**CI/CD/CD**

CI/CD/CD these are the backbone of modern software automation industry. Let’s go through and have an overview of what is CI/CD and CD.

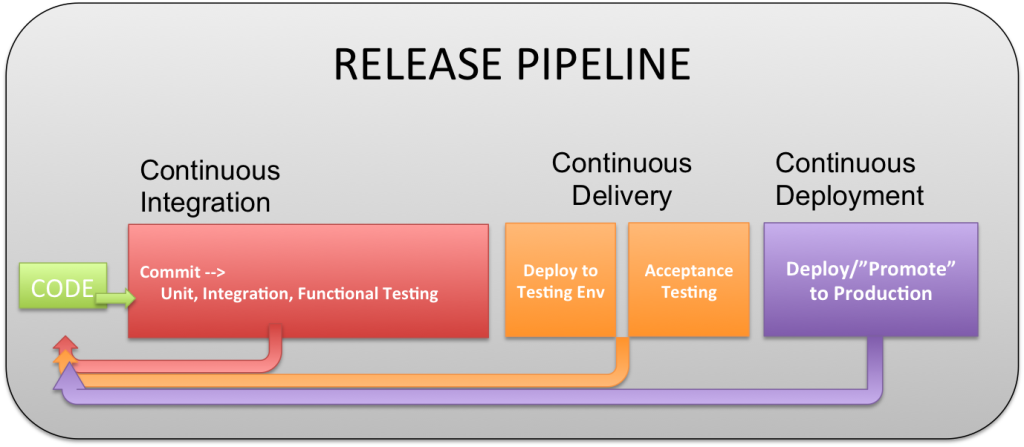
The beloved two letter acronyms are:

Continuous Integration (CI)

Continuous Delivery (CD)

Continuous Deployment (CD)

And this is how they fit into the software delivery chain:

[](http://devops.com/wp-content/uploads/2015/03/cicdcd.png)

This is one of many such diagrams, but the quick visual is useful for the rest of the conversations. What is not represented here well is the feedback loop that is fluid and never ending between all stages back to coding. Also note that this is the exploded view, so it shows all elements. But depending on the environment, everything could be collapsed into Continuous Deployment. The difference being the manual check points.

**Continuous Integration (CI)**

Continuous integration is a concept introduced by Martin Fowler in Extreme Programming development methodology, as one of its twelve practices. The evolved version of CI supposes the existence of four elements:

A testing strategy: Very important

“Flexible” infrastructure. Containers, VMs that can be spun up and down on a whim by developers and QA without the need for IT.

A version control tool like Git, CVS, SVN etc. with webhooks

An automated build and software release process  and tool like Jenkins, GO

A feedback mechanism: if a single test fails, the team is alerted the “build is broken” and they can fix the problems and reach a stable release, as soon as possible

A process for incorporating the source code changes, which can be manual or automatic (using a  continuous integration server)

Functional testing process and tools. This is often neglected, but in the best set up 75% of the entire process.

The advantages using CI

“Fail Fast” do high risk testing with no risk to production

Minimizes risks, by identifying code errors quickly

Because of automation reduces repetitive manual processes

Generates fast deployable software

Enables better project visibility to all Devops team members

Increases developer confidence in the software product, as feedback is provided early  (also the earlier assessment is provided, the less will cost to fix)

Sounds like the right direction to go, so what are the concerns that prevents teams from adopting CI?

Increased budget needed for maintaining the CI system, which is a wrong point of view as integration, testing, and deployment will be done anyway, and it’s not dependent on using or not CI

Too much change, which can be dealt with using an incremental approach: first builds and tests with a lower occurrence need to be added (like daily builds), and afterwards increase the frequency, as everyone gets more pleased with the results

Too many failed builds, which happens mainly when developers are not performing a private build before committing their code to the version control repository

Additional hardware/software costs. True, but they well worth the extra expense, for example a separate integration machine should be purchased,  which will compensate, later in the development lifecycle, the more costly process of finding problems. And the cloud solutions such as Sauce Labs, BrowserStack, Test Droid, etc to offload functional testing is a huge effort and time savings.

How is CI implementation done?

Start by selecting a CI environment ( cloud or on-prem) and take a number of factors into consideration in doing so:

Do you want it on-premise or to be hosted? Your selection will depend on wanting to run the infrastructure yourself or wanting to invest into a hosting solution managed by a 3rd party service provider.

What is the platforms you’re building for? If the hosted CI service is on Linux it may be suitable for Java, Ruby or Node.js, but not .NET,  for example AppHarbor is.

Scalability. As CI environments grow over time, they will require more hardware and more licensed software, so increased costs (a lot of CI packages are sold per build agent, so with an  upper limit number of builds that can run concurrently). And again the cloud solutions are low risk, and huge help here. They can run tests in parallel with brand new environments each time.

CI servers are usually compact and easy to setup, use, and configure, which is integrated with a wide variety of distributed version control systems, testing frameworks and  offers out of the box CI for any Ant, Maven or Gradle  project and Jenkins, which is free and open source, widely-used and well documented, used in combination with a large number of tools and technologies and which has a massive  community that supports it.

Once a CI server is selected a source control system should be considered, which will influence the build setup. Source Safe, Subversion, or Vault are more for development on a single branch with a related build, while Git or Mercurial  provide invaluable  support for development of features on multiple branches.

CI opened the door for other trends that are that are gaining momentum and sometimes are mistaken for CI: continuous delivery and continuous deployment.

**Continuous Delivery (CD)**

Continuous Delivery is about developing in a way that the software is always ready to be delivered to a production-like/stage environment and will run correctly.

Even if developers begin writing releasable code that satisfies all functional tests from  day one of the project, it does not demand when to release software, just that whenever the release occurs, the code is ready for the users.

The idea is to make a continuous-delivery a push-button system, which results in many daily deployments.

The infrastructure is seen as code and handled as an application changes to the infrastructure, are immediately integrated and tested, and, if it passes, deployed, as the ones for software applications, in an iterative delivery process.

The goal is to make visible the progress of builds from check-in to delivery to everyone. This will allow to see which changes have broken the application and which produced release candidates good enough for manual testing or release.

In terms of configuration management, there are four elements that are important to continuous delivery:

Ability to build, test and deploy the application in a fully automated way

Consistent management of the application’s deploy-time and runtime configuration

Possibility to create or apply a configuration change to every environment using a fully automated process.

All development done on a mainline, with larger features and restructuring implemented incrementally, so that the application is always kept working and ability to allow incomplete features to be made unavailable using configuration settings in the UI, if needed.

In case your organization it’s excited about continuous delivery implementation, it should be aware that attempting to build it over a non-existent or Unstable CI “layer” is a sure recipe for failure.

**Continuous Deployment (CD)**

Continuous deployment follows continuous delivery and deploys automatically all changes that passed the automated tests to production.

The main benefits for continuous deployment appear as a result of reducing lead time (the time elapsed between the identification of a requirement and its fulfillment), with two main consequences:

For each feature, after it is developed, an earlier return on investment, which lessen the need for large capital investments

Earlier evaluations from users on each new feature, which allows for A/B testing to establish which of several possible implementation is preferred by clients

Some of the fringe benefits are [Growth Hacking](http://devops.com/blogs/devops-growth-hacking-ways-rule-world/) and an extremely fast feedback loop from application usage to product management.

Continuous deployment has additional costs, as it relies on instrumentation, to ensure that new functionality does not result in bugs and also on infrastructure that allows effortlessly backing out new features when a defect has not been caught by automated tests.

Deciding to release a particular version of  the application should be done just by pushing a button, with the certainty that the release being deployed has gone through the entire pipeline successfully.

CI and the two CDs can provide an outstanding environment for the entire DevOps team, while the benefits greatly outweigh the challenges, and high quality can be achieved even for very large capacity and scale systems, while delivering value and innovation very fast in same time.

At the end of the day CI is for everyone Delivery is for most, and Deployment is for the rare few. Deployment requires massive applications and rapid feedback loops that are only for the high user, and transaction volume applications

**Complete step by step overview of CI/CD/CD**

**Continuous Integration**

Initially, Continuous Integration (CI) originated from XP practices. But as time has moved forward companies are using CI but not all XP practices anymore.

* We would like to start with explaining CI . What does it mean, when are you really practicing CI?
* CI is focused on integration of code and validation of the code on unit test and maybe integration test level. Your application is available in test.
* To achieve this, Continuous Integration includes:
* the developers check-in their code at least once a day
* the code is built at every check-in
* the code is automatically unit tested at every check-in
* everyone has access to the build and test reports
* the build is fast, so developers get feedback fast
* tests are executed on a scaled down version of the production environment
* deliverables are stored in a version controlled artifact repository
* deliverables are automatically deployed to a test environment after a successful build

If you have all this in place you are at a mature CI level and are able to take the next step to Continuous Delivery.

**Continuous Delivery**

Continuous Delivery (CD) it is the next step after CI. To get this in place you should do the following:

* Your software is deployable throughout its lifecycle
* Your team prioritizes keeping the software deployable over working on new features
* Everybody gets fast, automated feedback on the production readiness of their systems every time somebody makes a change to them
* You perform push-button deployments of any version of the software to any environment on demand
* There is a close, collaborative working relationship between everyone involved in delivery (often referred to as a "DevOps culture").
* Extensive automation of all possible parts of the delivery process has happened, usually using a Deployment Pipeline
* So what is the actual difference between CD and CI. To be able to deploy to production within a blink of an eye more is needed. Automation is the common part here. Taking the next step is to automate even more. Automate the regression test set so you are able to validate if the product is working properly without any manual testing. Beside that also automate the deployment of the application to production.
* The step to production is a call the business has to make. But when they make the decision it doesn’t take a lot of time and doesn’t cause a lot of stress. It will be the push of a button to put the increment in production.
* What are the benefits on top of Continuous Integration?
* Reducing the risk of a deployment to production. Because (almost) everything is automated (like tests, deployment, environment setup and deployment), the risk of anything going wrong is very small
* Progress of the development team is transparent. When the application is fully tested and deployed to production(-like) environment it is clear that it is really done. This way developers can’t say it is almost done and then need a week to deploy it to production.
* Final and most important one is that you get feedback fast. The risk of an idea that is not working for your customers is reduced because you can get feedback fast instead of months or years later.
* When you reached the Continuous Delivery level you are able to step it up a little bit further.

**Continuous Deployment**

Final step is Continuous Deployment (CD). The same abbreviation as Continuous Delivery and that’s maybe also why many people think of this as the same thing. But there is a slight difference between these two.

* When you are doing Continuous Delivery you determine when you really go to production. For Continuous Deployment you don’t. Every successful build will result in an automated deployment to production.
* So in both situation you can easily deploy to production but in case of Continuous Delivery it is a choice.
* Benefits on top of Continuous Deployment
* Every change goes to production directly. So the chance of missing a window of opportunity is very small.
* Feedback is returned even faster.
* By using feature toggles it is possible to deploy to production without using the new functionality. So parts that aren’t finished yet are already deployed to production. This way feedback about the deployment of that new part is received.
* Besides that it is also possible to let a small group of users test the new functionality again to receive fast feedback.

