**Start using Continuous Delivery**

[Continuous Delivery](https://octopus.com/devops/continuous-delivery/) (CD) is a concrete set of practices that make up the best-known way to deliver software. This isn’t just a bold claim, as research has shown Continuous Delivery predicts high performance for software delivery and against organizational goals.

**When to use Continuous Delivery**

For most organizations that develop software, delivering software is a constraining factor in achieving their goals. If software delivery is a constraint, Continuous Delivery will help you deliver changes safely and quickly. Continuous Delivery drives software delivery performance and organizational culture, improving your chances of hitting commercial and non-commercial goals.

You might decide to use Continuous Delivery to reduce:

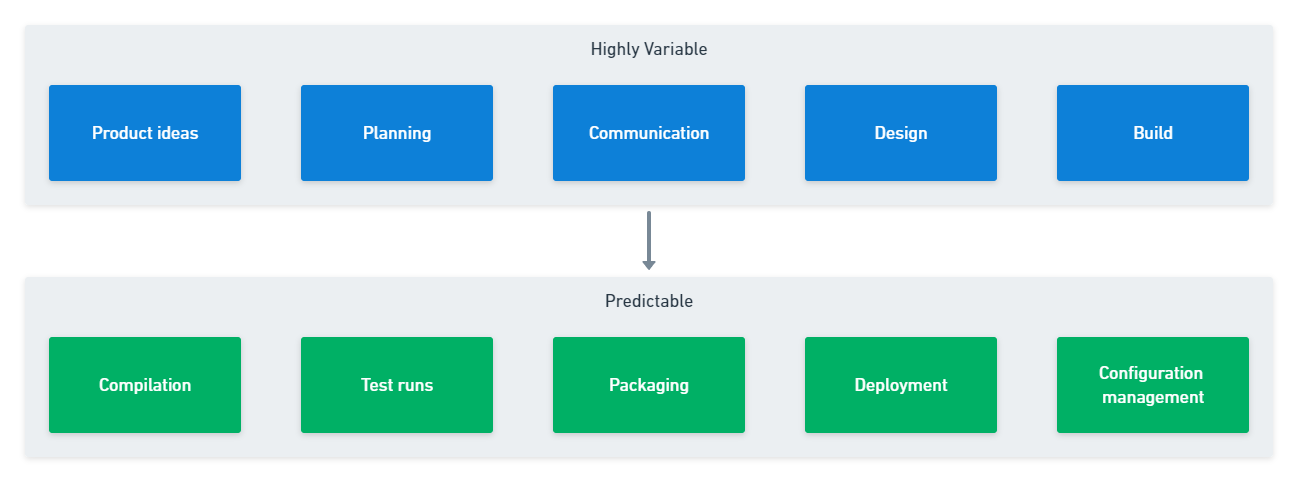
* The time it takes to get a change from a developer’s machine to the production environment
* Some of the pains in your deployment process.

No matter your motives to improve how you deliver your software, you’ll get benefits in many other areas, too.

**Where Continuous Delivery fits**

If you use the commit process to split your software development lifecycle into two phases, you’ll find different characteristics on either side of the commit stage:

1. Before the commit stage, things are complex and variable
2. After the commit stage, the process is fixed and predictable



**Pre-commit phase**

The pre-commit phase is like designing a chair. There are many factors you need to consider and many good options for each decision. Your possibilities are open-ended, and you’ll use a process to make decisions about geometry, materials, and dimensions that all impact the final product. There’s no single correct answer but many trade-offs.

In software, you need to generate and collect ideas. You need to rank or prioritize these ideas, which sometimes involves working out whether they’re worth the investment. You then need to share selected ideas with people who’ll design and build them.

Your ideas can spend months moving through this phase, or they can speed through in less than a day. The idea might need hundreds of changes or just a change to a single line of code. It varies per work item.

Continuous Delivery doesn’t cover the pre-commit phase. You don’t need to change your product management or software development process to use Continuous Delivery. Adaptive approaches based on Lean and Agile align to the same core ideas, but you can use other software delivery models as long as you can work in small batches and commit changes frequently.

**Post-commit phase**

The post-commit phase is like manufacturing the chair you designed. You made all the decisions and the design is final. You just need to make each part and assemble them. You probably have a checklist that details the components and how they all fit together. This phase is less about decision-making and more about getting a chair to the customer.

The same is true for your software. You have a series of steps that you must do in the correct order to progress the change through to your production environment. We sometimes call the approval stages the software must pass through to go live *quality gates*. There are also practical steps that allow validation of software, like deploying the correct software version to a test environment.

Continuous Delivery refers to this set of steps as the *deployment pipeline*. You can use the tools and practices of Continuous Delivery to deploy changes faster and more frequently. To do this, you automate most of the stages and approvals.

**Who uses Continuous Delivery**

There are lots of companies using Continuous Delivery. Household names like Google, Amazon, and Meta (Facebook) use Continuous Delivery. Companies like Paddy Power, Wells Fargo, and HP’s LaserJet firmware team also use it. The [Guardian newspaper](https://www.theguardian.com/info/developer-blog/2015/jan/05/delivering-continuous-delivery-continuously) in the UK increased its rate of deployment by 1,000x between 2011 and 2014 by adopting Continuous Delivery.

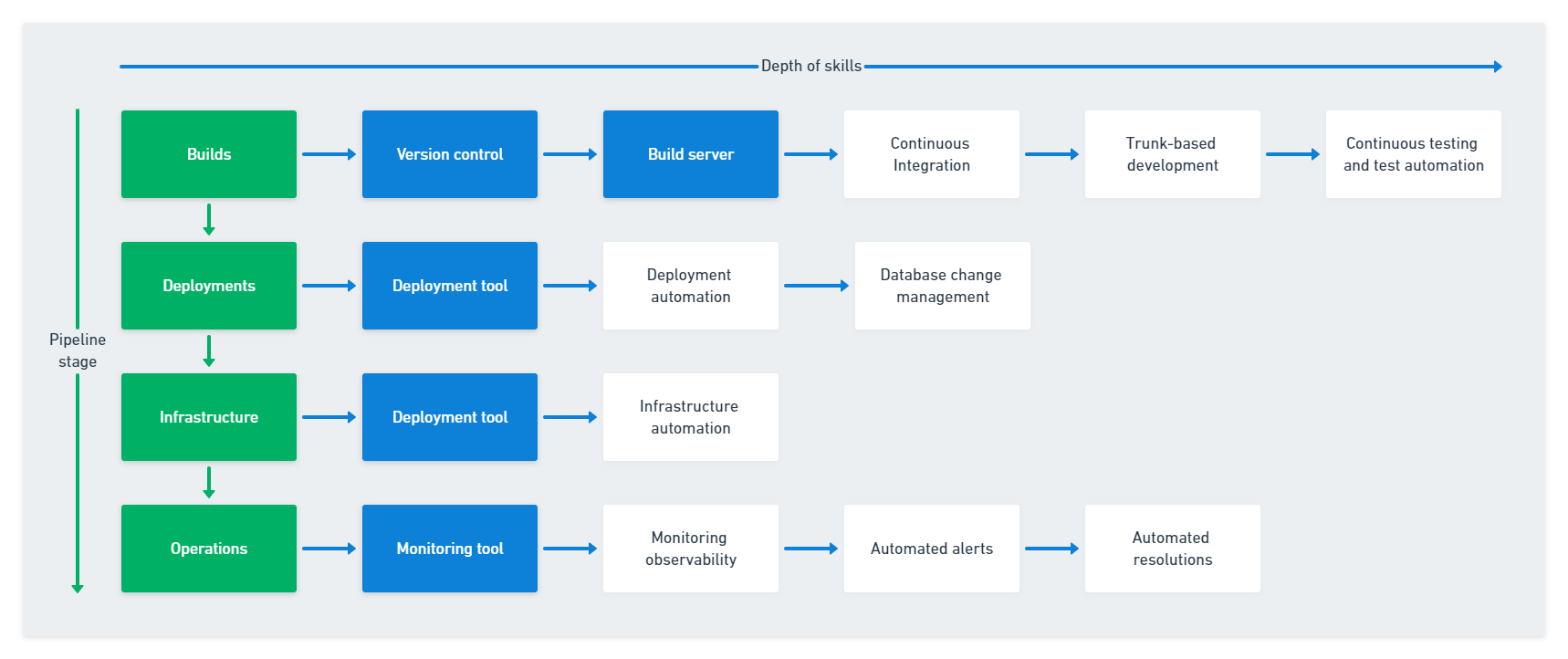
When you review case studies, it’s best to focus on examples from companies that are similar in size and complexity to your development organization. This allows you to avoid replicating practices that may not be relevant to your organization.

[Meta](https://www.metacareers.com/areas-of-work/engineering/) has around 25,000 to 30,000 developers (compared to about 3,200 at [Target](https://www.wsj.com/articles/target-cio-helped-retailer-find-its-tech-groove-11579705200)). With hundreds of changes made each hour, Meta adopted a [quasi-continuous release cycle](https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/) to cope with the high-velocity of change. For most organizations, this process would be a bad idea, but for Meta it’s necessary to handle the scale of change.

**How to start your Continuous Delivery adoption**

As part of a Continuous Delivery adoption, you should choose a smaller system to start with. An internal system that provides a non-critical function to a small number of users will be a less risky learning environment than your core public application. You can use your chosen system to learn the tools, develop the necessary skills, and apply what you learned to other systems.

The diagram below has a minimal set of tools and skills that you need to create a solid [deployment pipeline](https://octopus.com/devops/continuous-delivery/what-is-a-deployment-pipeline/). It shows related tools and techniques from left-to-right, with each row expanding the deployment pipeline into the next toolset. You can decide to adopt multiple tools in parallel, or you might prefer to progress row-by-row to gain confidence with each toolset before you move to the next one.



Your deployment pipeline may have other requirements that mean you need more tools. For example, you might perform security or load testing using various tools. You also need to include these tools in your deployment pipeline. If the tools need to be set up and run manually, try to find a way to automate them.

**Continuous Delivery tools**

Continuous Delivery involves a mix of tools and processes. If you use the right tools correctly, you can increase your return on investment. The best-in-class tools for Continuous Delivery provide mechanisms to quickly and safely move changes to your production environment.

You need four critical tools to increase your chance of successfully automating your deployment pipeline:

* Version control, like Git, GitHub, or BitBucket
* Build server, like TeamCity, Jenkins, GitHub Actions, or Atlassian Bamboo
* Deployment automation, like as Octopus Deploy
* Monitoring, like DataDog, New Relic, Splunk, or Dynatrace

Some of these tools may already be in place. Use the adoption diagram to decide when to add those you don’t already have.

You can increase the value of the tools by applying the proper practices.

**Continuous Delivery practices**

There are several practices listed on the adoption map. There are even more if you expand into a full [DevOps](https://octopus.com/devops/) approach. Introducing and mastering each skill can take time, so you need to balance skill building with automating stages further along the deployment pipeline.

* Continuous Integration
* Trunk-based development
* Continuous testing and test automation
* Deployment automation
* Database change management
* Infrastructure automation
* Monitoring observability

Rather than expand on every practice listed, let’s focus on the key practices of Continuous Integration and trunk-based development. The practices occur at the commit stage, which is the entry point for your deployment pipeline.

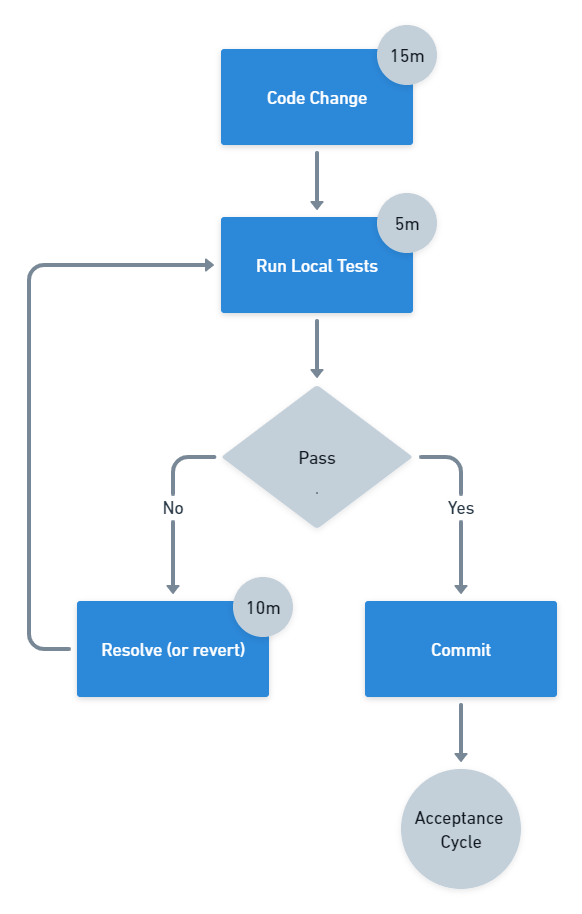
**Key practice: Continuous Integration**

Continuous Integration refers to the frequent commit of changes to the main branch in version control. Because Continuous Integration triggers the deployment pipeline, it’s a critical practice. This is why many people refer to Continuous Delivery as **CI/CD** (Continuous Integration/Continuous Delivery).

The commit cycle should happen after around 15 minutes of coding. The feedback from your changes should be available in about 5 minutes.

* If your tests take much longer than 5 minutes, it discourages people from running them often (and from committing changes frequently)
* If your tests are taking less than 5 minutes, you may be missing a chance to get early feedback from your tests. You can look for opportunities to add tests or move them from a later testing stage

The chart below shows the process for the commit cycle:



The commit stage in the diagram is where Continuous Integration happens. You can use a specific set of steps to do this well.

1. Merge changes from the main branch into the local copy
2. Build and run tests locally to validate the version is worth committing
3. Commit the change to the main branch
4. Have the build server build and test the code for the updated main branch

You should integrate your code changes with the main branch frequently. Any branches you have should be short-lived. This is called **trunk-based development**, where you have no more than three branches and merge them into the main branch in less than a day.

While it’s possible to keep a branch updated with changes to the main branch, this is no longer useful when the main branch itself is not up to date.

**Summary**

All organizations that depend on software delivery will benefit from Continuous Delivery. Software teams of all sizes for software and firmware delivery use it. Use the deployment pipeline to focus your automation and improvement efforts to get software to your users quickly and safely.

The big tech companies may do some things differently, but this is because they operate at a massive scale. Be wary of copying organizations with very different problems to solve. Instead, look for examples from organizations working at a similar scale to yours.

You can use the adoption diagram to decide when to focus on increasing skills versus extending your toolset to automate more of your deployment pipeline. The practice of Continuous Integration is an excellent place to start, along with shaping your version control and build tools.

There are several [benefits of Continuous Delivery](https://octopus.com/devops/continuous-delivery/why-adopt-continuous-delivery/), whether it’s reducing your deployment pain or performing better against your commercial goals.

There are no situations in software delivery where a smooth and automated deployment pipeline would fail to bring benefits. Using Continuous Delivery within a broader DevOps approach will amplify its positive effects on your team and organization.