Itkonen, Juha; Udd, Raoul; Lassenius, Casper; Lehtonen, Timo

Perceived Benefits of Adopting Continuous Delivery Practices

Published in:

DOI:
10.1145/2961111.2962627

Published: 08/09/2016

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY-SA

Please cite the original version:
Perceived Benefits of Adopting Continuous Delivery Practices

Juha Itkonen  
Aalto University  
Department of Computer Science  
P.O. Box 15400, 00076 Aalto  
juha.itkonen@aalto.fi

Raoul Udd  
Aalto University  
Department of Computer Science  
P.O. Box 15400, 00076 Aalto  
raoul.udd@aalto.fi

Casper Lassenius  
Aalto University  
Department of Computer Science  
P.O. Box 15400, 00076 Aalto  
casper.lassenius@aalto.fi

Timo Lehtonen  
Solita Oy  
Åkerlundinkatu 11, 33100 Tampere, Finland  
timo.lehtonen@solita.fi

ABSTRACT

Context: In continuous delivery, the aim is that every feature passes through the integration and deployment pipeline, resulting in an immediately deployable product. This practice has been proposed to accelerate value delivery, improve software quality and increase developer productivity.

Goal: We investigate the adoption of continuous delivery practices and evaluate the related benefits in a single customer-supplier organization. We focus on the perceived benefits of supplier and customer organizations during a five-year transition period.

Method: We performed an exploratory case study. We used semi-structured interviews and thematic analysis.

Results: Increased communication and collaboration between developers and customer was perceived as one of the core benefits. Other reported benefits were increased productivity, improved product quality, improved developer morale as well as infrastructural independence and organizational agnosticism.

Conclusions: The results indicate that the adoption of continuous software engineering practices bring various benefits for both customers and developers, beyond mere increased pace of production deployments.

CCS Concepts

- Software and its engineering → Software creation and management; Agile software development; Requirements analysis; Software verification and validation;

Keywords

continuous delivery; case study

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s).

ESEM '16, September 08-09, 2016, Ciudad Real, Spain
ACM 978-1-4503-4427-2/16/09.
http://dx.doi.org/10.1145/2961111.2962627

This work is licensed under a Creative Commons Attribution-ShareAlike International 4.0 License.

1. INTRODUCTION

Continuous delivery (CD) of features to users and value for customers is an increasingly emphasized theme in modern software development. Continuous software engineering [3] and DevOps [9, 12] extend beyond agile software development by emphasizing the immediate delivery and deployment of features instead of incremental or scheduled releases.

The continuous delivery and deployment practices have been proposed to enable accelerated value delivery, reduce the risk of failure, increase productivity, improve visibility, feedback and quality, see, e.g., [5, 4, 7, 2, 6, 10]. There are few empirical studies investigating the effects of continuous delivery practices and, furthermore, the reports commonly take only the development viewpoint. A recent mapping study found that the strength of the evidence on the benefits of CD is limited as many claims are based on industry reports and non-empirical studies [10].

Continuous delivery is a practice in which software is constantly kept releasable by automated continuous integration (CI), testing, and deployment processes [5]. CD extends CI by continuously testing that the software is of production quality through an automated release process. The difference to continuous deployment is that production deployments are done according to separate decisions.

In this study, we investigate the perceived effects of a five-year transformation towards CD both from the viewpoint of the supplier and the customer. Our analysis is based on a single case study in a customer-supplier organization developing software systems for a public sector customer.

2. CASE STUDY DESIGN

We apply an exploratory single case study method [13] to investigate how the adoption of CD was carried out and what the perceived effects were in a software development organization. We analyse the case over a five-year period using retrospective analysis based mainly on interviews with the involved personnel.

2.1 Research Questions

The overall research goal was to explore how the effects of CD adoption were perceived by both the supplier (de-
development) and the customer organizations. We stated the following open-ended research questions to investigate the perceived benefits in the context of this single systems development organization.

RQ1: How does the development organization perceive the benefits of CD adoption?

RQ2: How does the customer organization perceive the benefits of CD adoption?

2.2 Data Collection

In order to gain a deep understanding of the case and its context, we chose interviews as the source of data. We conducted interviews both with the supplier organization and their customer. The interviews were semi-structured, following a general interview guide approach [8], and consisting of open-ended questions regarding several themes. The interviews were conducted in a conversational style. The interview guide ensured that each interview covered the same themes and questions, thus making the interviewees’ responses more comparable. Some conversational elements allowed us to explore unexpected but relevant answers and concepts further. The interviews lasted from one to two hours each. In each interview, one interviewee and two researchers were present. In addition to written notes, the interviews were audio-recorded and transcribed.

We interviewed two employees at the customer organization: one IT architect, who was heavily involved in the collaboration with the supplier, and the lead user of one of the systems. The goal of these interviews was to understand how the customer perceived the changes to processes and methods, and to identify how, if at all, the adoption of CD has had beneficial effects on their work.

The perspective of the developers at the supplier organization was obtained through interviews with two developers and the supplier’s lead developer. These interviews covered the initial state when the developer was assigned to the project, the changes after that, and the current situation. Furthermore, we met several times with the supplier’s lead developers, gaining valuable insight into the case.

2.3 Data Analysis

We applied thematic analysis with an open coding approach on the transcripts of interviews and meetings. For the purpose of extracting the important events, benefits and explanations, the thematic analysis was chosen as the analysis approach.

Thematic analysis is a method of identifying themes, or patterns, within a body of information [1]. Thematic analysis acknowledges that the researcher plays a central role in identifying the themes that are discovered. There exists a conscious process, by which the researcher makes decisions on whether or not a section of the data constitutes a theme, that can be audited and reasoned about. There is no specific set of rules that define what is or is not a theme [1]. Rather, it is up to the researcher to assess the contextual importance and uniqueness of a piece of information.

Open coding [11] is a method that lends itself well to thematic analysis and allows the utilization of the full richness of the qualitative data. The coding method in this study was semi-open, as a set of categories of special interest already existed; we wanted to look at what happened (e.g. events, changes, incentives) and what the results were (e.g. benefits, challenges). However, the coding was performed in an open manner, i.e., unexpected phenomena and concepts were not discarded. All in all nine themes and over 120 codes were identified. The themes were: methodology, processes, collaboration, testing, deployment pipeline, monitoring, productivity, software quality, work satisfaction, and agnosticism.

2.4 Case Description

The case organization, Solita Oy, is a Finnish provider of digital services. The organization as a whole employs around 450 people. We studied a software development organization with 13 developers responsible for developing and maintaining 16 different applications for a single public sector customer.

The developers organize around small, 2–4 developer projects in order to deliver a large feature or a set of features. These projects typically involve changes to several applications. A single developer may participate in several projects at any time. In addition to projects, the development organization has a separate “lane” to handle small and urgent development tasks outside of projects, such as bug fixes.

The applications each have a designated lead user in the customer’s organization. The operational systems are managed by the IT management of the customer and reside in the customer’s environments.

2.4.1 Initial State

The starting point for the CD adoption was in 2010 when the current supplier took over the responsibility for the whole application suite of the customer. At that time, about 4 de-

<table>
<thead>
<tr>
<th>Year</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>The supplier takes the development of all of the customer’s core applications</td>
</tr>
<tr>
<td>2011</td>
<td>CI-server (Hudson) is taken into use</td>
</tr>
<tr>
<td>2012</td>
<td>Nightly dumps of production databases for development use</td>
</tr>
<tr>
<td>2013</td>
<td>Build tool switch (Ant replaced with Maven)</td>
</tr>
<tr>
<td>2013</td>
<td>Database migrations are automated</td>
</tr>
<tr>
<td>2013</td>
<td>Application server migration initiated (WebLogic replaced with JBoss)</td>
</tr>
<tr>
<td>2013</td>
<td>The first deployment scripts are taken into use</td>
</tr>
<tr>
<td>2014</td>
<td>Separated environment configurations enable environment independent builds</td>
</tr>
<tr>
<td>2014</td>
<td>All applications have deployment scripts</td>
</tr>
<tr>
<td>2014</td>
<td>Server configuration is automated (Ansible &amp; Vagrant)</td>
</tr>
<tr>
<td>2014</td>
<td>Data center is moved</td>
</tr>
<tr>
<td>2015</td>
<td>Successful commits trigger deployment to customer acceptance test server (trial)</td>
</tr>
<tr>
<td>2015</td>
<td>Customer environments are monitored and smoke-tested (Dataloop.io &amp; Smokemonster)</td>
</tr>
</tbody>
</table>
developers were involved in the development. The development could be described as an undefined iterative waterfall model. Heavy specifications were written for each new feature by the customer, delivered to the developers, developed in relative isolation and finally tested. This way of working caused the most salient challenge of the organization at that time: it was difficult for the customer to know and specify their actual needs in detail. Especially the developers noticed that even if the end result corresponded with the specifications, the implementation still required changes. Reworking features lead to considerable amounts of wasted work, method sometimes led to unnecessarily expensive solutions to the customer’s requirements as the customer specified solutions without proper knowledge of the architecture and other technical aspects.

In the beginning, there was hardly any automation of testing, integration or deployment. For each release, environment dependent artifacts were built and then transported to the customer on a flash drive, where the applications were deployed on site. The suppliers had the role of application providers, and carried no responsibility for the operations or environments.

2.4.2 Adoption of CD Practices

The software development organization worked continuously during the five year time period to improve the working practices as well as the technical infrastructure and tools towards more continuous development practices. Table 1 presents the major technical events regarding the development and deployment environments during the study period.

In addition to the improvements in tools, frameworks and environments, the organization improved the development process and collaboration practices among the supplier and customer organizations. The most central changes to the development process and collaboration practices are summarized in Table 2.

The main improvement areas were collaboration, development method, deployment and testing. Collaboration practices were improved by establishing a more continuous dialogue and collaboration between the customer and the vendor with many practices and tools. The development method evolved first into Scrum practices and three month sprints, and further separating a specific “fast lane”, separate from longer projects, by applying a Kanban approach to implementing and deploying certain tasks more rapidly. The deployment pipeline was developed by starting with CI automation, proceeding with homogenization and automation of the environments and configurations, and lately automating the deployment pipeline throughout to customer’s test and production environments. Test automation and tooling was developed, even though high coverage of test automation for legacy code is practically not achievable.

Table 2: Challenges and adopted practices

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaboration</td>
<td>Collaboration on the customer needs</td>
</tr>
<tr>
<td></td>
<td>Collocated development</td>
</tr>
<tr>
<td></td>
<td>Common ticketing system</td>
</tr>
<tr>
<td></td>
<td>Instant messaging communication</td>
</tr>
<tr>
<td>Development</td>
<td>Adopting Scrum and Kanban practices</td>
</tr>
<tr>
<td>method</td>
<td>Separating larger projects and “fast lane” development</td>
</tr>
<tr>
<td></td>
<td>Decreasing the task and project size — adopting the minimum viable product-mindset</td>
</tr>
<tr>
<td></td>
<td>Prototyping practice</td>
</tr>
<tr>
<td>Deployment</td>
<td>CI pipeline and automated integration</td>
</tr>
<tr>
<td></td>
<td>Homogenization of the environments and automation of deployment</td>
</tr>
<tr>
<td></td>
<td>Production environments and scrambled databases made available to developers</td>
</tr>
<tr>
<td></td>
<td>Environment independent builds</td>
</tr>
<tr>
<td></td>
<td>Deploying features continuously to customer’s test environment</td>
</tr>
<tr>
<td></td>
<td>Monitoring on CI and deployment pipelines and notifications on failures</td>
</tr>
<tr>
<td>Testing</td>
<td>Increasing visibility and collaboration between developers’ testing and customer acceptance testing</td>
</tr>
<tr>
<td></td>
<td>Extending test automation of new and changed features</td>
</tr>
<tr>
<td></td>
<td>Developing testing tools to assist efficient customer testing</td>
</tr>
</tbody>
</table>

3. PERCEIVED BENEFITS OF CD

In this section we present the results of the qualitative analysis of the interview data regarding the benefits of CD practices. A summary of the benefits is shown in Table 3.

3.1 Increased Productivity

Increased productivity is one of the reported benefits of CD [2, 5]. The productivity gains were perceived both in the supplier and customer organizations.

3.1.1 Supplier’s Productivity Gains

The increasing level of automation decreased the developer effort needed for repetitive, menial tasks related to operations. However, establishing the new tools and practices required substantial effort, sometimes more than expected, due to learning unfamiliar tools and techniques. When asked if the changes were worth it in the end, the unanimous response was positive. Not only has the automation of tasks decreased the amount of repetitive work, such as manual deployments and configuration, but the time spent waiting for deployment decisions, feedback, and customer communication has been significantly decreased due to the closer collaboration practices and proper tools. The potential deployment capability, defined by the deployment pipeline and the automated environment configuration, has increased. Developers now have the capability to deploy fixes even during the same day. The only remaining restriction is that deployments have to take place outside of office hours.

Another contributing factor to increased productivity is the more agile way of specifying requirements. The wasted effort due to developing against badly designed or misinterpreted specs is now largely avoided through continuous discussion and prototyping. These practices also decrease the need for emergency fixes and patches immediately after the deployment.
3.1.2 Customer's Productivity Gains

Prior to the improvements, deployments could take from a few hours to the entire evening and night. Customer staff had to take care of preparation, send out notifications on the expected downtime to stakeholders and users, configure redirection of incoming requests and monitor the entire process until a stable deployment was confirmed. The situation has improved by deployment automation and towards the goal of making deployment a non-issue, even though the production deployments still require short maintenance breaks to the service.

Streamlining the specification work and making requirements elicitation more agile and continuous has improved productivity also in the customer organization. No longer are all features specified to the same extent and the prototyping process makes the requirement and specification work faster and reduces the effort. Using prototypes and making iterative changes by prototyping requires less effort than writing up change requests, new specifications and implementing changes to released features.

The coordination of testing has allowed the customer to focus more on interesting cases based on their domain knowledge rather than testing everything. Furthermore, the custom test automation tools developed by the supplier has cut down on acceptance testing effort of the lead users.

3.2 Improved Collaboration

The increased frequency and amount of communication decreased the amount of redundant work, but it was also regarded as a benefit in and of itself. The collaboration improvements were perceived both by the supplier and customer organizations.

3.2.1 Supplier’s Collaboration Improvements

The continuous discussion on development decisions and specifications was perceived to have improved transparency, as developers started getting continuous input on the viability of different solutions.

Collaborative improvements arose as benefits when discussing almost any theme during the interviews. One such theme was the feedback on deployed changes, which now arrives notably faster than five years ago. Developers also have a much lower threshold for contacting the customer regarding smaller issues or questions during development. Another example is the communication related to coordination of testing responsibilities. It has become more straightforward for developers to decide what tests to write and what to manually check after making changes.

3.2.2 Customer’s Collaboration Improvements

The new ways of collaborating and working together were also a central theme in the customer side discussions. The lead users felt relief that the responsibility of creating perfect specifications of changes no longer rests entirely on their shoulders. Furthermore, the continuous communication and discussion regarding the specifications and application domain is appreciated.

The introduced communication tools, Jira and Flowdock contributed to the benefits of collaboration. The use of a shared Jira project has alleviated the issues of heavy bureaucracy that was present when having separate change management systems. Flowdock provides group discussions and archival of discussions, which was not previously possible. Furthermore, easier access to the shared documentation in Confluence enabled means that developers are now more actively collaborating on that platform.

The lead user also pointed out that while the new ways of communicating are superior and rewarding, it can feel a bit exhausting to be available and active to the degree required by close collaboration on a daily basis.

3.3 Improved Quality

The perceptions of product quality improvements varied among customer representatives. Quality improvement was clearly discerned by the more technically oriented architect. The lead user’s perspective was, instead, that no distinguishable change in quality had occurred during the recent years. However, the quality assurance process was more complicated earlier when several other suppliers were involved, leading to many errors in production. Furthermore, prototyping, which in essence intends to ensure a result more in line with the actual user requirements, was cherished by the lead user as well.

From the perspective of the customer’s architect, many changes had contributed to the fact that the quality has increased. Considerably less bugs after production deployment is attributed to the adoption of automated testing, static code analysis, production-like environments in the deployment pipeline, and major refactoring and platform improvement projects. Many of these improvements have been made under the hood and for the infrastructure, in ways that are not so obvious for the user. The focus in quality assurance seems to be on keeping the quality of the code base from degenerating and achieve a acceptable level of maintainability. Larger undertakings regarding the platform, such as the migration from WebLogic to JBoss, were claimed to have extended the lifetime of some applications with several years, increasing the quality and maintainability considerably form the customer’s viewpoint.

3.4 Reduced Risk of Release Failure

Manual steps in any part of the development process introduce risk of failure due to human error. Four factors were identified as having decreased release and deployment risk significantly. First, the environment independent builds have mitigated both the risk of faulty builds due to erroneous manual configuration, and the impact of environment changes on existing releases. Second, the automatic testing of all changes has reduced the risk of uncovering bugs in the applications after production deployments. Third, the fact that the amount of steps needed to access the customer’s environments has been reduced means that there are fewer things that can go wrong in any procedure re-
quiring such access, for example, deploying to production or making changes to the environments. Last, developers now have access to environments quite similar to those in production, which has been argued to reduce the risk of errors that can be discovered only once the software is running in the production environment.

3.5 Organisational Independence

The supplier’s organization perceived the benefits of reducing the development organization’s dependence on individual developers or tacit knowledge possessed by few.

The automation and deployment pipeline has reached a stage where any developer can deploy an application to the test or production environments. No longer is a vast body of contextual knowledge needed in order to build and deploy an application. The level of mission critical tacit knowledge in the developing organization has decreased.

There is also less need for specialized roles amongst developers. Responsibilities are largely temporary and are administered according to the current interest of the developers. This mentality goes along well with the principles of a DevOps organization, which is seen as a benefit when pursuing CD.

Furthermore, the increase in automation, the new communication practices and many other smaller improvements has allowed the developing organization to grow without larger organizational obstacles. Making constant improvements to the ways of working has enabled a scalable team in this case.

3.6 Infrastructural Agnosticism

The separation of environment configurations from the application source, along with virtualization and automation of the configuration and deployment has improved the readiness for infrastructural changes. In fact, large parts of the automation that exists today was developed in preparation for the data center and infrastructure provider migration in 2014.

These improvements mean that the entire project is less dependent on infrastructure providers, which allows the customer to more flexibly select a suitable provider, when needed. It also means that the impact of infrastructural changes on regular development is mitigated, and improvements or changes to the environments can be made with lower risk. Currently, the decision to move to a new server room with different hardware would not pose a significant threat to the progress of feature development.

3.7 Improved Developer Morale

Developers described a better, less stressful working environment, better work morale and job satisfaction. Much of the reduction in stress levels results from the reduced risk of failing deployments and the additional work that those require.

There are probably few things more frustrating to a developer than developing something entirely useless. Two important changes towards CD have impacted morale positively in this regard. First, being part of the specification process and being able to provide input on design decisions through continuous collaboration with the customer is unanimously regarded as a positive development by the developers. This allows them to feel that they have a say in defining their own work, and also helps avoid the disappointment of developing something that ultimately does not meet the needs of the users. Second, the continuous prototyping, enabled by the deployment pipeline and related automation, allows developers to get faster validation of their interpretation of the requirements, and feedback on their work.

Another improvement to development work was the reduction of repetitive, menial tasks. Interviewees reported that many of the ideas that led to changes simply stemmed from individual developers being annoyed with having to perform some task manually, and the improvements were often decided upon collectively among the developers. These kinds of bottom-up changes have resulted in increased developer satisfaction.

3.8 Discussion

Most of the benefits of CD adoption were identified by the developers. Customer stakeholders recognized partly the same benefits as developers, but emphasized more the benefits of closer collaboration for specification processes and mutual trust, increased product quality, and more efficient use of time.

Regarding the acceleration of value delivery, the potential has clearly improved, but the benefit is not obvious to all stakeholders. This aspect is particularly interesting as faster value delivery is proposed as the primary driver for CD in existing literature. Our results describe how there are multiple benefits of adopting CD that may be more relevant in certain contexts. In our case, the customer organization did not feel pressing need for continuous deployment of all features, even though they valued the benefits that building the continuous delivery capability in close collaboration with the supplier brought to them.

Deployments are clearly not as risky as they used to be and require less effort. Productivity improvements were perceived by almost all interviewees, as the customer spends less time on acceptance testing and redundant specification, and the developers spend less time on non-value adding tasks. User feedback can be gained more quickly, but is limited by the schedules of the customer stakeholders. However, from developers to customer, the feedback cycle showed great improvement.

Both customer representatives commented on the positive effects on software quality that the transition had provided, but it was less visible to the user. Arguably, many of the hazards mitigated by the changes, such as the risk of deployment failures, may not have been very visible to the users.

However, the benefit that emerged as the central theme in the interviews was the improved collaboration. This is somewhat surprising, as collaboration is a crucial enabler of CD and is often seen as a prerequisite rather than a benefit as such. In earlier studies, the close collaboration of development with operations and testing has been identified, e.g., [6, 7]. In this case, collaboration was clearly regarded as a benefit of itself and most prominently by the customer, emphasizing the quality of the interaction with development— not only the speed of feature delivery. In fact, in this case the outspoken need for rapid deployments from the users was surprisingly mild. While the developers strive for same day deployments, the user was satisfied with delivery in a few weeks, as long as it was reliable.

Two other distinct benefits were the organizational independence and infrastructural agnosticism. Stakeholders from both sides of the account saw the ability to switch out both
hardware and people, without threatening the performance of the project, as very valuable. These benefits are particularly interesting as they are somewhat unexpected and not normally a reason for pursuing CD. In this case, infrastructural agnosticism was in fact one of the major reasons for improving the environment independence and automation.

These insights of the potential benefits of CD can be used to more accurately estimate the potential benefits of CD adoption endeavours in practice. Many of the identified benefits provide incremental improvements for the supplier and customer already during the adoption process, before the true CD mode of working is achieved.

Our analysis is limited to the context of the customer-supplier organization in question and to the perceptions of the interviewed persons. The organization is still in active adoption process of CD and, thus, has not adopted fully working set of practices. E.g., production monitoring is in very early stages. Our results can be used to give insight into the various benefits of CD, but no conclusions are possible regarding how common or specific these benefits are in industry. Also, the potential impact of full adoption of CD and DevOps practices cannot be seen from this case data.

4. CONCLUSIONS AND FUTURE WORK

We investigated the perceived benefits of adopting continuous delivery practices in a customer-supplier software development organization. Our results indicate various benefits of the adoption of more continuous software engineering practice. A central finding was the emphasized role of customer-developer collaboration in the case organization. The collaborative working practices were perceived as an important benefit and improvement by both the customer and development organization.

The customer's perceptions on the benefits focused on the collaboration, productivity and quality, whereas the commonly proposed benefits of rapid value delivery and reduced risk of release failures were not recognized by the customer representatives. These results indicate that the adoption of continuous software engineering practices bring various benefits for both customers and developers, other than more immediate production deployments. These benefits and the objectives of CD adoptions need to be better understood, since different customer's needs probably differ in different ways of implementing the CD practices.

Our results raise a few interesting themes for future research. Probably the most prominent theme is studying the needs, motivation and benefits of continuous delivery and deployment in different, well described contexts. There are documented benefits of continuous software engineering practices that focus on the advantages of rapid value delivery, but our results illustrate that the benefits for the customer and end users extend beyond the fast deployment pace.

Another identified topic for future research is the measurement of CD and DevOps in general. Based on our initial analysis, simple metrics, such as lead times and release cycle metrics do not give good enough picture of how the development works. In the DevOps practice measuring and monitoring production systems is a central capability that was not applied much in this case. However, it is not clear what those metrics and monitored aspects of production systems should be, are the metrics highly context specific or could we identify some generally beneficial metrics for CD and DevOps type of software development. Finally, it would be interesting to examine how metrics can be used as a tool for communication with the customer and the users. Metrics could potentially be used to change mindsets, argue the needs for changes and to demonstrate results.

5. ACKNOWLEDGMENTS

This research is part of N4S-Program, funded by Digile and TEKES — the Finnish Funding Agency for Innovation. The case companies made this research possible by providing the researchers with the access to the empirical setting and data.

References