Continuous Delivery
Huge Benefits, but Challenges Too

Lianping Chen, Paddy Power

// This article explains why Paddy Power adopted continuous delivery (CD), describes the resulting CD capability, and reports the huge benefits and challenges involved. This information can help practitioners plan their adoption of CD and help researchers form their research agendas. //

CONTINUOUS DELIVERY (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. CD is attracting increasing attention and recognition.

CD advocates claim that it lets organizations rapidly, efficiently, and reliably bring service improvements to market and eventually stay a step ahead of the competition.[1] This sounds great. However, implementing CD can be challenging—especially in the context of a large enterprise's existing development-and-release environment.

Here, I explain how we adopted CD at Paddy Power, a large bookmaking company. I describe the resulting CD capability and report the huge benefits and challenges involved. These experiences can provide fellow practitioners with insights for their adoption of CD, and the identified challenges can provide researchers with valuable input for developing their research agendas.

The Context

Paddy Power is a rapidly growing company, with a turnover of approximately €6 billion and 4,000 employees. It offers its services in regulated markets, through betting shops, phones, and the Internet.

The company relies heavily on an increasingly large number of custom software applications. These applications include websites, mobile apps, trading and pricing systems, live-betting-data distribution systems, and software used in the betting shops. We develop these applications using a range of technology stacks, including Java, Ruby, PHP, and .NET. To run these applications, the company has an IT infrastructure consisting of thousands of servers in different locations.

These applications are developed and maintained by the Technology Department, which employs about 400 people. A software development team's size depends on the application's size and complexity. Our teams range from two to 26 people; most teams have four to eight people.

The release cycle for each application also varies. Previously, each application typically had fewer than six releases a year. For each release cycle, we gathered the requirements at the cycle's beginning. Engineers worked on development for months. Extensive testing and bug fixing occurred toward the cycle's end. Then, the developers handed the software over to operations engineers for deploying to production. The deployment involved many manual activities.

This release model artificially delayed features completed early in the release cycle. The value these features could generate was lost, and early feedback on them wasn't available.

Many releases were a "scary" experience because the release process wasn't often practiced and there were many error-prone manual activities. Priority 1 incidents caused by manual-configuration mistakes weren't uncommon. In addition, the release activities weren't efficient. Just setting up the testing environment could take up to three weeks.

To improve the situation, Paddy Power started an initiative to implement CD. The company established a dedicated team of eight people, which has been working on this for more than two years.
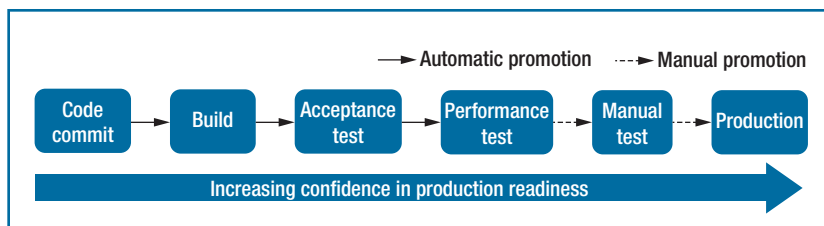
The CD Pipeline

Because we needed to support many diverse applications, we built a platform that lets us create a CD pipeline for each application. Our team operates and maintains this platform. When an application development team needs a new CD pipeline for its application, we create one.

An application's pipeline might differ slightly from another application's pipeline, in terms of the number and type of stages, to best suit that application. Figure 1 shows an example pipeline.

Code Commit

The code commit stage provides immediate initial feedback to developers on the code they check in. When a developer checks in code to the software-configuration-management repository, this stage triggers automatically. It compiles the source code and executes unit tests.

When this stage encounters an error, the pipeline stops and notifies the developers. Developers fix the code, get the changes peer reviewed, and check in the code. This triggers the code commit stage again and starts a

FIGURE 1. An example continuous delivery (CD) pipeline. Promotion—advancing the pipeline's execution from one stage to another—can be automatic or manual. Our confidence in a build's production readiness increases as the build passes through each pipeline stage.

new execution of the pipeline. If everything goes well, the pipeline automatically moves to the next stage.

Build

The build stage executes the unit tests again to generate a code coverage report, runs integration tests and various static code analyses, and builds the artifacts for release. It uploads the artifacts to the repository that manages them for deployment or distribution. All later pipeline stages will run with this set of artifacts.

Before we moved to CD, the binaries released to production might differ from the tested binaries. This was because we built the software multiple times, each for a different stage. Each time we built the software, we ran the risk of introducing differences. We've seen the bugs these differences cause. Fixing them was frustrating because the software worked for the developers and testers but didn't work in production. The CD pipeline eliminated these bugs.

If anything goes wrong, the pipeline stops and notifies the developers; otherwise, it automatically moves to the next stage.

Acceptance Test

The acceptance test stage mainly ensures that the software meets all specified user requirements. The pipeline creates the acceptance test environment, a production-like environment with the software deployed to it. This involves provisioning the servers, configuring them (for example, for network and security), deploying the software to them, and configuring the software. The pipeline then runs the acceptance test suite in this environment.

Previously, setting up this environment was a manual activity. For one of the very complex applications, setup took two weeks of a developer's time. Even for a smaller application, it took up to half a day.

For a new project, setup took even longer. The developers needed to request new machines from the infrastructure team, request that the Unix or Windows engineering team configure the machines, request network engineers to open connections between the machines, and so on. This could take a month.

With the CD pipeline, the developers don't need to perform these activities. The pipeline automatically sets up the environment in a few minutes.

Similar to the other stages, if any errors arise, the pipeline stops and notifies the developers; otherwise, it moves to the next stage.
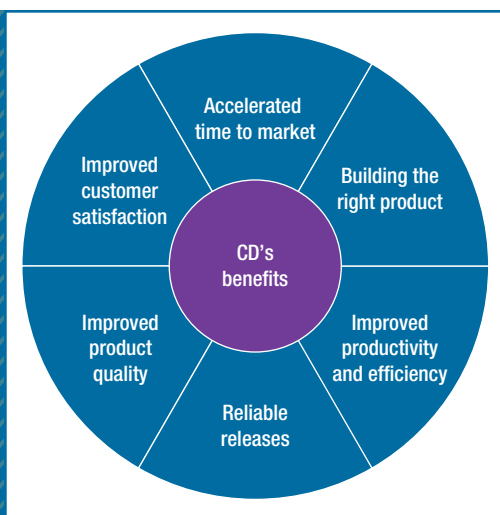
FIGURE 2. CD's benefits. Motivated by these benefits, the company is increasing its investment in CD.

Performance Test

The performance test stage gauges how the code change will affect the software's performance. The pipeline sets up the performance test environment, runs a suite of performance tests in this environment, and feeds the results into the tool that centrally manages software quality.

Previously, owing to the considerable effort of setting up a performance test environment, performance testing didn't occur during development. We performed it only before the big release.

With the pipeline, performance testing occurs for each code commit that passed the previous stages. This lets developers get immediate feedback if the code change has degraded the software's performance. Diagnosing and fixing problems at this time is much cheaper than doing so before a big release.

Manual Test

Although our automated testing is quite comprehensive, manual testing is sometimes necessary (for example, when the testers perform exploratory testing[2] and the business users perform user acceptance testing).

Previously, the testers had to set up a manual-testing environment, which they said was a headache. There were many manual, error-prone steps.

With CD, they no longer need to worry about this. The pipeline automatically sets up the test environment and notifies the testers with email containing the information required to access the deployed application.

When the tests complete satisfactorily, the set of artifacts is promoted from "potential release candidate" to "release candidate." At this point, the software has passed all the quality checks and is ready to deploy to production.

Production

Deployment into production takes just the click of a button.

Previously, such deployment sometimes failed because of errors in the deployment process and scripts. CD has no manual deployment steps, and the deployment process and scripts have been tested many times in previous stages.

Benefits

So far, we've moved 20 applications to CD. They're developed by one of the largest software development groups. Their main users are business people in the company. All the development teams have adopted an agile approach called Kanban[3] while moving their applications to CD.

On these applications, CD has produced the following six main benefits (see Figure 2).

Accelerated Time to Market

The release frequency has increased dramatically. Previously, an application released once every one to six months. Now, an application releases once a week on average. Some applications have released multiple times a day when necessary.

The cycle time from a user story's conception to production has decreased from several months to two to five days.

CD lets us deliver the business value inherent in new software releases to our customers more quickly. This capability helps the company stay a step ahead of the competition, in today's competitive economic environment.

Building the Right Product

Frequent releases let the application development teams obtain user feedback more quickly. This lets them work on only the useful features. If they find that a feature isn't useful, they spend no further effort on it. This helps them build the right product.

Previously, teams might have worked on features that weren't useful but didn't discover that until after the next big release. By that time, they had already spent months of effort on those features.

Improved Productivity and Efficiency

Productivity and efficiency have also improved significantly. For example, developers used to spend 20 percent of their time setting up and fixing their test environments. Now, the CD pipeline automatically sets up the environments. Similarly, testers used to spend considerable effort setting up their test environments. Now, they don't need to do this, either.

Operations engineers used to take a few days' effort to release an application to production. Now, they only need to click a button; the pipeline automatically releases the application to production.

Furthermore, developers and operations engineers used to spend much effort on troubleshooting and fixing issues caused by the old release practice. The CD pipeline eliminated these issues. The effort that otherwise would have been spent fixing these issues can be used for more valuable activities.

Reliable Releases

The risks associated with a release have significantly decreased, and the release process has become more reliable.

As we mentioned before, with CD, the deployment process and scripts are tested repeatedly before deployment to production. So, most errors in the deployment process and scripts have already been discovered.

With more frequent releases, the number of code changes in each release decreases. This makes finding and fixing any problems that do occur easier, reducing the time in which they have an impact.

Moreover, the CD pipeline can automatically roll back a release if it fails. This further reduces the risk of a release failure.

The engineers commented that they don't feel the same level of stress on the release day that they did previously. That day becomes just another normal day.

Improved Product Quality

Product quality has improved significantly. The number of open bugs for the applications has decreased by more than 90 percent.

With CD, immediately after a code commit, the whole code base undergoes a series of tests. If the tests find a problem, the developers fix it before moving to another task. This eliminates many bugs that otherwise would have been open in the bug-tracking system with the old release practice.

Previously, the bug-tracking system recorded many open bugs. Approximately 30 percent of the workforce was fixing bugs. Now, usually nobody is working on customer-found bugs. Bugs are so rare that the teams no longer need a bug-tracking system.

On the rare occasion that a bug is discovered in production, it's added to the team's Kanban board and gets fixed and released in a few days. Before, customers had to wait for the next big release to get the bug fix. The time frame was usually months.

In addition, priority 1 incidents in production have decreased significantly. This is because, apart from the reasons we just listed, the CD pipeline has eliminated the errors that might result from manual configurations and error-prone practices.

Improved Customer Satisfaction

Before we moved to CD, distrust and tension existed between the users' department and the software development teams, owing to quality and release issues. The managers commented that the relationship has improved enormously. Trust has been established.

Challenges

Motivated by these huge benefits, the company is increasing its investment in CD. Expanding the adoption of CD across the company and improving the CD platform are receiving top priority. Nevertheless, implementing CD involves considerable challenges.

Organizational Challenges

The biggest challenge has been organizational. Release activities involve many divisions of the company. Each has its own interests,

> Frequent releases let the application development teams obtain user feedback more quickly.

ways of working, and perceived territories of control. Tension existed between divisions due to competing goals. For example, we needed root access to the servers, and another team controlled this permission. Arriving at a solution involved much consultation and negotiation over six months.

To address the organizational challenges, the leadership team restructured the organization to break down barriers among teams and promote a collaborative culture. The situation has improved since.

Although literature on organizational change exists,[4] little, if any, research focuses on introducing CD to an organization. Further research on this topic—for example, understanding the challenges in more depth and developing strategies and practices to tackle them more effectively—will significantly help an organization's smooth adoption of CD.

ABOUT THE AUTHOR

LIANPING CHEN is a senior software engineer at Paddy Power and a part-time doctoral researcher at Lero—The Irish Software Engineering Research Centre at the University of Limerick. His research interests include software requirements and architecture, continuous delivery, DevOps, and software product lines. Chen received an MS in software engineering from Northwestern Polytechnical University. Contact him at lchen@paddypower.com.

and technologies as building blocks. Avoiding vendor lock-in is challenging. Work on developing widely accepted standards, defining open APIs, and building an active plug-in ecosystem will help alleviate the challenge.

Dealing with applications that aren't amenable to CD (for example, large, monolithic applications) is also challenging. A huge number of such applications exist in the industry. Research is needed on understanding their characteristics and identifying and developing the best strategies or practices to tackle them.

We'd like to solve the challenges we just described through close collaboration with researchers and companies, so that more organizations can easily avail themselves of CD's benefits.

For a brief look at other research related to CD, see the sidebar.

Process Challenges

Many traditional processes hinder CD. For example, a feature that's ready for release normally must go through a change advisory board.[5] This can delay the release for up to four days. If a feature takes only a few days from conception to being ready for release, this four-day period accounts for too much of the feature's total cycle time.

Research is needed to identify these processes (covering areas of business, software development, operations, and so on) and develop and verify alternatives that suit CD.

Technical Challenges

A robust, out-of-the-box, comprehensive, and yet highly customizable solution for CD doesn't exist yet. So, we developed our own solution, which was costly. Tools that fill this gap will save companies considerable resources.

When we're building the CD platform, we use many different tools

References

1. J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley Professional, 2010.
2. C. Kaner, J. Falk, and H.Q. Nguyen, *Testing Computer Software*, 2nd ed., John Wiley & Sons, 1999.
3. D.J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*, Blue Hole Press, 2010.
4. R. Todnem By, "Organisational Change Management: A Critical Review," *J. Change Management*, vol. 5, no. 4, 2005, pp. 369–380.
5. A. Rob, *Effective IT Service Management: To ITIL and Beyond!*, Springer, 2007.