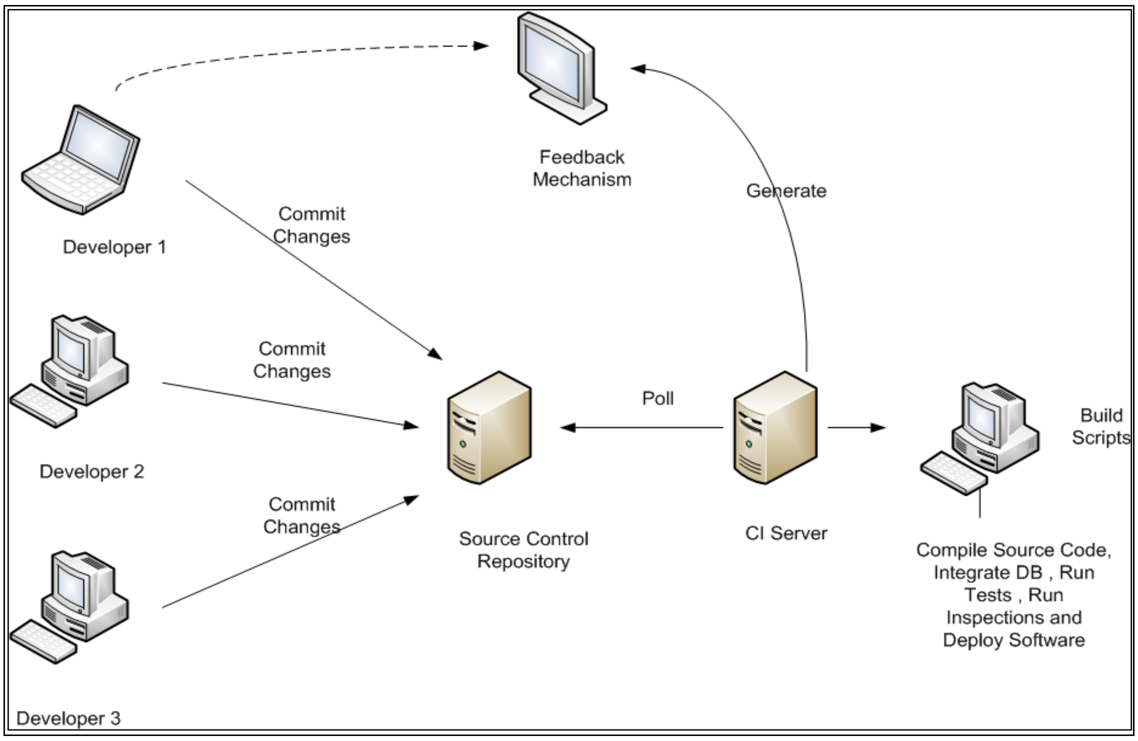
3.4 CONTINUOUS INTEGRATION ROLES AND RESPONSIBILITIES

This section maps the different roles in a project to the responsibilities in a project. The list is an addition to the standard projects management task . Also these tasks are indicative and can be modified to suit the needs of the specific project.

Role	Responsibility
Project Manager	• To participate in discussions related to selection of different tools and to take a final decision on the tool for the project. • To ensure availability of licenses (if needed) for different tools. • To ensure that the resource (Hardware, Software) is available to the team for implementing Continuous Integration.
Project/ Team Lead (Development and Testing)	• To ensure that every team member is following the process for Continuous Integration. • To ensure that the necessary reports related to Quality are being generated and preserved in the Source Safe Repository.
Team Members (Development)	• Follow all the process laid for implementation of Continuous Integration. • To ensure that relevant tool execution is happening correctly for each piece of code. • To fix defects reported by the tool. • To ensure that broken code is fixed at developer level before checking in to main repository. • To capture metrics related data as required by the tool.
Team Members (Testing)	• To create automated test scripts for testing of the code. • To configure the testing tool correctly. • To capture testing related metrics data as required by the tool.
Specialist Skill or Project Lead	• Installation and configuration of the Continuous Integration server. • Handling issues related to Continuous Integration server during the course of the project. • Creation and maintenance of Build Scripts. • Configuring different tools (like Code Inspection, Testing tools) with Continuous Integration server.

3.5 Environment

Continuous Integration can be implemented with minimal additional needs. The diagram below shows a typical environment for Continuous Integration.



Apart from the standard project environment, a machine with standard organization's configuration would suffice as a Continuous Integration server. It is assumed here that the project would already have a Build environment as specified by the project requirement. In case it is not available, this would also be an additional requirement. This is needed as Continuous creation, inspection and testing of builds would be a performance bottleneck if this build machine is used for other purposes.

4. APPENDIX / REFERENCES

The Table below gives a sample listing of Java and .NET tools:

S.No	Area	Sample Java Tool Names	Sample .NET Tool Names
1	Continuous Integration Server	LuntBuild, Hudson, CruiseControl,Bamboo,Anthill	CruiseControl.NET, LuntBuild, Bamboo, AntHill Professional
2	Build Tools	ANT, MAVEN	NANT
3	Code Review Tools	ePMD	FxCop
4	Testing Frameworks	Junit	NUnit

5. DOCUMENT HISTORY

Version	Date	Author (function)	Reviewed by	Approved by	Nature of changes
Issue 1.0	26-Feb-15	Sourabh Sharma	Sandeep Sapre/Madhuri Kenjale/Devendra Gujar	Amitabh Shanker & Madhuri Kenjale	First Issue