Continuous Integration /Continuous Deployment

Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily.

Solve problems quickly

Because you're integrating so frequently, there is significantly less backtracking to discover where things went wrong, so you can spend more time building features.

Continuous Integration is cheap. Not integrating continuously is expensive. If you don't follow a continuous approach, you'll have longer periods between integrations. This makes it exponentially more difficult to find and fix problems. Such integration problems can easily knock a project off-schedule, or cause it to fail altogether.

Continuous Integration brings multiple benefits to your organization:

- Say goodbye to long and tense integrations
- Increase visibility enabling greater communication
- Catch issues early and nip them in the bud
- Spend less time debugging and more time adding features
- Build a solid foundation
- Stop waiting to find out if your code's going to work
- Reduce integration problems allowing you to deliver software more rapidly

More than a process

Continuous Integration is backed by several important principles and practices.

The practices

- Maintain a single source repository
- Automate the build
- Make your build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable version
- Everyone can see what's happening

- Automate deployment

How to do it

- Developers check out code into their private workspaces
- When done, commit the changes to the repository
- The CI server monitors the repository and checks out changes when they occur
- The CI server builds the system and runs unit and integration tests
- The CI server releases deployable artefacts for testing
- The CI server assigns a build label to the version of the code it just built
- The CI server informs the team of the successful build
- If the build or tests fail, the CI server alerts the team
- The team fixes the issue at the earliest opportunity
- Continue to continually integrate and test throughout the project

Team responsibilities

- Check in frequently
- Don't check in broken code
- Don't check in untested code
- Don't check in when the build is broken
- Don't go home after checking in until the system builds

Many teams develop rituals around these policies, meaning the teams effectively manage themselves, removing the need to enforce policies from on high.

Continuous Deployment

Continuous Deployment (CD) is closely related to Continuous Integration and refers to the release into production of software that passes the automated tests.

By adopting both Continuous Integration and Continuous Deployment, you not only reduce risks and catch bugs quickly, but also move rapidly to working software.

With low-risk releases, you can quickly adapt to business requirements and user needs. This allows for greater collaboration between ops and delivery, fueling real change in your organization, and turning your release process into a business advantage.

Why continuous delivery?

It is often assumed that if we want to deploy software more frequently, we must accept lower levels of stability and reliability in our systems. In fact, peer-reviewed research shows that this is not the case—high performance teams consistently deliver services faster and more reliably than their low performing competition. This is true even in highly regulated

domains such as financial services and government. This capability provides an incredible competitive advantage for organizations that are willing to invest the effort to pursue it.

The practices at the heart of continuous delivery help us achieve several important benefits:

Low risk releases. The primary goal of continuous delivery is to make software deployments painless, low-risk events that can be performed at any time, on demand. By applying patterns such as blue-green deployments it is relatively straightforward to achieve zero-downtime deployments that are undetectable to users.

Faster time to market. It's not uncommon for the integration and test/fix phase of the traditional phased software delivery lifecycle to consume weeks or even months. When teams work together to automate the build and deployment, environment provisioning, and regression testing processes, developers can incorporate integration and regression testing into their daily work and completely remove these phases. We also avoid the large amounts of re-work that plague the phased approach.

Higher quality. When developers have automated tools that discover regressions within minutes, teams are freed to focus their effort on user research and higher level testing activities such as exploratory testing, usability testing, and performance and security testing. By building a deployment pipeline, these activities can be performed continuously throughout the delivery process, ensuring quality is built in to products and services from the beginning.

Lower costs. Any successful software product or service will evolve significantly over the course of its lifetime. By investing in build, test, deployment and environment automation, we substantially reduce the cost of making and delivering incremental changes to software by eliminating many of the fixed costs associated with the release process.

Better products. Continuous delivery makes it economic to work in small batches. This means we can get feedback from users throughout the delivery lifecycle based on working software. Techniques such as A/B testing enable us to take a hypothesis-driven approach to product development whereby we can test ideas with users before building out whole features. This means we can avoid the 2/3 of features we build that deliver zero or negative value to our businesses.

Happier teams. Peer-reviewed research has shown continuous delivery makes releases less painful and reduces team burnout. Furthermore, when we release more frequently, software delivery teams can engage more actively with users, learn which ideas work and which don't, and see first-hand the outcomes of the work they have done. By removing the low-value painful activities associated with software delivery, we can focus on what we care about most continuously delighting our users.

If this sounds too good to be true, bear in mind: continuous delivery is not magic. It's about continuous, daily improvement—the constant discipline of pursuing higher performance by following the heuristic "if it hurts, do it more often, and bring the pain forward."

References:

- https://www.thoughtworks.com/continuous-integration
- https://continuousdelivery.com/