ENHANCING SOFTWARE QUALITY

INDIUM

Implementing Continuous Integration Testing

Prepared by:

Mr Sandeep M

Table of Contents

1. Abstract

Assume a scenario where an application gets codes from several testers/developers from several branches at different intervals of times. Each branch of code is generating a new build in the application. And these applications have to be released frequently after going through the process of integration.

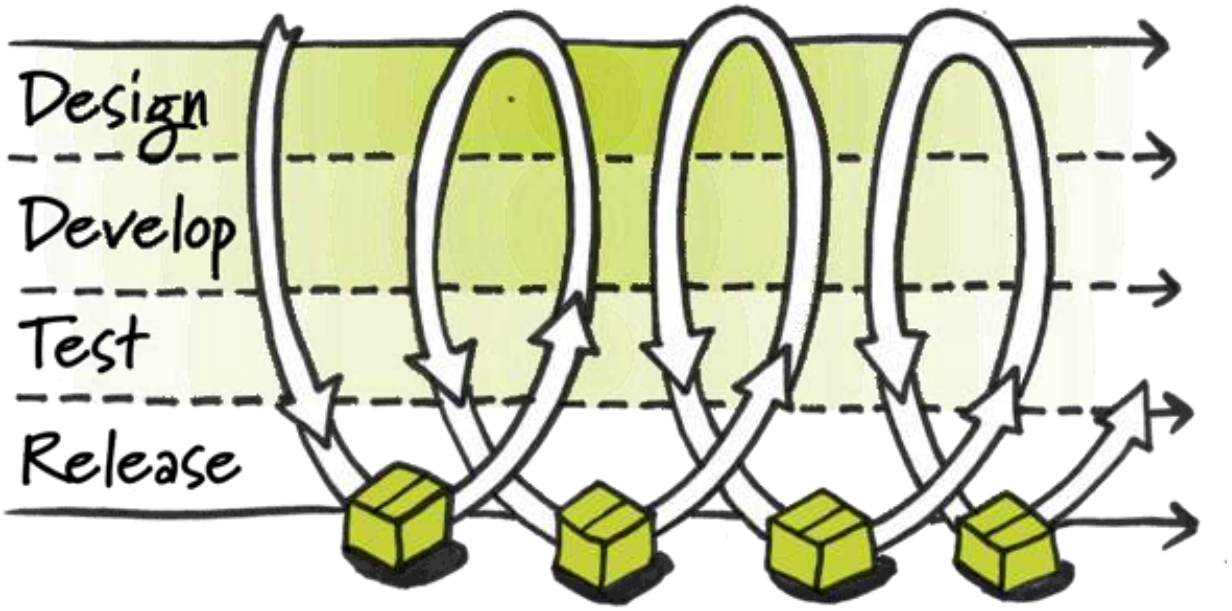
Apart from these, the application has to be tested each and every time new build is released. The QA engineers would face hectic job in testing the application and achieving the desired quality in an environment of this kind.

It had become an impossible mission to come up with a solution for this. But the solution for this was found using the concept of Continuous Integration.

Continuous Integration is a practice followed in Software engineering, where the code will be merged from all the developers several times a day. This will make sure that the software/application is built or tested regularly.

By using CI, tests can be run each time when a new code is deployed or a new application is developed. To accomplish this, we need a tool suitable to perform the task which will not have much of the programming skills. The tool must also be able to handle applications from different technologies.

2. Introduction to Continuous Integration (CI)



There is a bad way in Software Development, where the developments keep going, and merging code happens from all branches and many times, but there will be no track of what changes happened where and whether it will affect any other code in the application. This would lead to “Integration Hell”. Later during the build, it will be realised if there is any code breakage and give full pressure for the Testers as well as Developers.

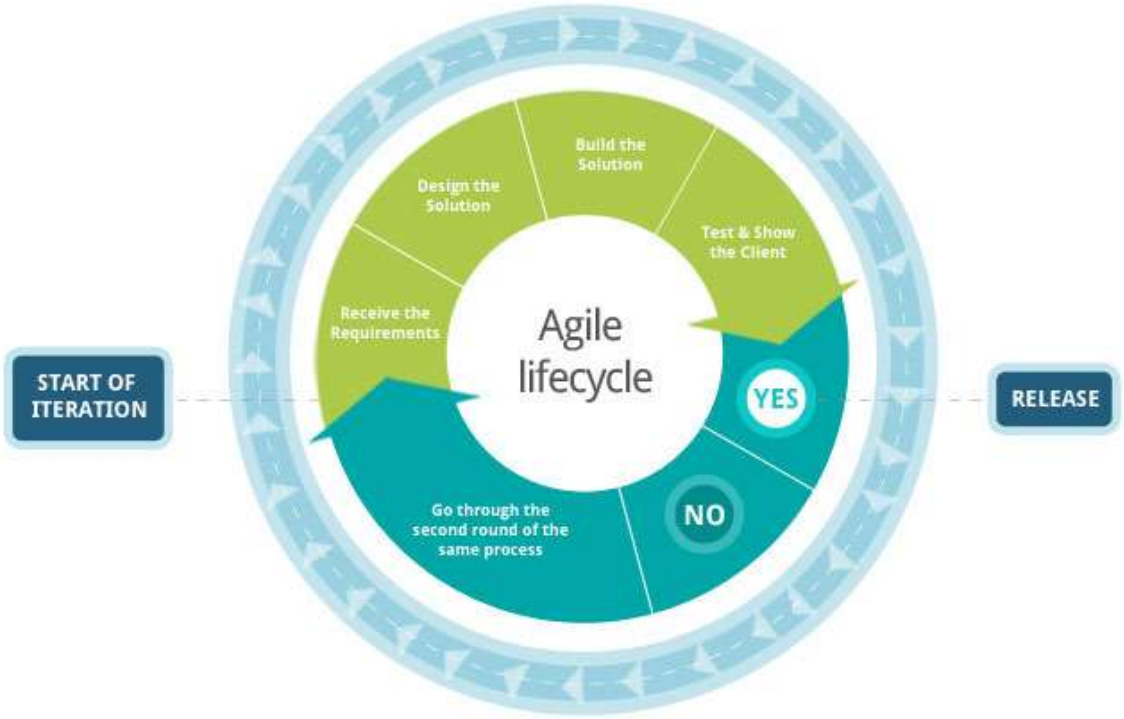
2.1 What is CI?

Continuous Integration is a practice implemented in software development, wherein the integration of the works happens frequently from the Team members. Normally, single person would integrate at least once in a day, which will lead to multiple integrations on a single day. Each of these integrations must be verified to detect the integration errors as quickly as possible, so that it can be fixed at the very same moment. This would reduce the integration problems and help the developer to produce a cohesive product / application more rapidly.

2.2 Why CI?

The main reason to have CI is to get rid of “Integration Hell”. It is not said that CI is done to overcome or fix errors. It is to make sure that the application is has less problems or defects when integrations are done and also to develop the product rapidly.

3. CI for Agile Methodology

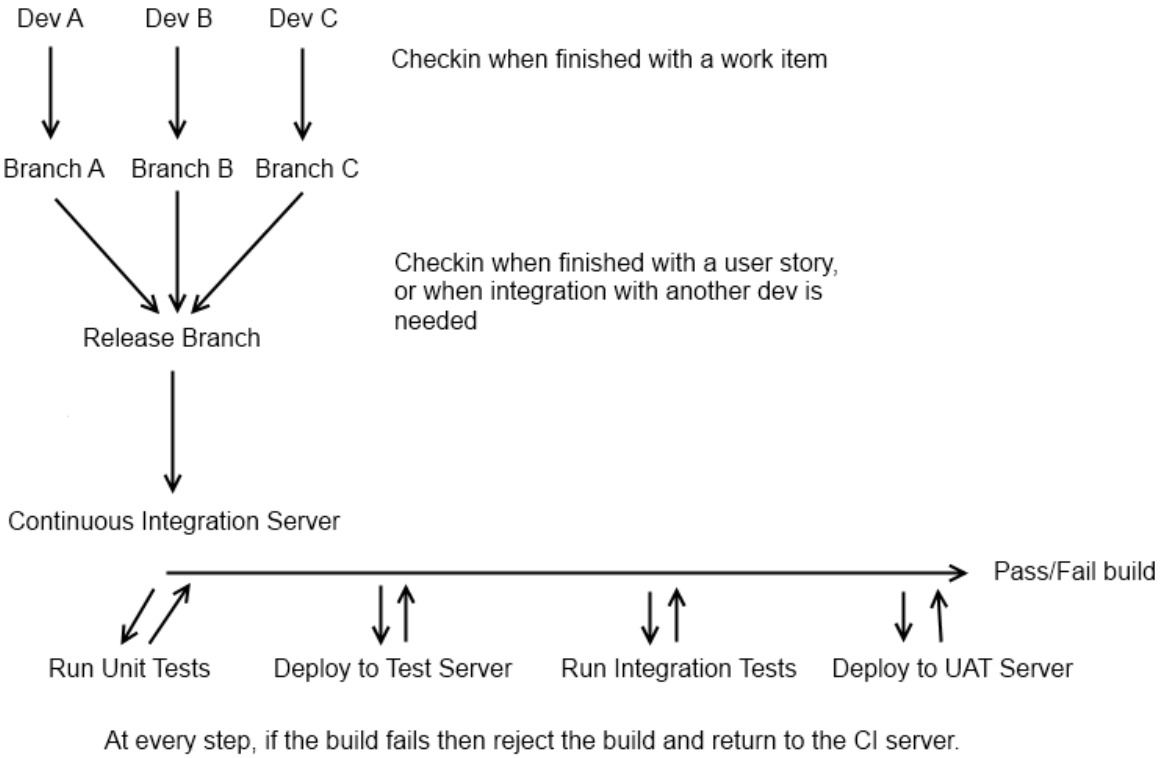


The concepts brought by Agile Method to software development changed the way how teams organize their work, adapt to changing requirements, and release software. Continuous integration (CI) was created for agile development, so the agile approach is the context for any CI discussion.

The idea is not to try to solve every issue up front but, to focus on what you already know. So the team designs, builds, and tests what they know about the desired functionality. This creates a working product based on a subset of the complete product's requirements.

Test-driven development fills the gap of evaluating the work integrated by the developer. With TDD, you build the test and then develop functionality until the code passes the test. As each new addition to the code is made, its test can be added to the suite of tests that are run when you build the integrated work.

4. Work Flow



At every step, if the build fails then reject the build and return to the CI server.

1. While working on a change, developer will take a copy of the current code from the base on which he has to work.

2. As developers submit the changed code to the repository, this copy gradually reflects in the repository code.

3. Not only there will be change in the existing code, but new code can be added along with the new libraries, and other resources that could create some dependencies and the potential conflicts.

As long as the code remains checked out, the risk of multiple integration conflicts and failures will be huge when the developer branch is reintegrated. When developers insist to submit code to repository, they should update their code first to reflect the changes from the repository.

Continuous integration will involve early and often integration, so as to avoid the risk of "Integration Hell". This practice reduces rework and thus reduces the cost, time and effort.

5. Best Practices

Here is a list of best practices suggested on how to achieve continuous integration, and how to automate this practice.

Maintain a code repository	• Practice of using revision control system would be helpful for the developers and testers. • Everything required for the project should be placed in the repository. The trunk should be the place for the working version of the software.
Automating the build	• Single command must have the ability to build the system. • Automation of the build must also include automating the integration, which may include deployment into a production.
Making build self-testing	• Once the code is built, all tests should run to confirm that it behaves as the developers expect it to behave.
Everyone commits to baseline every day	• By regular commits to the repository, the number of conflicting changes can be reduced. • Checking in a week's work will run into the risk of conflicting with other features and can be very difficult to resolve.
Every commit should be built	• The commits must be built to the current working version with a view to verify that they integrate correctly.
Keep the build fast	• The builds must be complete rapidly, so that, the problems with the integration are quickly identified.
Test in a clone of the production environment	• Have a Pre-Production Environment, scalable version of Actual Production. Perform testing there and send build for Production
Making easy to get the latest deliverables	• Making all the builds readily available to testers can reduce the amount of rework when rebuilding a feature that doesn't meet requirements. • In addition to that, early testing reduces the chances that defects survive until deployment.
Everyone can see the results of the latest build	• It will be easy to find out which build breaks and, if so, who made the relevant change.
Automate deployment	• Most of the CI systems allow running scripts after a build finishes. • It is also possible to write a script that can deploy the application to a live test server.

6. Testing in CI

The keys for successful CI are speed and frequency. Speed, because we need to run our builds and unit tests as the code is checked in, deploy and perform testing as the software is built. Frequency, because we want to check in our code regularly and see differences from previous check-in.

However, complete and thorough testing would consume lot of time and long-running tests can take up to 12hours or even longer to complete. On one hand, we want to run as many tests as possible; on the other hand, we need to know if our change is responsible for any problems.

In order to get the best, we need to consider two different build types: builds that are triggered by code changes and the scheduled builds.

Builds that are triggered by code changes can respond quickly enough in order to:

- ✓ Compile the source code
- ✓ Run unit tests

Adding steps like deploying the built modules to production system may cause a delay in getting feedback from the CI system. So you should also configure scheduled builds, that run regularly, but which are not triggered by a code change. This will allow you to:

- ✓ Compile the source code
- ✓ Run unit tests
- ✓ Deploy to a production system
- ✓ Run advanced tests

These builds can be scheduled whenever you choose—daily, nightly, weekly, etc. This can be chose in a combination, like, scheduling nightly builds on the working week, which runs sanity tests, and schedule a longer-running build on the weekend, which will perform a full build, deploy it, and run a full battery of tests on the deployed system.

7. Types of CI Tests

There are different types of tests that can be run on a CI system. Here we will describe the most common ones.

Deployment Tests

- Ensures Installation is working fine
- Ensures correctness off all modules and the whole system also

Integration Tests

- Performs similar to Unit Tests, but work against real implementations
- Performed on a System where all modules are deployed and configured correctly

Smoke Tests

- Tests basic functionality to ensure further or more testing can be performed

Functional Tests

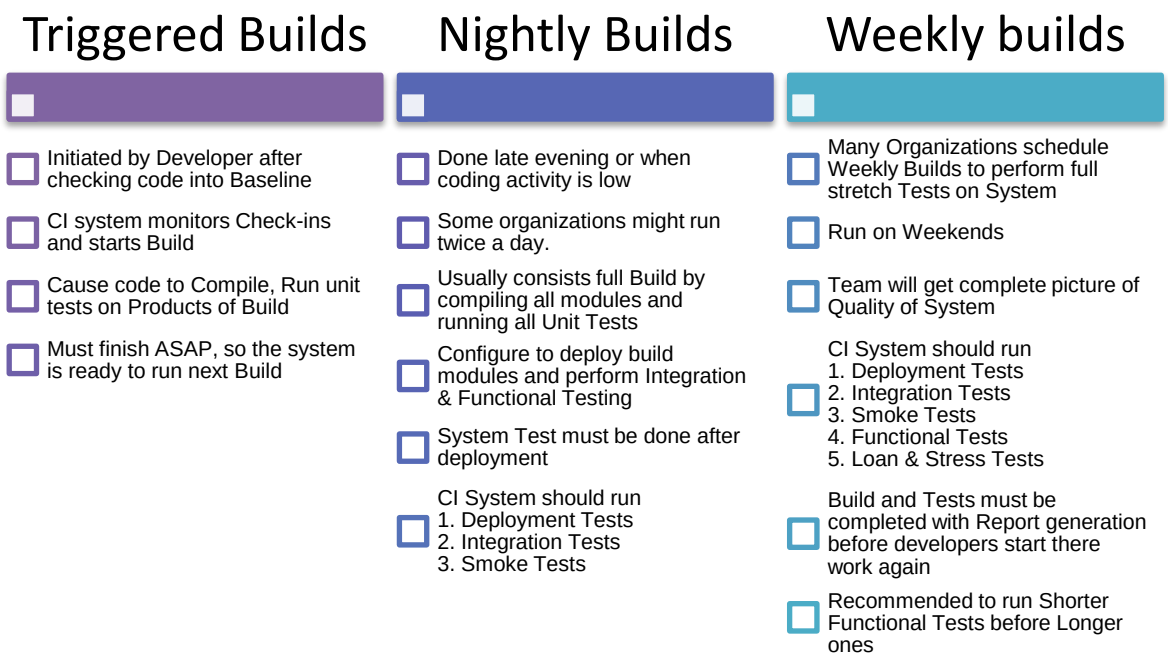
- Executes the scenarios from User's perspective, simulating User's behaviour
- Performed directly against User-Interface of AUT

Load/Stress Tests

- Ensures System performs perfectly when subject to different loads
- Usually tested generating flow or drop in concurrent users or by running different scenarios concurrently

8. Build Executions

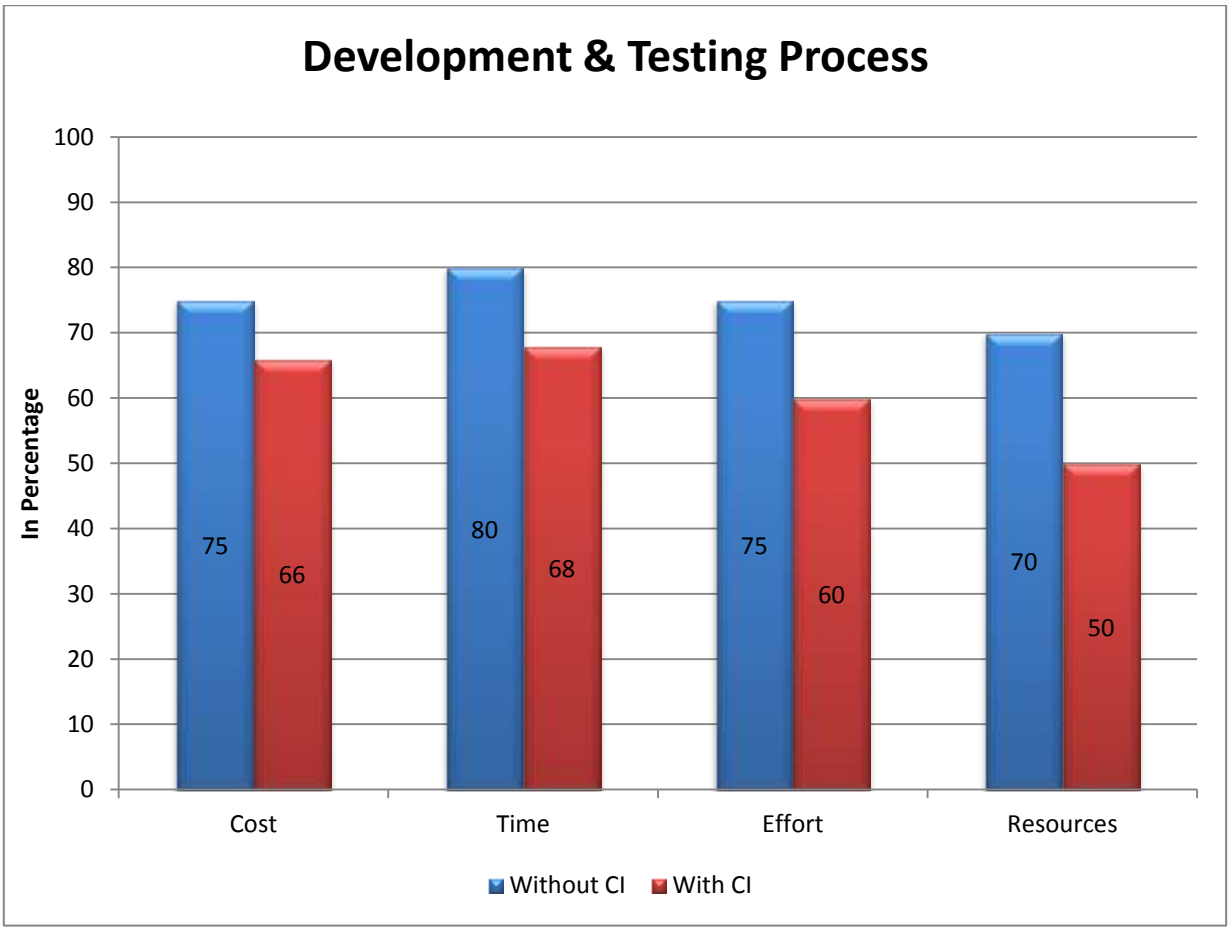
As we know from previous, we can trigger the execution for a build or schedule the same. Here we see what all can be done in different types of builds.



9. Benefits of CI

10. A small Example

The below Graph depicts the cost, time, effort and resource usage for development and testing of product with and without Continuous Integration. This chart was prepared gathering the data from some of my friends and colleagues who had a brief experience of working with CI.



It was also found that the amount of Bugs found after the Integration of Modules were decreased by around 40% with the usage of Continuous Integration.

11. Tools for CI

Organizations wishing to experiment with CI tools have plenty of options. Popular open source tools include Hudson, Jenkins (the fork of Hudson), CruiseControl and CruiseControl.NET. Commercial tools include ThoughtWorks' Go, Urbancode's Anthill Pro, Jetbrains' Team City and Microsoft's Team Foundation Server.

Most of the players in the CI market are pretty well established and the tools provide similar services.

Continuous integration tool capabilities

A common feature among the available CI tools is strong support for software configuration management (SCM) tools. CI tools have plug-in systems that allow them to integrate with SCM tools as well as add functionality.

Other features which are to be considered when evaluating CI tools cited by the experts are: visualization of the release process, a customizable and easy-to-read dashboard, trending, tool integration (issue management, peer review, testing, etc.), and the ability to create a process workflow.

Ultimately, though, the CI tool that you choose should integrate with your ALM system. The idea is that you should be looking for best of breed and what will meet your needs, and then make sure it can integrate with your system.

12. References

- Wikipedia.com
- SearchSoftwareQuality.com
- Programmers.StackExchange.com
- Quora.org
- TeamTreeHouse.com

13. Author's Biography

Sandeep has 2 years of experience in IT industry. He has worked as an Intern (Junior Developer) in Buffered Software Solutions for 6 months. Presently he is associated with Indium Software since February 2013. He acquired his Post Graduation in MCA under Bangalore University. He is a DotNet and 3D Animation certified professional. He is interested in development of small applications for mobiles and computers. In his spare time, he reads novels, plays games, watches movies and listens to music.