The IT Manager's Guide To Continuous Delivery

A practical guide to accelerating your release process with Continuous Delivery

INTRODUCTION TO CONTINUOUS DELIVERY

Although startups like Netflix and Etsy, have been using Continuous Delivery for some time to bring software to market faster, the practice is increasingly becoming mainstream. Continuous Delivery is a key part of what allows organisations to innovate and implement technology change at pace.

Although it’s essential that businesses can turn good ideas into marketable features, many IT managers are daunted by the combination of processes, tools and mindset change needed to bring Continuous Delivery into reality.

In addition, many have found the road to Continuous Delivery challenging. Sometimes the allure of Continuous Delivery hasn’t matched the reality.

This short eBook aims to help IT managers that are on the path to implementing Continuous Delivery. By understanding the steps that you need to take to get there and the tools available, you’ll be on the road to accelerated time to market, reduced cost, improved ROI and reduced risk.

WHAT IS CONTINUOUS DELIVERY?

Thoughtworks’ Martin Fowler defines Continuous Delivery as “a software discipline where you build software in such a way that the software can be released to production at any time.”

Many organisations use their Continuous Integration efforts as a “base camp” for Continuous Delivery. In organisations looking to adopt Continuous Delivery, Continuous Integration has already been adopted and is running, usually in conjunction with the implementation of Agile development methodologies.

If you’re already using Continuous Integration, you’ll be familiar with the concepts. Continuous Integration is the process of integrating code into a shared repository many times in a day. Every time that code is committed to a shared repository, it is verified by an automatic build test and run. This ensures that new commits don’t cause existing features to break.

Continuous Delivery takes this automation to the next level, by automating deployment. When code has been verified by an automatic build test, products are then rolled out. New features are usually deployed through a progression of environments from development systems, through one or more test environments, before final promotion and release to production.

HOW CAN ORGANIZATIONS BENEFIT FROM CONTINUOUS DELIVERY?

An organisation that is operating Continuous Delivery is able to delivery working software to market faster. A 2013 survey by Puppet Labs found that development teams who are operating using Continuous Delivery are able to deploy code 30 times faster than those without it. Many large organisations are able to deploy new versions of their software every day, or have moved to achieving multiple releases per day. The key benefits to adopting Continuous Delivery are:

1. Bring new features to your users faster
2. Discover, isolate and fix bugs faster
3. Get faster feedback from your users
4. Relieve stress on your developers
5. Keep up with the competition
6. Increase release quality
7. Reduce the risk of individual releases

KEY PRINCIPLES FOR CONTINUOUS DELIVERY

David Farley has defined eight principles for Continuous Delivery, in the book he co-authored with Martin Fowler. The book goes into great detail about how Continuous Integration has evolved, and how Continuous Delivery can be achieved through “build pipelines.” Farley argues that Continuous Delivery is about using automation in a smart way to create a repeatable and reliable process for delivering software. In order to achieve Continuous Delivery, almost everything has to be automated. Manual steps will simply get in the way, or become a bottleneck. Farley’s eight principles are:

1. The process for releasing software must be repeatable and reliable
2. (As a result) everything must be automated: a process which is manual can never be described as repeatable and reliable.
3. If something’s difficult, do it more. Although on the surface this sounds counter-intuitive, Farley is arguing that you should not put off doing difficult things. If you’re doing things that are tricky, that’s a sign that you should be automating them.
4. Keep everything in source control.
5. Done means “released” - a project is “owned” until it is in the hands of the user and working.
6. Quality should be built in. Certainly never accelerate broken processes, but also build good, targeted quality metrics into your project. Projects that have these are easier to maintain and more reliable.
7. The release is everyone’s responsibility. A programme which is sitting on a developers laptop can never make any money for the company. Companies make money by bringing software to market, therefore the release process is in everyone’s interest. Project managers should plan projects with attention to deployment. Teasers should test for deployment defects as well as code defects (and these steps should be automated).
8. Continuous improvement. Continuous improvement means that your system is always evolving and is therefore always easier to change.

CONFIGURATION MANAGEMENT STEPS FOR CONTINUOUS DELIVERY

As we've detailed in the previous section, Continuous Delivery entails developing in a way that code is always ready to be delivered to a like-live stage and capable of running correctly. The idea behind Continuous Delivery is to make software delivery a push-button system, which means that deployments can happen many times daily.

Typically in a Continuous Delivery architecture, the infrastructure is seen as code, which means that code is written to manage configurations and automate provisioning of infrastructure.

With Continuous Delivery (and particularly with patterns such as full stack deployment), the infrastructure is handled as an application changes to the infrastructure, is handled and tested, and, if it passes, deployed.

The aim of Continuous Delivery from a configuration management perspective is to make the progress of builds available to everyone. This means that everyone can see which changes have broken the application, and which have produced release candidates.

From a configuration management perspective, there are four main
sysadmin activities to Continuous Delivery:

- Being able to build, test and deploy the application in a fully automated way
- Consistently manage the application's deploy time and runtime config
- Being able to apply a configuration change to every environment using a fully automated process.
- All development is done on a mainline, with larger features implemented incrementally, so the application is always in a releasable state.

Continuous Integration is a prerequisite for Continuous Delivery - without it, you simply can't guarantee that software will always be in a releasable state.

HOW TO IMPLEMENT CONTINUOUS DELIVERY

It's not going to be possible to put Continuous Delivery in place overnight. One of the biggest mistakes organizations make is rushing to the finish line.

It's important to ensure in particular that Continuous Integration (CI) is in place first, followed closely by automating the deployment of the software assets to the initial development systems. If CI is a pre-requisite to Continuous Delivery – deployment automation is its foundation. It is imperative that we can reliably release updates before we considering accelerating the process.

That's why, when we're working with a client, we tend to insist on them working through each of the following steps before continuing onto the next. We're also making certain assumptions here, such as having some form of version control in place.

1. DEFINE YOUR RELEASE

In terms of your release, what does "done" look like? It sounds like a simple question, but it's important before tackling Continuous Delivery to think about what defines a release. Are you going to push a new release every time a change is made to the code?

Maybe you'll take the approach of creating specific branches for new releases? What processes are in place for deciding what makes it into a release and what gets left out?

In the era of the agile enterprise, all of this knowledge should be captured, stored and maintained, so all team members understand what constitutes a release.

2. BUILD AUTOMATION

Build Automation is especially useful when you have a large number of simple, manual steps that you have included in your release cycle. Build Automation connects your CI tool (such as Jenkins, Bamboo), with your source control, so it can run and monitor builds for each release.

When you're starting to do Continuous Delivery, you should run your build tool in the background or overnight. This will enable you to understand how your process fits with load and your team's workflow.

Then, when you are comfortable, you can add more items to the build.

HOW TO IMPLEMENT CONTINUOUS DELIVERY (CONTINUED)

3. CODE ANALYSIS

With Build Automation in place, it's time to use Code Quality tools to detect things like code smell. Code analysis tools enable you to analyse your code syntax and semantics, spell check documentation, verify dependencies and more. Common tools include Sonar, Pylint and PMD - but there are hundreds of options available.

4. TEST AUTOMATION

Many of you will have already been using some testing tools before you even started at looking at Continuous Delivery. The tools that are most appropriate for you will depend on your development environment. Your test automation tool should produce a report of the tests that ran, and the results of those tests. These reports can be read automatically by your Continuous Integration tool, and highlight where failures have occurred.

If the tests have succeeded, the tool will pass the process on to the Deployment Automation tool. If they have failed, it's up to you to define what the next step in the process looks like.

5. DEPLOYMENT AUTOMATION

Once your software has been tested, it needs to be deployed. For those building enterprise software, this will likely mean to a staging environment where your team and trusted users can start testing it.

It's also important to define what is being deployed. Many people are increasingly opting for "full stack" deployments, where the virtual machine is replaced with a fresh one, the OS is reprovisioned, and any dependent services are recreated.

Organizations can also significantly reduce their IT infrastructure by only running infrastructure (be it dev, test and prod) only when required.

6. MONITORING

When you have deployed your application, you need a way of ensuring and verifying that it is still performant.

There are numerous metrics that you can use for this: including usage of the new feature, changes in company revenue, or application speed.

More efficient monitoring also means that you can adopt practices such as autoscaling: scaling resources up and down as required.

BEST PRACTICES FOR CONTINUOUS DELIVERY

Continuous Delivery practices can help you create high quality software in an efficient way. Here are some recommendations for how to best go about implementing Continuous Delivery.

1. IMPLEMENT CONTINUOUS DELIVERY STEP BY STEP

You'll want to implement Continuous Delivery in a gradual way. Avoid using a big bang or waterfall approach. Look for quick wins and low hanging fruit, and automate steps that will save you the most time and improve software quality by reducing the number of errors.

Continuous Delivery implementations work best when you implement them step by step. If you're working with an agile project approach, plan a small amount of Continuous Delivery work for each sprint. This will give team members who are less used to working in a Continuous Delivery way a chance to get used to this way of working. This gets you more buy in during the implementation phase.

2. FOCUS ON TESTING

Test and review everything - and often! Continuous Delivery involves the automation of the entire process of software delivery, so any error can have serious consequences. With continuous delivery, the frequency and delivery of your release cycle will increase.

To protect the quality of your software, you will need to put a lot of automated testing in place, because testing manually simply will not be feasible.

3. VERSION EVERYTHING, COMMIT OFTEN

All of your product knowledge should be in your version control system. The specific tool that you use for version control is less important than you using it. Your version control system / repositories should contain:

- Design documents
- Code
- Database design
- Database scripts
- Environment configuration
- Test scripts
- Deployment tools
- Deployment scripts

The only thing that should not be stored in version control is sensitive data, or environment dependent settings. Commit frequently and integrate early.

BEST PRACTICES FOR CONTINUOUS DELIVERY

4. PEER REVIEW EVERYTHING

Make use of your peers to review everything you make. Peer review is vital to increasing code quality, and helps ensure that knowledge is spread throughout your team.

Your team members will be able to give feedback on your methods and approach, ensure that your structure is compliant with architecture guidelines, and may find things that you have overlooked.

Alongside this review, there is added value in the knowledge transfer of the things that you have created. By reviewing your code, your team member will become familiar with your deliverable, and will save them unpicking your code when they later come to work on it.

5. ACCEPT FEEDBACK

When someone is providing you with feedback, accept it with an open mind.

Create a culture of accepting feedback as an important step towards improving the product or the process. Feedback can be an important opportunity to learn from your experiences and improve yourself and the team.

6. CONSCIOUSLY INCREASE RELEASE FREQUENCY

When you are doing Continuous Delivery, look for opportunities to speed up your release frequency. By speeding up your release frequency, you are continuously testing your release process, and making smaller changes. When the size of changes is small, it's easier to trace faults, making debugging easier.

7. ASK SOMEONE ELSE TO TRY YOUR IMPLEMENTATION

Always be asking other people to try your implementation. By handing over your software solution to someone else, you'll reveal differences in interpretations and test your implicit understanding of preconditions.

Letting someone else test your software will validate its documentation, functioning, structure and efficiency. It's a great test of the software that you're bringing to market.

8. THE PIPELINE MUST FLOW

The top priority for Continuous Delivery teams is to deliver software continuously. This can only be achieved when the entire pipeline is functioning. If there is a bottleneck in the Continuous Delivery pipeline, this has to be fixed as a top priority. When the pipeline is functioning, you do not have proper safeguards and controls on your codebase. Adding more code to this situation will only make the situation worse.

9. USE THE CLOUD

Iterative development, intensive testing, and frequent deployment requires that you are able to reset your development environment.

It’s no use going to your test environment and finding that it is unusable. Cloud tools can provide you with the ability to use like-live environments in their original states. This allows frequent testing of release and deployment, increasing software quality.

Cloud and virtualization tools enable you to use snapshots or scripting tools to recreate environments from scratch.

10. CONTINUOUS DELIVERY IS NEVER ONE AND DONE

Implementing Continuous Delivery is never a one and done process. It’s an on going process that is about evolving and improving with each step. It’s not one person’s responsibility to drive automation forward. It rests with the whole team to constantly be looking for ways to make software delivery as efficient as possible.

SUMMARY

BENEFITS OF CONTINUOUS DELIVERY

Continuous Delivery enables organizations to:

- Bring new features to users faster
- Discover, isolate and fix bugs faster
- Get faster feedback from users
- Relieve stress on developers
- Keep up with the competition
- Increase release quality
- Reduce the risk of individual releases
- How to Implement Continuous Delivery

Continuous Delivery requires change and continuous improvement from teams across your organization. CD requires a focused investment in tooling, hardware and people. All of this takes time, but all of it is possible.

IMPLEMENTING CONTINUOUS DELIVERY

Each and every organization moving to Continuous Delivery faces its own development bottlenecks. However, a successful shift to Continuous Delivery should be based on your own business needs and your ability to take on the following elements:

- Required changes in your setup from start to finish, especially collaboration aspects for everyone involved in the development process
- Investment and adoption in new tools and infrastructure to make sure you can build, test and deploy as required.
- The ability to respond to immediate feedback through each iteration, which enables you to tweak your product to your users real needs.

It doesn’t matter if you’re building a customer facing application that is used by millions of users, or core internet infrastructure that you never see, the above elements all play their part in a successful transition to Continuous Delivery.

The transition can lead to a satisfied user base, and once you’ve got a happy and delighted bunch of users onboard, every effort in making that move to Continuous Delivery will be worth it.