WHITETRAPER

Introducing Continuous Delivery in the Enterprise
The Situation Today

Not too long ago customers and prospects had to find a computer to visit your site. In stark contrast with just a few years ago, this same customer is now online 24/7 via mobile devices and can be reached and can reach out continuously. The connection between customers and organizations is now fluid, and this phenomenon will only grow in the years to come.

Today, customers expect a reliable and completely up to date service anywhere, anytime. For your organization, this translates into constant pressure to add features and updates to your services almost instantaneously. This makes the quality of your online service more important than ever, as issues with your online system would become visible in an instant. This also means that applications to support the business need to be deployed not only at a much faster rate, but also with more predictability.

For many organizations, a new approach to delivering software isn’t simply about development and deployment practices; it’s critical to doing business.

Challenges

In most organizations, the delivery of software is a time consuming, stressful and costly process. As soon as your application goes live, issues pop up, forcing the IT organization into another costly release cycle. A software release is typically a risky, unreliable procedure that costs businesses both time and expense. Transforming the release process into an automated, reliable, high quality continuous delivery pipeline that “just works” is critical to competing in today’s marketplace. A process that enables the addition of new features to your online service in a matter of hours instead of days. Faster, better, cheaper.

<table>
<thead>
<tr>
<th>Time to market</th>
<th>Higher quality</th>
<th>Lower cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Automated packaging</td>
<td>- Automated test / QA</td>
<td>- Commodity platform</td>
</tr>
<tr>
<td>- Automated testing</td>
<td>- Auto scalable platform</td>
<td>- Less organizational overhead</td>
</tr>
<tr>
<td>- Automated deployment</td>
<td>- Early feedback on code</td>
<td>- No manual intervention</td>
</tr>
<tr>
<td>- Automated provisioning</td>
<td>- 100% reproducible</td>
<td>- Better utilization</td>
</tr>
<tr>
<td>- Instant platform delivery</td>
<td>- No stress</td>
<td>- Predictable platform</td>
</tr>
<tr>
<td>- Self service platform</td>
<td>- Tested &amp; proven deployments</td>
<td>- No more misconfigurations</td>
</tr>
<tr>
<td>- Automated to the max</td>
<td>- Standardized</td>
<td>- Pay platform when needed</td>
</tr>
</tbody>
</table>
What is Continuous Delivery?
Continuous Delivery is a software delivery strategy aimed at moving your ideas into production as fast and efficiently as possible. This allows you to validate your business assumptions with the customer at a very early stage of your delivery process, allowing you to adjust when required. Your ideas will reach the customer faster than is the case with non-automated delivery environments. Because part of the solution is based upon automation of your delivery pipeline, which in itself constitutes a very error-prone process, your delivery will become more reliable.

What does automating your delivery pipeline imply?

1. Code is managed in version control.
2. Compilation and packaging of code is performed centrally in a completely automated fashion.
3. Testing is automated and is treated as an integral part of the delivery process.
4. Applications are automatically deployed, end-to-end, to one or more environments.
5. Server environments are created and torn down with a push of the button.
6. Clear, quality-focused status reports are provided based on facts, not stories.

1. Continuous Integration
- Automated testing
- Automated packaging
- Automated distribution
- Software Library (DSL)
- Clear reports
- Fewer passing errors
- Audibility

2. Application Release Automation
- Automated end to end deploy < 10 min.
- Roll back in < 10 minutes
- Clear reports
- Cheaper deployments
- Fewer deployment errors
- Audibility

3. Provisioning
- Build / teardown environments < 15 min.
- Standardized environments
- Reproducible environment configuration
- Policy controlled creation and scaling
- Fewer errors
- Clear reports
- Less environment errors
- Audibility

A delivery environment that supports Continuous Delivery often consists of the following three mechanisms:

Continuous Integration – overall, this mechanism ensures on a continuous basis (i.e. whenever the code changes), that code is integrated (i.e. all bits compile and run together), tested and is made ready to deploy.

Application Release Automation – forms the bridge between the development and the operations organization; development provides the deployment units plus configurations, where deployment happens based on automated triggers (i.e. tests passed) and is controlled by, often automated, guards or conditions. It provides the operations organization with an easy to use and standardized gateway through which development delivers its applications. It relieves operations from the need of reading through long and tedious installation manuals and configuring/loading individual components (i.e. application servers, databases, http servers, load balancers, queuing mechanisms, services busses) by hand. Deployments become a matter of minutes instead a matter of hours.

Provisioning – used by service management or the operations organization to automatically install building blocks of code to the server and properly configure the systems around it. It further ensures system reliability by enabling to deploy known-good applications to actual known-good environments. Bringing up or tearing down dedicated environments (i.e. quick test or new production environment), now becomes a matter of minutes instead of a matter of days.
What does Continuous Delivery provide your organization?

It lowers your cost

**Traditional:** Deployments require manual tasks, expert scripting and frequent troubleshooting sessions and in that respect represents an investment. As long as the amount of deployments progress, the amount of expenditure grows. Due to rising cost (and deployment duration), the maximum amount of deployments per hour is also bound to limits.

**Continuous Delivery:** The amount of deployments does not have a large effect on overall cost. Once a deployment pipeline is configured, subsequent deployments happen automatically or at the ‘push of the button’. The maximum amount of deployments is not bound to any limit.

![Cost vs. Number of deployments graph]

It shortens your time to market

**Traditional:** Too many changes are delivered in one big release. The time between releases is long and a large effort is required to deploy. A big release with many changes almost inevitably gets delayed because you can’t get a large amount of features to work together in one go. Furthermore, the release process is not efficient: i.e. a release contains 100 changes. When 3 changes fail the test, the 97 changes that actually are correct have to wait for the 3 defects to be fixed before a move to production can be made.

**Continuous Delivery:** Small batches of changes are moved to production in a continuous manner and become instantly visible (if configured that way). Changes can be made immediately available to your customers and a new feature means immediate business value. Feedback from the customer can be gathered almost instantaneously. When a feature is ready for production, it can be moved to production.

![Change vs. Time graph]
It mitigates your risk

**Traditional**: Adding a large amount of changes into an environment introduces risk. Due to the large timeframe between releases, there is a high chance that environments also might have been changed. Every deployment becomes a “big-bang” that touches many moving parts. The chance of hitting some untested combination is high. Every deployment becomes unique, making it impossible to rely on experience from a previous deployment.

**Continuous Delivery**: If it hurts, do it more often. An automated deployment only has to be configured and tested once, and from thereon can be repeated many times in a row (e.g. once per hour). Deployments are proven on a continuous basis and in that respect the risk of a deployment is lower than is the case with manual, infrequent deployments. Simply put: the release process becomes far more reliable.

It (re-)builds trust within your IT organization

**Traditional**: When an activity fails on a regular basis, skepticism grows. When a person is skeptical, this person will try to duck the responsibility or activity as a whole. The same applies to organizations. Operational organizations are responsible for continuity and by nature do not welcome change. Often a conservative approach is taken: implementing large and thorough procedures or even scheduling slack time into the planning often occurs as a measure of mitigation.

**Continuous Delivery**: A deployment runs completely automatically, on a daily basis and is reliable. As the deployment is proven over and over again, confidence will grow. Once the process is in place and runs correctly, there is far less reason for building up resistance between development and operations organizations. Confidence (re-)builds trust.

It helps you to understand your customer
Traditional: An idea is first fully designed and scoped before the project can progress to the task of implementing the idea itself. The result (and success) of an idea is only visible once the full investment has been made. Prior to starting a project, full budget is required and one must be fully certain that ideas are worth the investment. While the project is ongoing, the customer has no way of verifying progress or features.

Continuous Delivery: Envision an idea, design and realize it bit-by-bit. As the project continues, investments are capitalized as features become available. The delivery process provides the possibility to adjust scope & requirements and support the decision to stop investing in features that actually appear not to be used by the end user. Because customer behavior can be measured as the product evolves, the product is better aligned with demand than is the case in traditional situations.

It raises the overall quality of your application

Traditional: Code is compiled and packaged manually whenever this is deemed necessary. Manual tests are performed once code is in its final stages, making test results visible only at the end of a project. When a test fails, it is hard to find the solution as there is no real 1:1 correlation between what was ‘changed’ and what needs to be fixed, which costs a lot of valuable time. The project needs to go live, so at a certain moment the code is promoted to production, even though not all issues have been solved.

Continuous Delivery: The process of assembling, compiling and testing is completely automated, forms an integral part of the delivery process and is quick and easy. Possible coding-issues become immediately visible and can be fixed accordingly when the ink is still wet. When code is checked in as ready, this code is actually really ready.
The Maturity Model

A chart measuring Continuous Delivery maturity is provided below. You can use it yourself to see where you stand and what topics require attention.

### DevOps

<table>
<thead>
<tr>
<th>Level 5: Competent</th>
<th>Level 4: Advanced</th>
<th>Level 3: Average</th>
<th>Level 2: Beginner</th>
<th>Level 1: Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations and development are both part of the multidisciplinary delivery team and share responsibilities.</td>
<td>An array of operations works along in projects, an array of stakeholders work along in operations.</td>
<td>Development and operations work together when they are necessary.</td>
<td>Code is accompanied with release notes with which operations should install and manage the application.</td>
<td>Operations are stopped at the end of the project.</td>
</tr>
<tr>
<td>100% fully automated tests all the way to production.</td>
<td>Dashboard provides insight from different perspectives and shows history and progression; through a build server to all.</td>
<td>Automated static code and security analyses after code check-in.</td>
<td>Automated tests are initiated as soon as code is checked in. Tests are focused on unit component testing only.</td>
<td>Reports generated on request by system administrator. Reports are test based.</td>
</tr>
<tr>
<td>Self-service portal or request for environment.</td>
<td>Automated dynamic quality tests like security scans, functional and performance tests guarantee quality of code.</td>
<td>Environments are identical. Several tools used to provision and configure an environment.</td>
<td>Scrum, continuous integration testing only.</td>
<td>All tests require manual activity. Some tests are automated but have to be initiated by hand.</td>
</tr>
<tr>
<td>Continuous end-to-end deployments.</td>
<td>Environments can be created and torn down by a push of the button. Operating systems is virtualized.</td>
<td>Environments are identical. Roll out of applications performed by a push of the button. Auto-deployment to B, T, A, and B.</td>
<td>Manual installation and configuration of software for middleware, databases, application servers, etc.</td>
<td>Manual installation and configuration of software for middleware, databases, application servers, etc.</td>
</tr>
<tr>
<td>End-to-end automated builds.</td>
<td>Test-guided deployments of end-to-end applications. Deployments occur over multiple environments.</td>
<td>Build on commit. Artifacts are made available for reuse by other teams.</td>
<td>Deployment through execution of separate deployment, and deployment. Manual configurations and install tasks.</td>
<td>Build are performed on local workstation by one of one or more repeatable build jobs.</td>
</tr>
</tbody>
</table>

### DevOps

In what manner do operations and development work with one another and what is the structure of the complete delivery team on an overall level? Are applications moved from one organization to another, or do disciplines work well together, define a real team and feel end-to-end responsibility?

### Reporting

In what way manner can we gain actual insight into the state of the projects? Are status reports created manually and do they only report on the items that people are actually aware of, or is there a dashboard providing actual status (i.e. provides a holistic view) on all aspects one needs to be aware of?

### Testing

What is the attitude towards testing? Are tests performed when a project almost finishes or are tests automatically performed at the start of the delivery process, guaranteeing quality as early as possible? Is testing treated as an integral aspect of a development project and is most of it automated?

### Provisioning

In what manner are the systems in which your application runs configured? Is this done manually, introducing the risk of configuration drift, misconfigurations and time-consuming mistakes, or is this configured as a repeatable process with which you install version-controlled, standardized platforms in a matter of minutes?

### Deploying

When deploying and configuring end-to-end applications across multiple servers, are these activities performed manually, or is this implemented as an automated, repeatable and predictable process based on Application Release Automation (ARA)?

### Building

Are builds performed locally on the developer’s workstation or at regular intervals, in a central place, tested, while making the resulting components accessible for reuse by other teams?
The Principles

When considering a move to Continuous Delivery, some basic software development process principles apply:

\[ \text{higher quality} + \text{automation}^{(\text{test} + \text{provisioning} + \text{deployment})} = \text{cost reduction} + \text{acceleration} \]

Higher quality

Applying quality mechanisms from the very start of your software delivery process will alleviate the potential for costly issues to be revealed only at the end of the project, when the pressure is high and software needs to go live. Raising the bar on quality requires attention to automated testing features, standardization of your environment and minimizing the amount of human interactions that are required for creating and tearing down (test) environments.

Automation test

Over the last couple of years, major improvements have been made in test automation. Every type of test – functional, performance, stress - can be automated, allowing test activities to be treated as an integral part of your software delivery process. Automated tests will become faster and more consistent than would be the case if these tests were executed manually and can be executed at lower cost.

Automation provisioning

Constructing and tearing down the Development, Testing, Acceptance and Production (D, T, A, P) environments is a highly error-prone part of the software delivery process. A totally manual install and configuration of software, middleware, databases and servers requires such a large number of steps that it is nearly impossible to get right in one go, and especially to get right consistently across environments. Automating the provisioning of your system will allow you to construct and tear down environments with the push of a button, making this activity faster, less complex, less costly and far less error prone.

Automation deployment

Another error-prone aspect of the software delivery process is the actual deployment of applications. Using a tool for Application Release Automation (ARA) will enable you to deploy and redeploy applications over your D, T, A and P environment in a matter of minutes, over and over again. Long and tedious installation manuals are replaced by standard instruction sets (templates), moving you to a state of errorless deployments can be performed on a repetitive basis. This, of course, will save you a lot of unnecessary time and costs.

Cost reduction

As long as your software installation and application deployments process are not automated, the delivery and installation of each new D, T, A and P environment will cost you a large portion of your budget. By standardizing these processes and applying them automatically across all your environments, you invest only once and benefit from this investment over and over again. Add to that the possibility of tearing down environments at any time when not in use, and you will improve the operational effectiveness of your organization as you go.

Apart from making your operational environment more reliable, automated testing will also lower your costs. Expensive errors or bugs will not pop up close to the end of your project, or even after go-live, but at the beginning, when repair is comparatively cheap and causes still easy to find. This allows your teams to no longer have to focus on fixing costly issues but instead on what feels productive: creating software.

Acceleration

Through automation and a far less error-prone software delivery process, you will be able to deliver new functionality to production on a continuous basis. The amount of time-consuming manual activities are reduced to a minimum and, by taking an integrated and automated test approach, fewer errors will occur all the way at the end, resulting in faster feature delivery.

All in all, provisioning your environments, testing your software and then deploying your applications will become an activity of hours or minutes, not days or even weeks.
Steps to Getting Started

Assessment
Take a quick assessment to gain a clear picture of the current status of your delivery processes. Assess the status of topics identified by the Continuous Delivery maturity model. More detail on this model is provided in the next chapter. The assessment is essential to determine order, approach and expected timelines for implementing Continuous Delivery.

Engage the development team
When a plan has been determined, engage with your development team. Discuss the approach and engage them in implementing a new, fresh and automated delivery process. Experience shows that development teams love to upgrade their way of working, so this is the nice part. Encourage the development team to participate in determining what is needed in this new environment and especially where they think the current process can be optimized. Define a strategy and way of working and utilize Value Stream Analysis (VSA) to see where the greatest benefits can be found.

Implement and align tools & processes
This is the phase in which configuration management systems are consolidated, build and automated test tools are aligned, the Definitive Software Library (DSL) is introduced and Application Release Automation (ARA) mechanisms are added. Concepts and guidelines are explained and written out, ensuring proper use for subsequent development teams that will start utilizing this new approach. In this phase, processes and guidelines are optimized.

Formalize the environment
Once the new tools and principles are in place, it is time to introduce other development-related initiatives to the new Continuous Delivery environment too. Make sure short and descriptive explanations of the new process are available and guide people in applying them correctly. This phase might take some time, as people have to grow into the new way of working. Slowly step away from release-like thinking for the projects that utilize this new environment. Move away from big-bang releases towards gradually bringing new functionality into production.

Achieve Continuous Delivery
The final phase is defined to remove all roadblocks standing in the way of actually applying Continuous Delivery as the standard in your organization. Actual Continuous Delivery is achieved when all involved are capable of delivering new functionality in an agile manner. This final phase is aimed at further streamlining all parties involved: from business all the way to operations, everybody has to share a product-like mindset and feel responsible for the end product. For example: Break the build = fix the build, meaning if one leaves the code broken, one fixes the code before going home as code that is in the pipeline has to be potentially shippable at all times.
Continuous Delivery is about releasing high quality software fast by connecting Continuous Integration with build, test and deployment automation. Deployit, the market-leading Application Release Automation solution from XebiaLabs, offers a best practices approach to Continuous Delivery. Our specialized deployment automation software integrates with best-of-breed build and provisioning tools, enabling you to achieve Continuous Delivery and all of its benefits including cycle time reduction, real-time software release validation and more predictable delivery costs. Deployit gives you the platform to integrate all of the components you need to get started with continuous delivery today.

Learn more: www.xebialabs.com/continuous-delivery

About XebiaLabs

XebiaLabs is a global provider of Application Release Automation software. Deployit, the market-leading deployment automation solution, helps customers across all industries drive financial and business value by automating application deployment, improving visibility and ensuring compliance, while saving on operational costs. www.xebialabs.com

At XebiaLabs, we define the Continuous Delivery principle as the standard way of working. If you are interested in how XebiaLabs can help your organization get started with Continuous Delivery, please contact us for a demonstration: info@xebialabs.com