Continuous Integration (CI)

Introduction

A long standing problem for software development teams has been to maintain the stability of an application while integrating the changes made by multiple developers. The later that integration is delayed the more potential there is for risk and failure in the integration process. Continuous Integration (CI) tries to mitigate this risk by frequently integrating small changes into an evolving code base.

This white paper discusses –the CI process– and shows how we at CC Pace practice it.

Intended Audience

This document is aimed at:

- Developers who will incorporate the CI phase of application development process
- Lead programmers and data architects who need to complete the groundwork tasks required before the CI phase
- Information Services (IS) managers who will provision the infrastructure required to support the CI phase
- Project managers who need to understand what tasks must be completed before the CI phase kicks off and the best practices to be incorporated as a team culture

What is it?

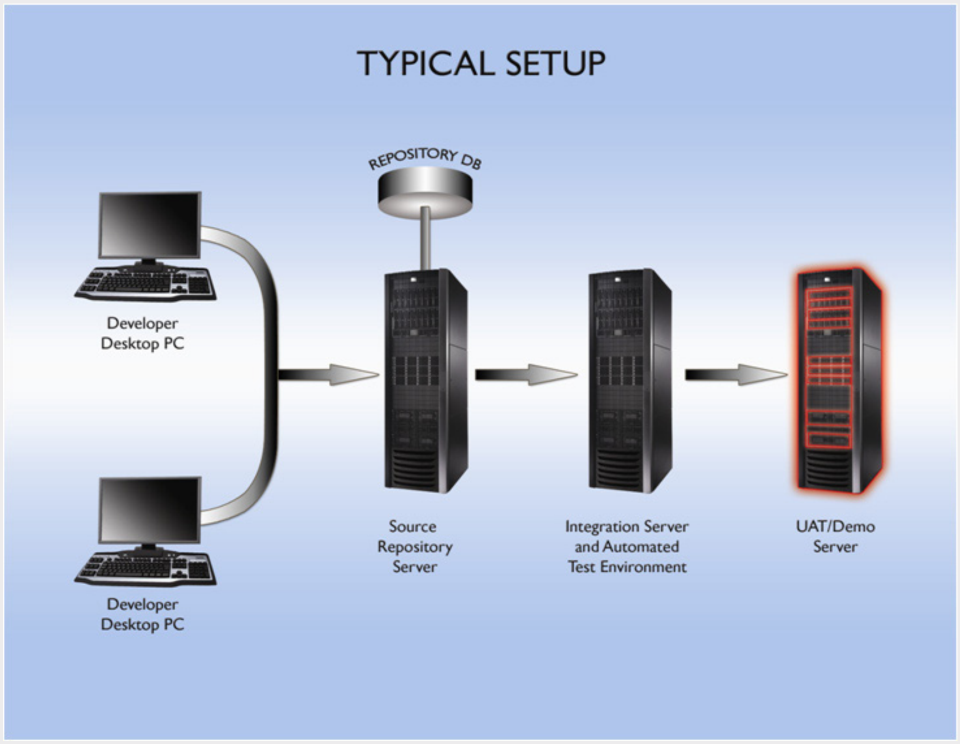
CI is the process designed to ensure that your software is always working, and that you get comprehensive feedback in a few minutes as to whether any given change to your system has broken the functionality. It exposes integration problems as early as possible. It also aims to have a built, tested, and potentially releasable build at every stage of a project.

As defined by Martin fowler:

“Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily—leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.”

So, how do we start?

We start by setting up the CI server components described below and illustrated in Figure 1.



Developer workstations host the chosen IDE, source/version control client, and have access to a per-workstation sandboxed database schema.

Source repository server hosts the Version Control System (VCS).It is the central location for all the source code. It allows multiple developers to collaborate projects simultaneously. VCS must be set up to allow non-exclusive checkouts with possible merging – i.e., two features may affect the same managed file at the same time. No exclusive checkout locks should be configured, as this may prevent a developer from finishing a task.

Integration Server hosts the continuous integration build agent, which constantly monitors the source control system for check-ins. When check-ins is detected, the build agent initiates a complete automated rebuild of the system, and runs the complete suite of automated regression tests, if any. Alerts will be sent to the team if the build fails for any reason.

UAT/Demo Server is dedicated to the use of the business team to verify product increments.

Can I have some examples of CI configurations?

CC Pace has incorporated the following CI configurations in the delivery of JAVA and .NET solutions.

Table 1 Various CI configurations

Flavors	Development Environment	Source Repository Server	Integration (CI) Server	UAT/QA Environment
Java EE	• *IDE: Eclipse, IntelliJ* • *TDD Framework: Junit, Mockito* • *Build Tool: Apache Ant* • *Source Repository Client : SVN Client* • *Database Client : SQL Client*	*Subversion* *Git*	*TeamCity* *Hudson*	Fitnesse Selenium
Microsoft .NET	• *IDE: Microsoft Visual Studio* • *TDD Framework: Nunit, MSTest, Moq* • *Build Tool: Nant, MS Build* • *Source Repository Client : SVN Client* • *Database Client : SQL Client* • *Code Coverage: NCover*	*Subversion* *Team Foundation Server* *Git*	*TeamCity* *Team Foundation Server*	WatiN Selenium

What is the process?

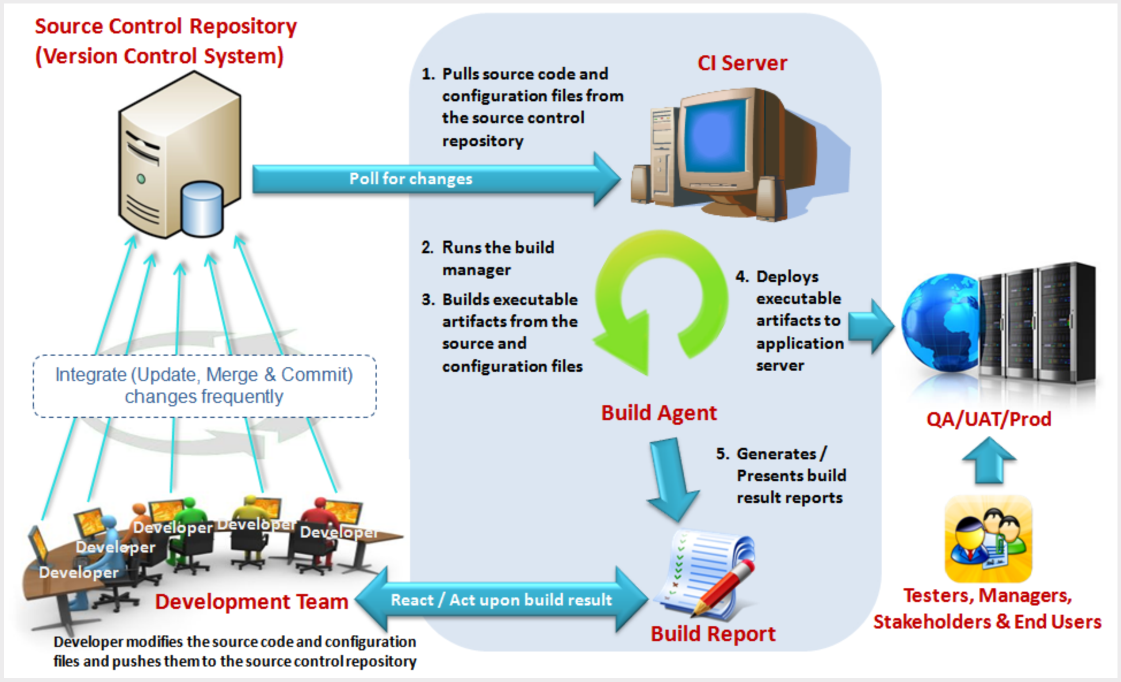
Ideally, CI is a completely automated process that interferes as little as possible with a developer's productivity. To start the process, developers are required to do three things when integrating their code with the VCS:

1. Update their local code base with the latest changes from the VCS.
2. Merge any conflicts – this is a manual process if different developers have changed the same lines of code.
3. Build and run tests locally.
4. Commit the local code base to the VCS only after your local build is successful. Never commit broken code.

Once the code is committed to the VCS, the CI server detects the change and pulls the latest code and dependencies from the VCS. The code is re-built and automated tests are run to verify that nothing has broken. Other tasks in the build process are also run, such as database schema updates and test data setup.

The CI server provides ongoing status updates as each step in the build process is completed. The developer can check this status and react to any failed builds by making the necessary changes and checking in again.

The end result of a successful integration is a compiled and tested artifact that can be promoted to another environment, such as UAT.


Source Control Repository (Version Control System)
Poll for changes
CI Server
1. Pulls source code and configuration files from the source control repository
2. Runs the build manager
3. Builds executable artifacts from the source and configuration files
4. Deploys executable artifacts to application server
Build Agent
QA/UAT/Prod
5. Generates / Presents build result reports
Integrate (Update, Merge & Commit) changes frequently
Developer
Developer
Developer
Developer
Developer
Development Team
React / Act upon build result
Build Report
Testers, Managers, Stakeholders & End Users
Developer modifies the source code and configuration files and pushes them to the source control repository

How often does the build run?

Build Type	Trigger / Usage
Incremental/Continuous	*Triggers when code is checked in. Does a quick compile and runs unit tests*
Daily/Nightly	*Trigger once daily. More resource intensive. Does a compile and full suite of unit tests and possibly additional automated integration and functional tests*
Weekly	*Trigger once every week. Does a compile and full suite of unit tests and possibly additional automated integration and functional tests*

What are the barriers for CI adoption?

Developer: *"I can't seem to bring my work to an integration point in less than <insert long period of time here>."*

Response: *Well-practiced Test-Driven Development (TDD) will force smaller units of code*

Developer: *"Won't I be spending a lot of time merging my code with others?"*

Response: *Not a lot of time, and much less than with a "big bang" integration Unit tests tell you that your merge succeeded Integrate early and often!*

Infrastructure Engineer: *"Some of these CI tools are really tough to configure!"*

Response: *True. Start small, not too ambitious, not too many inspections, etc. Examine the universe of tools*

Key highlights

All the code, build scripts, test scripts etc should be versioned in a single version control repository.

- **Successful build should depend on passing all unit tests**
- **Separate and isolated Integration Server**
- **Frequent incremental commits**
- **Speed up builds by**
  - Designing independent and parallel build-agents for separate code modules. This may require separate database instances for each build-agent if they run in parallel.
  - Using a mock testing framework to limit the interactions of tests with the database.
- **Rapidly communicate status (success, failure or in-progress) of a build**