Connecting the dots between

CI-CD, GIT and Magic xpa

This document provides an overview on the following subjects:

What is Version Control?

Version control systems are a category of software tools for managing changes to source code over time.

Version control tools keep track of every modification to the source code. If a mistake is made, developers can roll back the changes made or compare their code with earlier versions to help fix bugs and errors while minimizing disruption.

Why is version control used in a development environment?

- Maintain a complete **long-term change history** of every file
- **Branching and merging:** allow team members to work concurrently on independent streams of changes
- **Traceability**: Trace each change made to the code and annotate each change with a message describing the purpose and intent to assist with root cause analysis and other forensics
- **Task management:** Version control tools connect with project management and bug tracking software such as Jira

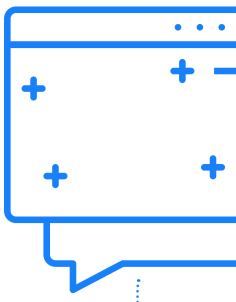
Types of Version Control systems

Centralized VCS

These systems (such as CVS, Subversion, VSS, TFS) have a single server that contains all the versioned files and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

Distributed VCS

In a DVCS, clients don't just check out the files' latest snapshot; instead, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is a complete backup of all the data. Most of the operations are done through the local repository.



Agile methodology

Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development and the 12 Principles behind it that was released in 2000 and gained tremendous popularity as a common approach to software development since.

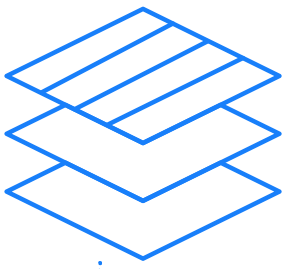
The agile approach seeks the continuous delivery of working software created in rapid iterations. In practical terms, agile development is all about delivering small pieces of working software quickly and continuously.

Usually, agile software development requires small teams of software developers and business representatives to meet in-person throughout the software development life cycle. Agile is a lightweight approach to software documentation that embraces—rather than resists—changes at any stage of the life cycle.

What is DevOps?

A shortcut for Development Operations, DevOps is about bringing together historically separate functional areas. It involves bridging the gap between software development ("Dev") and IT operations ("Ops") teams, often to be able to release software faster and with better stability.

Since its introduction about a decade ago, DevOps has become an umbrella buzzword of sorts for any and every trend in the software development + IT operations space. DevOps is still evolving, encompasses so many areas, and is adapted and adopted based on its specific business objectives, priorities, and existing knowledge base.



What is Git?

Git is, by far, the most widely used version control system in the world today. Numerous commercial as well as open-source software projects rely on Git for version control.

Git was developed by Linus Torvalds, the creator of Linux, in 2005 for the development of the Linux kernel, with other kernel developers contributing to its initial development. Git is free and open-source software distributed under GNU General Public License Version 2.

Git is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control system). In Git, every Git directory on every computer is a full-fledged repository with a complete history and full version-tracking abilities, independent of network access or a central server. Rather than having only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion.

In addition to being distributed, Git has been designed with performance, security, and flexibility in mind.

Git Repository

The purpose of Git is to manage a project, or a set of files, as they change over time. Git stores this information in a data structure called a repository.

A Git repository tracks all changes made to files in your project, building a history over time.

What is the difference between Git and other VC systems?

The major difference between Git and other version control systems is the way Git stores and retrieves the data.

Conceptually, most other systems store information as a list of file-based changes. These other systems think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).

Git thinks of its data more like a series of snapshots of a miniature file system. With Git, every time you commit or save the state of your project, Git takes a picture of what all your files look like at that moment and stores a reference to that snapshot.

What are the main attributes and advantages of Git?

Branching and Merging

The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model.

Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

Lightweight and Fast

Git is fast. With Git, nearly all operations are performed locally, giving it a speed boost over centralized systems that constantly have to communicate with a server.

Distributed

One of the most useful features of any Distributed SCM, Git included, is that it's distributed. This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

Data Assurance

It's impossible to get anything out of Git other than the exact bits you put in. The data model that Git uses ensures the cryptographic integrity of every bit of your project. Every file and commit is check-summed and retrieved by its checksum when checked back out.

Staging Area

Unlike the other systems, Git has something called the "staging area" or "index." This is an intermediate area where commits can be formatted and reviewed before completing the commit.

Does Magic xpa support version control systems?

Magic xpa supports version control systems such as TFS, Subversion providing they have an MSSCCI interface. Magic xpa supports Git, which is a distributed VCS. (refer to Git below)

Does Git apply only to huge applications?

The application size is not a factor; Git can support any kind of application.

Is Git relevant for a single developer software house?

No. Even small teams or one-man shops can use Git and gain from its advantages.

Can Git be used for the existing projects or only for new projects?

Git can be used with all projects, including existing ones. Usually, a development team would prefer to keep the history of an existing project once managed by some other VCS, in Git.

How do I get started with Magic xpa and Git?

When working with Git, development teams make use of branching and merging to support agile development. Since Magic xpa has an internal code structure, MSE provides a tool for Comparing and Merging its XML sources.

This tool has a unique user interface, similar to the Magic xpa studio, comparing and showing the differences between sources in different branches, and providing merging capabilities, therefore referred to as “Compare & Merge.”
The C&M tool’s license is per developer and has to be purchased via the local Magic branch.
The C&M tool needs to be installed on every dev machine and then activated with the license provided by MSE.

Once you open a project that is under a Git repository for the first time, Magic xpa will ask you for the location of the merge.exe executable.

What is CI/CD?

The CI/CD is one of the best practices used in DevOps for delivering code changes more frequently and reliably.

Continuous integration

Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

Continuous Delivery

Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that you also have automated your release process on top of having automated testing, and you can deploy your application at any point of time.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as often as possible releasing small batches that are easy to troubleshoot.

Continuous deployment

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes your production pipeline stages is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take the pressure off the team as there isn't a Release Day anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.

How does Git fit into CI/CD and the agile methodology?

Git is the most widely used version control system in the world and is built for continuous integration. It supports an efficient branching and merging mechanism. Git is the de facto standard for agile software development when it comes to version control systems.

GIT is flexible enough to support a range of workflows that suit any given software team's needs. It's distributed -- rather than centralized – nature gives it superior performance characteristics and allows developers the freedom to experiment locally and publish their changes only when they're ready for distribution to the team.

Besides the benefits of flexibility and distribution, there are key functions of Git that support and enhance agile development. Think of Git as a component of agile development; changes can get pushed down the deployment pipeline faster than working with monolithic releases and centralized version control systems. Git works the way your agile team works (and should strive to work).

CI/CD pipeline

A CI/CD pipeline is a series of steps that must be performed in order to deliver a new version of the software.

A CI/CD pipeline introduces monitoring and automation to improve the process of application development, particularly at the integration and testing phases, as well as during delivery and deployment. Although it is possible to manually execute each of the steps of a CI/CD pipeline, the true value of CI/CD pipelines is realized through automation.

The steps that form a CI/CD pipeline are distinct subsets of tasks grouped into what is known as **a pipeline stage**. Typical pipeline stages include:

PUSH 1

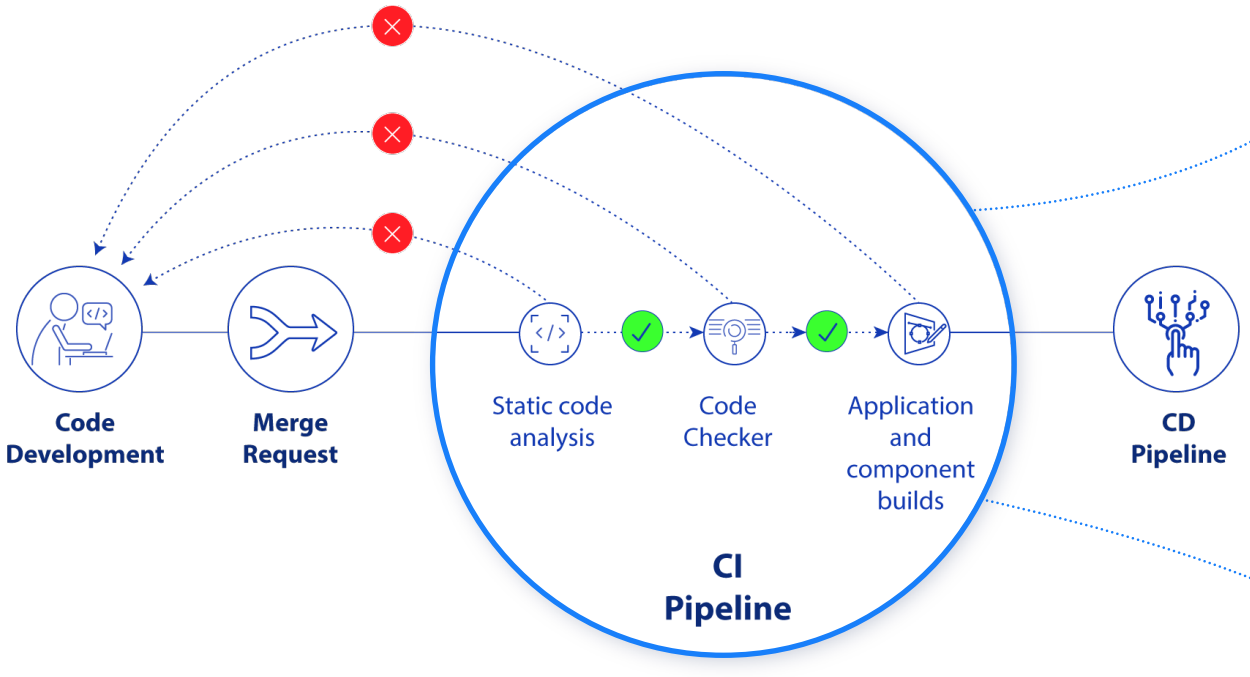
Pushing code to a shared repository (develop branch) for integration testing

TEST 2

Testing with an automated process that builds the app in a dedicated server to confirm and validate the integration of new code in current code

FIX 3

Fixing any bugs, malfunctions or anomalies detected, before continuing to add more features and Continuous Deployment (CD)



What would my management gain from adopting the CI/CD development methodology?

- Better collaboration between the development teams, provides better visibility and communication.
- Pushes developers to create modular, less complex code
- Optimized processes and avoids last-minute chaos at release dates
- Awareness of project status through automatic generation of metrics, such as code coverage, complexity, and features completed
- Constant availability of a “current” build for testing, demo, or release purposes
- More time available for adding features
- Saves time and money in the project lifecycle

What would the development team gain from adopting CI/CD development methodology?

- Early detection and tracking of integration bugs, eliminating long and tense integrations
- Frequent committing of the code for integration tests prevents integration problems; only a few changes are lost when reverting to a bug-free state.
- Focus on developing functional, quality code and supporting the development of team momentum
- Less time testing and debugging
- Immediate feedback on the system-wide impact of local changes

Does Magic have any plans to provide more CI/CD related features within the product?

Magic has plans to expand its product to include more features supporting CI/CD, allowing seamless integration of Magic xpa as an IDE with CI/CD pipelines. These include the ability to run Magic xpa’s checker in the background, pre-load checker output onto the studio, support for CI DevOps operations on Linux, and more.