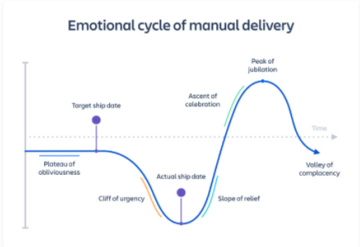
Business value of continuous delivery

Continuous delivery improves velocity, productivity, and sustainability of development teams.

BY JUNI MUKHERJEE

Why continuous delivery?

What emotions does the word "release" trigger in you? Relief? Elation? A fist-pumping sense of accomplishment? When new features are finally out to customers and bugs are fixed, everyone's happy, right? Well, the dark secret in many organizations is that shipping a release takes a huge amount of effort. If your team is still living with manual testing to prepare for releases and manual or semi-scripted deploys to perform them, your feelings may be closer to "dread" and "blinding rage".



That's why software development has moved towards continuity, through methodologies like agile and DevOps. In the continuous paradigm, quality products are released frequently and predictably to customers. Therefore, the ceremony and risk around releasing are reduced. If you're relying on your pipelines daily, you will notice (and resolve!) its deficiencies much quicker than if they flow once every few weeks or months. That is, reduce difficulty by increasing the frequency of your product releases.

The emphasis on continuous delivery, including continuous integration, continuous testing, constant monitoring, and pipeline analytics all point toward an overall trend in the software industry - helping teams to react to market changes. Make no mistake - CD is not the exclusive domain of "unicorn" companies and tech darlings. Every team – from the humblest start-up to the stodgiest enterprise – can and should practice continuous delivery.

This article takes a look at the business case for making this switch. It will discuss the work that lies ahead and the benefits it will yield to ship software using CD pipelines.

Top business benefits of continuous delivery

Continuous delivery improves velocity, productivity, and sustainability of software development teams.

First off, velocity

Automated software delivery pipelines help organizations respond to market changes better. The need for speed is of utmost importance to reduce shelf time of new features. With a low Time2Market, organizations have a better chance to outmaneuver their competition and stay in business.

Remember that speed by itself is not a success metric. Without quality, speed is useless. There is no value in having continuous delivery pipelines shoot erroneous code into production at speed.

RELATED TUTORIAL

Continuous Delivery Tutorial

Try this tutorial →

SUBSCRIBE

Sign up for more articles

Email

email@example.com

Subscribe

So, in the world of continuous delivery, velocity means responsible speed, and not suicidal speed.

Second, productivity

Productivity translates to happiness, and happy teams are more engaged.

Productivity increases when tedious and repetitive tasks, like filling out a bug report for every defect discovered, can be performed by pipelines instead of humans. This lets teams focus on vision while pipelines do the execution. And who doesn't want to delegate the heavy-lifting to tools?

Teams investigate issues reported by their pipelines and once they commit the fix, pipelines run again to validate whether the problem was fixed and if new problems were inadvertently introduced.

Third, sustainability

Businesses aim to win marathons, not just sprints. We know that cutting ahead of the pack takes grit. Consistently staying ahead of the pack can be even harder. It takes discipline and rigor. Working hard 24/7 will lead to premature burnouts. Instead, work smart, and delegate the repetitive work to machines, which by the way don't need coffee breaks and don't talk back!

Every organization, whether or not a tech company, is using technology to differentiate. Automated pipelines reduce manual labor and lead to eventual savings since personnel is more expensive than tools. The steep upfront investment can cause concern to inexperienced leadership, however, well-designed pipelines position organizations to innovate better and faster to meet their customers' needs. CD provides the business with more flexibility in how it delivers features and fixes. Specific sets of features can be released to specific customers, or released to a subset of customers, to ensure they function and scale as designed. Features can be tested and developed, but left dormant in the product, baking for multiple releases. Your marketing department wants that "big splash" at the yearly industry convention? With continuous delivery, it's not only possible, it's a trivial request.

Top challenges of continuous delivery

While we firmly believe continuous delivery is the right thing to do, it can be challenging for organizations to design and build resilient continuous delivery pipelines. Because CD requires a large overhaul in technical processes, operational culture, and organizational thinking, it can often feel like there's a large hurdle to getting started. The fact that it requires a hefty amount of investment in a company's software delivery infrastructure that may have been neglected over the years, can make it an even more bitter pill to swallow.

There are many problems that organizations face and the following three are the most common pitfalls - budget, people, and priority.

Budget: Is yours too low?

Construction of continuous delivery pipelines consumes your best people. This is not a side project whose cost can be shoved under the carpet. I have always been surprised at how some organizations start out by allocating junior members and by cutting corners on purchasing modern tools. At some point, they course correct and assign their senior architects to invest in architecture decoupling and

senior architects to invest in architecture decoupling and resilient continuous delivery pipelines.

Don't lowball. Based on your vision, set aside an appropriate amount of funds to make sure execution is uninterrupted. Deliver a continuous delivery pipeline MVP (minimum viable product) and then scale it throughout your organization.

Do you have forward-thinkers?

Even when you have budget, at the end of the day, execution could be a people problem.

Teams should fearlessly automate themselves out of their jobs, and move on to new projects. If you have people who are apprehensive of automated agents performing tasks they were otherwise doing manually, you are housing the wrong people.

In case you feel stalled, shift gears. Know how to give your team a car when all they have asked for is a faster horse! Jumpstart with the help of experienced champions who will see you through this initial hump. People, after all, are your greatest assets and train them to do the right thing. Make it easy to do the right thing, and hard to do the wrong thing, and you will be pleasantly surprised at the outcome.

Lack of priority

"Let's stop the line and build continuous delivery pipelines!" said no product owner ever.

In their defense, they are focused on cutting ahead of competition with new features that wow the world. At the same time, you know you have a problem if in every sprint planner, pipelines are weighed against product features and are traded off.

In some product backlogs, pipelines, if they appear at all, hang on for their dear lives near the bottom. Near sighted leadership classify work related to pipelines as expenses, rather than investments that will stand the teams in good stead. They remain in denial over the long-term damage they create, and unfortunately, sometimes get away with it.

Pipelines are hygiene. If you want to stay in business, ask yourself "Is hygiene important?". You bet it is!

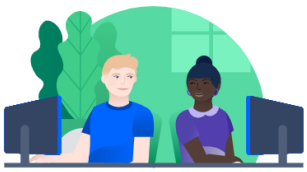
Continuous delivery metrics

OLTP (online transaction processing) and OLAP (online analytical processing) are two well known techniques in the industry. Both concepts apply to continuous delivery pipelines and help generate insights that steer organizations in the right direction. Let's see how.

Pipelines see hordes of transactional data

Imagine a typical day in a software development team's life. The team commits a feature that business prioritized, commits tests for that feature, and integrates deployments with the continuous delivery pipeline, so that every change deploys automatically. The team realizes that the application got sluggish after adding this new feature, and commits a fix for the performance issue. The team also adds performance tests to make sure that bad response times would get caught before promoting the application from test to staging.

Think of each of those commits as a transaction. And that's how software development teams roll - one transaction after another - till a product is born that wows the world. And then off they go again. Multiply these transactions across all engineers and teams in your organization, and you have hordes of transactional data at your disposal.



This is a good segway into the next section on pipeline analytics and how to make the most out of that transactional data.

Let's analyze the pipeline's transactional data

Could we analyze the transactional data to extract nuggets of information? Sure, we can!

Like with all transactional data, the sheer volume prevents us from making any sense. That's why we should aggregate, and perform analytics to glean insights into our organization. Analytics help us see the forest through the trees, and here are three examples how we improved our practices from pipeline analytics and insights.

Out of the hundreds of deployments that happen every week, we learned that the number of deployment failures of application A are thrice as high as those of application B. This discovery led us to investigate application A's design choices on environment stability and configuration management. We learned that the team used flaky virtual machines in their data center to deploy, while application B was containerized. We prioritized an investment into immutable infrastructure and checked back in a month to make sure the return on that investment was good. Sure enough, it was. What can be measured, can be fixed.

Another example is when we learned that application B's static code analysis errors have steadily trended upward over the last few quarters. This could mean that the team behind application B needs to be (re)trained to write better code. We also discovered that the static code analyzers reported false positives, which means they flagged coding violations when there were none. So, we upgraded their analyzer to a well-known industry-standard tool and the false positives reduced to some extent. We organized a coding workshop where we discussed and resolved the legitimate static analysis errors. By the end of it, the team was humming along again.

Another interesting insight was that application A's unit tests had less code coverage than application B and C, and yet application A had the least number of production issues in the last year. Writing unit tests and measuring code coverage are fine. Overdoing that exercise is unproductive for the team and useless for the customers. Lesson learned.

Key Performance Indicators (KPIs)

We can not rely on opinions when it comes to steering the organization in the right direction. First, we need to define KPIs based on what success looks like. Second, we need to make data-driven decisions by analyzing KPIs across months, quarters, and years.

Organizational KPIs vs. departmental KPIs

Many times we have seen individual departments define their own success metrics. It's a good thing for departments to understand what success looks like for them, as long as those metrics tie in to the organization's goals.

Failures on test vs. staging vs. production

A few organizations require developers to own the test environment, QA to own staging, and Ops to own production. Instead of developers getting buried under code coverage reports for unit tests that run in the test environment, it is important for them to step back and look at the big picture on all environments - whether they own it or not.

The percentage of failures in staging due to performance tests could be high, and it could be due to incorrect performance benchmarks or sluggish code. A comparative analysis might show that pipelines fail the most on integration smoke tests in production, and that would warrant an investigation. The root cause could be real bugs in the product or could be buggy test code, inaccurate test data, incorrect test configuration, misunderstandings between product and engineering, and the like.

Further digging could reveal that incorrect test configurations are rampant, and you could prioritize fixing those issues to fix frequent integration failures. Additionally, developers owning their code all the way to production is

also in accordance with the DevOps paradigm.

Stability index

Once we define KPIs, it is critical for us to understand if a single KPI has bias and skews heavily in any one particular direction. In case it does, we need to balance it with other KPIs that puts the center of gravity close to the middle. One such KPI is stability.

Developers measure stability with FeatureLeadTime, which is the time taken for one feature to go live in production. A feature comprises multiple commits, and hence a more granular measurement of FeatureLeadTime is CheckIn2GoLive, which is the time taken for a check in to go live in production.

Measure CheckIn2GoLive via pipelines, since this can be approximated by the time a pipeline takes to promote code from test to staging to production. Additionally, CheckIn2GoLive also reflects MTTR (mean time to resolve) defects, since the bug fix travels through the same pipeline from test to staging to production.

Interestingly, when I asked Operations, speed often has a negative connotation since they are incentivized to be risk-aversive. They measure the number of escaped defects to reflect failure, and they define stability by the percentage of defects that are caught by the pipeline as opposed to escaped defects.

Business defines stability by customer satisfaction or the number of repeat customers. While this sounds subjective, you can approximate this metric by the number of defects raised by customers or with customer feedback surveys.

Stability index is a classic example, where Dev, Ops, and Business are opinionated from their own perspective, however, the organization is better off creating a blend, instead of relying on any one. Hence, create an organizational index for stability that is impartial.

Code quality index

Another example where different viewpoints need to be factored in is code quality. Some say the quality of our code is reflected by code coverage measured by unit tests, while others say it is cyclomatic complexity. Standard static analyzers report code duplication, security vulnerabilities, and potential memory leaks. All these are true measures of code quality and hence create an index where all these, and possibly others, play a role.

Business KPIs vs. technical KPIs

Another popular KPI that organizations like to keep an eye on is the value delivered in a sprint. A common bad practice is to record the number of releases, which by themselves don't add value. You could move bits from point A to point B without moving the needle for your business, and that's not good enough. Some organizations measure the number of tests freshly added in that sprint or the total number of tests executed, and those do not reflect business results either, just engineering effort. The value delivered in a sprint has to be relevant to business.

A couple of examples of business KPIs could be the number of customer acquisitions in the last quarter and the number of advertisement clicks in the last month. Pipelines do not directly influence these business metrics. The only reason we try to map business KPIs to technical KPIs is to understand the relationship of technical craftsmanship to business goals.

Business KPIs when mapped to pipelines also help us calculate the RoI (return on investment) of pipelines. Leadership teams use these metrics to understand possible areas of improvement and to plan for budget.

Embark on your journey

Don't waste time debating whether continuous delivery is right for you, or whether continuous integration is enough, or if continuous deployment is your seventh heaven. Now is the time to find out. If you shy away now, and later your business runs out of steam, you will have only yourself to

blame. This isn't so much about where you are in this journey. The fact that you have embarked on this journey will open up avenues of opportunities for your team to continuously improve! It will broaden your horizon by helping you experiment without fear. And if you factor in your employees who won't burn out due to "midnight oil" releases, and people who get to work on making their own worlds better instead of propping up a building that's constantly falling apart, the value becomes obvious. Would you agree?

SHARE THIS ARTICLE

JUNI MUKHERJEE

Juni is a thought citizen in the DevSecOps space and has made deep investments in the field of Continuous Delivery. She has helped organizations build Continuous Delivery Pipelines, and would love to solve the problems that plague our industry today. She has authored a couple of books.

Bitbucket CI/CD tutorials

Continuous Delivery articles

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 →

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.

Read this article →

Sign up for more CI/CD articles and tutorials.

Email

email@example.com

Subscribe

Up Next
Value Stream Mapping →