Continuous Delivery Principles

Uncover the basic principles behind continuous delivery with these getting-started guides.

Continuous delivery principles

Continuous delivery or CD for short is a collection of many prior successful agile and organizational best practices. CD focuses an organization on building a streamlined, automated software release process. At the heart of the release process is an iterative feedback loop. The feedback loop revolves around delivery of software to the end user as quickly as possible, learning from their hands-on experience, and then incorporating that feedback into the next release.

READ ON BELOW

Continuous delivery articles

ARTICLE

Continuous integration vs. continuous delivery vs. continuous deployment

CI is a practice that makes preparing for a release easier. CD may refer to "delivery" or "deployment," which are similar but not quite the same.

ARTICLE

Business value of continuous delivery

The business value of continuous delivery is not limited to tech darlings. CD improves velocity, productivity, and sustainability of software dev teams.

ARTICLE

Value stream mapping

Value stream mapping is an analysis technique that can help optimize your continuous delivery pipeline. Learn how and why this technique is used.

ARTICLE

Configuration management: definition and benefits

Learn about configuration management and its use in agile CI/CD software environments.

ARTICLE

DevSecOps: Injecting security into CD pipelines

You've heard of DevOps, but what is DevSecOps? Hint: it has to do with security. Learn more as we discuss this trend in continuous delivery.

ARTICLE

Feature branching workflows for continuous delivery

Using feature branching workflows in your continuous delivery pipeline keeps your most important branches in a clean and releasable state and allows develo

ARTICLE

Super-powered continuous delivery with Git

Now that Git has solved the pain of merging, learn more about branching workflows and best ways to do continuous delivery with Git and branching techniques

ARTICLE

Why agile isn't agile without continuous delivery

In order to reap the benefits of agile, you need to be agile through all phases of the software development lifecycle.

TUTORIAL

Continuous delivery tutorial

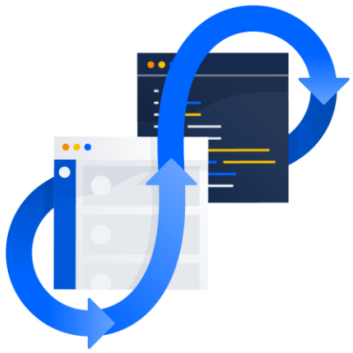
In this guide, we'll see how you can use Bitbucket Pipelines to adopt a continuous delivery workflow. Read on!

Try this tutorial →

[CONTINUED]

CD is an org-wide inclusive methodology that brings in non-engineering teams like design, product, and marketing. CD encourages developers to focus on delivering the end user product, whereas non-

and marketing. CD encourages developers to focus on delivering the end user product, whereas non-CD environments may incentivize “over the wall” behavior, in which the QA team becomes the primary user experience that developers are concerned with. The next sections will discuss specific principals that lay the foundation for CD workflows.



Repeatable reliable process

Organizational processes have their own development lifecycle, They usually start as manual checklists or “playbooks”. These playbooks are lists of tasks that are performed manually. Later they may be automated with software tools and scripts. Committing these playbooks to software scripts ensure that they are repeatable. If the checklist needs to be run again, a team member can execute the script. Reliability is gained when these playbook scripts are run consistently between environments. For example, the playbook for deploying code to a development or staging environment should mirror the production environment as closely as possible. This reliable consistency between environments and executions eliminates a whole class of consistency bugs.

Automate everything

Automation is a key value of CD. Human time is expensive and should be conservatively spent on creative exercises instead of tedious playbook task running. A manual process is not truly repeatable and reliable until it has been committed to code and is executable automatically on demand. Automated tasks can be composed together to create further levels of automation. Automate as much as possible; tests, releases, configuration changes and more.

Version control

A cornerstone of CD, version control is an absolute must for any serious software project. Version control enables a team of developers to efficiently collaborate on a shared codebase. Git is the most widely used version control system and a great companion for CD. Version control enables ‘undo’ functionality by allowing rollbacks to previous release candidates. In addition to code; configuration, scripts, databases, documentation should all be version controlled to track edits throughout history.

Build in quality

In CD, quality is not an afterthought that is kicked to the QA team. Quality is baked into every step of the release pipeline. The central feedback loop of CD is a constant re-examination of the quality being delivered to end users. New features are delivered with sets of automated tests that ensure new code is bug-free and meeting quality expectations. Project planning for new feature releases should include considerations around analytics, performance monitoring, and automated testing instrumentation tasks.

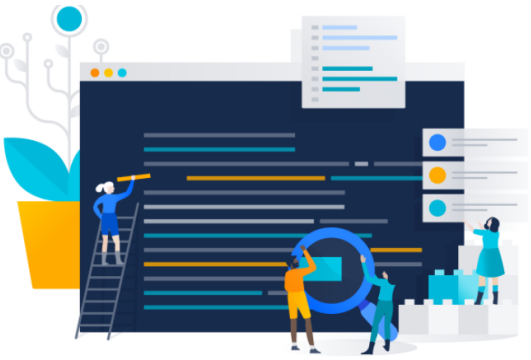
Do the hardest parts first

Painful, time-consuming or error-prone tasks compound over time. Painful tasks should be addressed as soon as possible to prevent a compounding loss of energy. Imagine a painful chore that takes 20 minutes to do and is run 5 times a week. That compounds to a 100 painful minutes a week. ~400 painful minutes a month, etc. Imagine you could address this chore and optimize it to prevent the painful time altogether, obviously, that would be a win.

“Do the hardest parts first” is also an exercise to help identify weaknesses in the organizational process. If there is a task that is procrastinated or actively avoided it is an indicator that it could be an area of improvement and should be actively pursued. Teams should regularly touch hard parts to stay familiar and keep them at the forefront of planning conversations.

Everyone is responsible

The entire organization should be focused and incentivized to ensure the end user deliverable is as high quality as possible. Product Managers should plan with attention to deployment and quality assurance. The Security team should be actively involved in the release process. QA team members should test development and staging environments with as much rigor as they would on production to catch any failures before eventual release. Developers should actively be planning for production release.



“Done” means released

Software companies are in business to deliver software to end users. There's no business if an app works solely on one developer's machine. “It works for me” is common red flag phrase that indicates a lack of awareness for the overall business goal and empathy for the end user. CD is entirely focused on shipping software to the end customer. Additionally, ‘done’ doesn’t mean when an individual team members contribution is done, but when the entirety of the team's contribution is complete.

Continuous improvement

The CD methodology doesn't stop at deployment. As discussed earlier, the feedback loop is the heart of CD. This means once a deployment has happened there is an additional phase of monitoring and measuring the effectiveness of that deployment. This phase again uses automated tools to measure the impact of the deployment on the end user. Obvious metrics like business revenue are monitored but more granular metrics like user conversion rates and engagement time are measured to identify correlation.

Continuous delivery value

Hopefully, the preceding sections have started to illustrate the high-level value adds of CD. At a macro level, CD promotes execution efficiency, cross team communication, product market fit, agility, and overall organizational transparency.

At a micro level, CD can be instrumented with measurements of explicit tracking metrics. Some valuable CD metrics might be:

- Time from new feature design phase to production release.
- How many production bugs encountered by users.
- Level of user engagement on new features.
- The frequency of new feature releases

In addition, CD can be used as a foundation to build organizational performance metrics like KPIs. Finally, bottom line business revenue and financial health is a great way to measure the impact of organizational practices.

Getting started with continuous delivery

With an understanding of the benefits and philosophy of CD, the next steps are to implement it. A good starting point is continuous integration. Continuous integration or CI is the precursor to CD. CI focuses on automating the workflow of code release. It does this through the use of automated code testing tools and quality assurance tasks. Once CI is in place CD processes can be built on top of it to deploy code to end users, and develop a feedback loop which will steer future releases.