Synthesizing Continuous Deployment Practices Used in Software Development

Akond Ashfaque Ur Rahman, Eric Helms, Laurie Williams, and Chris Parnin
Department of Computer Science, North Carolina State University,
Raleigh, North Carolina, USA
{aarahman, edhelms, lawilli3, cjparnin}@ncsu.edu

***Abstract* – Continuous deployment speeds up the process of existing agile methods, such as Scrum, and Extreme Programming (XP) through the automatic deployment of software changes to end-users upon passing of automated tests. Continuous deployment has become an emerging software engineering process amongst numerous software companies, such as Facebook, Github, Netflix, and Rally Software. A systematic analysis of software practices used in continuous deployment can facilitate a better understanding of continuous deployment as a software engineering process. Such analysis can also help software practitioners in having a shared vocabulary of practices and in choosing the software practices that they can use to implement continuous deployment. The goal of this paper is to aid software practitioners in implementing continuous deployment through a systematic analysis of software practices that are used by software companies. We studied the continuous deployment practices of 19 software companies by performing a qualitative analysis of Internet artifacts and by conducting follow-up inquiries. In total, we found 11 software practices that are used by 19 software companies. We also found that in terms of use, eight of the 11 software practices are common across 14 software companies. We observe that continuous deployment necessitates the consistent use of sound software engineering practices such as automated testing, automated deployment, and code review.**

Keywords—agile; continuous deployment; continuous delivery; industry practices; internet artifacts; follow-up inquiries

I. INTRODUCTION

Agile practitioners consider end-user satisfaction through early and continuous delivery of software as their highest priority [3]. Continuous deployment is a software process that focuses on rapid delivery of software changes to end-users. *Continuous deployment* is a software engineering process where incremental software changes are automatically tested, and frequently deployed to production environments. Facebook [10], Github [39], Netflix [30], and Rally Software [31] are some of the many software companies who are using continuous deployment to deploy their product. Gartner has identified the foundational practice of continuous deployment in web-scale information technology (IT) as one of the top ten strategic technology trends for 2015 [14].

Software companies who are using continuous deployment have reported several benefits of this software process, such as improved customer satisfaction, improved software quality, and savings in development effort [27]. Despite being an emerging software process that provides several benefits, software practitioners have identified the 'lack of understanding' of continuous deployment as an adoption challenge [9].

One way to understand continuous deployment is to learn about the practices and techniques used by software companies to implement continuous deployment. Some of the software companies have used Internet artifacts, such as blog posts, slide and audio presentations, to share their experiences and techniques of using continuous deployment. However, searching the Internet to collect all relevant artifacts might require significant time and effort [28]. Software practitioners can benefit from a study that systematically analyzes the software practices used by software companies to implement continuous deployment. Relative to the adoption of new technologies, software practitioners often prefer to learn through the experiences of other software practitioners who belong to the same industry [29]. This group of practitioners can benefit from a study that maps the software practices and the software companies who are using these practices in continuous deployment.

The goal of this paper is to aid software practitioners in implementing continuous deployment through systematic analysis of software practices that are used by software companies. We studied the continuous deployment practices of 19 software companies by performing a qualitative analysis of Internet artifacts and by conducting follow-up inquiries.

Continuous deployment is closely related to DevOps as a concept. Continuous deployment aims at rapid delivery of software changes to end-users via automated build, test and deployment [19]. On the other hand, DevOps has emerged as a methodology that is involved in the entire product lifecycle through marketing, planning, human resources, and sales along with development and operations [42]. In our paper we focus on analyzing the software practices used in industry to implement continuous deployment.

We state the following research question: *How frequently are different software practices used by software companies that perform continuous deployment?* To answer this research question we collected the software practices used by software companies who are using continuous deployment. We define these software practices as *continuous deployment practices*. This paper uses the qualitative analysis of Internet artifacts and follow-up inquiries using e-mails and social networking to study and analyze the continuous deployment practices.

We summarize the contributions of this paper as following:

a. A summary and concise definition of continuous deployment practices
b. A mapping of continuous deployment practices and the software companies who are using these practices
c. The techniques adoptees have used to realize each of the identified continuous deployment practices

The rest of the paper is organized as follows: in Section II, we briefly describe the background of continuous deployment and prior academic work related to our paper. In Section III, we explain the research methodology of this study in detail. In Section IV, we report our findings. We use Section V to describe the continuous deployment practices with appropriate details. We use Section VI to discuss our findings. We present the limitations of our study in Section VII. Finally we conclude and discuss about future research directions of our study in Section VIII.

II. BACKGROUND AND RELATED WORK

In this section, we first provide definitions. Next, we briefly describe prior works that have studied adoption of software engineering practices, use of agile and lean in industry, practice of rapid releases, and use of DevOps as a practice.

A. Definitions

The motivation of continuous deployment is delivering software changes and features to the end-uses rapidly. In 2004, Beck and Andres introduced the concept of daily deployment as a corollary agile practice [4]. The authors defined the practice of 'putting new software into production every night' as *daily deployment*, similar to the concept of continuous deployment. In 2006, Humble et al. [21] introduced several guidelines of implementing continuous deployment as a practice in software development.

Continuous delivery and continuous deployment are two software engineering processes that focus on delivering software changes quickly to end-users. Humble and Farley [19] defined *continuous deployment* as a software process that releases software changes automatically to end-users after they pass the required automated tests. According to Martin Fowler [12], *continuous delivery* is the software engineering process that builds software in such a way that it is releasable at any time; and *continuous deployment* is the software process that actually releases software to production as soon as they are ready, resulting in many deployments to production every day. Humble and Farley and Fowler's definitions of continuous deployment are similar, though Fowler does not stipulate the use of automated tests.

Some software companies do not strictly follow either of these definitions of continuous deployment. Their deployment rate varies from one another, e.g. Facebook deploys its software changes twice per day [10], whereas, Etsy deploys its software changes 30 times per day [5]. To facilitate a discussion that includes software companies who are deploying different times per day, we use our own definition of continuous deployment in this paper. We define *continuous deployment* as a software engineering process where incremental software changes are automatically tested, and frequently deployed to production environments. We define a software company that is using continuous deployment to deliver software changes to end-users as a *continuous deployment adoptee* or *adoptee,* in short.

B. Adoption of New Practices

Prior works have discussed the motivating factors and challenges of adopting a new practice and necessary mitigation techniques to overcome these challenges. Passos et al. [35] focused on motivation of adopting a new software engineering practice and investigated the motivating factors of adopting a new software engineering practice. Claps et al. [7] focused on technical and social challenges of adopting continuous deployment. In a recent work, Leppanen et al. [27] reported the continuous deployment capability of 15 different companies, and found that none of the 15 companies use a completely automated deployment pipeline to deliver software changes. Olsson et al. [33] in their work studied the adoption challenges of a software company as they shifted from the use of continuous integration to the use of continuous deployment. The authors reported three challenges that the company faced as they transitioned into continuous deployment, including network configuration and upgrade issues, issues related to internal verification loop, and lack of clarity inside the company. Our study provides a summary of continuous deployment practices used in software development using Internet artifacts and follow-up inquiries.

C. Agile and Lean Usage

Lagerberg et al. [26] studied the impact of adopting agile practices for two software development projects developed in Ericsson. They found adoption of agile practices and principles resulted knowledge sharing and balanced use of internal software documentation, correlated with increased project visibility and possible increased productivity, and had no correlation with pressure and stress. Azizyan et al. [1] in their work discussed the sets of tools that are used in agile project management by analyzing survey responses. The authors also discussed the most and least desirable aspects of the tools that are used in agile project management. Rodriguez et al. [38] investigated the extent and impact of adopting agile and lean principles in a Finnish software industry. They identified the number of companies in the Finnish software industry who are using agile and lean principles, their motivations for using these principles, adoption challenges, and how the adoption of these principles are affecting the software organizations in terms of productivity and success. In our work, we focus on studying continuous deployment practices by extracting information from Internet artifacts and follow-up inquiries.

D. Practice of Rapid Releases

Kerzazi and Khomh [24] examined the release data of a software organization, and identified several types of factors that facilitate rapid release cycles. They identified three factors: technical factors, which includes code merging and integration; organizational factors, which includes design and management of branches; and interactional factors, which

includes coordination policies amongst teams. Paasivaara et al. [34] described how a globally distributed development team at Ericsson adopted the practice of rapid releasing to provide applications and services that use an everything-as-a-service platform. In our study we take a different stance. We have analyzed the continuous deployment practices of 19 adoptees and identified the similarities in terms of using those practices.

E. Use of DevOps

Velasquez et al. [43] collected and analyzed survey responses to identify the adoption trend, and impact of DevOps as a culture. In their work they also identified a list of DevOps practices such as use of version control systems, use of automated testing, and monitoring system and application health. In our study, we have identified DevOps and continuous deployment as two different concepts and focused on continuous deployment practices used in industry. We have summarized these practices with definitions, provided a mapping between these practices and the software companies who are using them, and reported the techniques to implement these practices.

III. RESEARCH METHODOLOGY

We describe the major steps of our research methodology in this section. As shown in Figure 1, the first step of our research methodology was to identify adoptees. The next step was to search and identify the Internet artifacts required to understand the adoptees' continuous deployment practices. We used Internet artifacts and follow-up inquiries to analyze software practices used by adoptees. We describe how we perform follow-up inquiries later in this section.

A. Identifying Adoptees:

We used the Google search engine to identify adoptees. From the search results, we identified a software company as an adoptee if the following criteria were satisfied:

a. The search result contains the keyword 'continuous deployment' or 'continuous delivery'

b. The search result states any one or both of the following:

I. the software company deploys software changes at least once a day; and/or

II. the software company uses any of the 11 continuous deployment practices as part of their software development process

We used the first criteria because some of the software companies have interchangeably used the terms 'continuous deployment' and 'continuous delivery' [41] to describe their software development process. We used the second criteria because deployment rate and use of continuous deployment practices varies across software companies. We determined if an adoptee used any one of these 11 practices by performing a simple keyword search. The reader can find the 11 continuous deployment practices in the next subsection.

B. Searching Internet Artifacts:

We used the Google search engine to search Internet artifacts that describe the adoptees' continuous deployment practices. As adoptees have used the terms 'continuous deployment' and 'continuous delivery' interchangeably to describe their software development process, we used types of two search strings:

a. "continuous deployment at <adoptee name>"

b. "continuous delivery at <adoptee name>"

Here <adoptee name> represents that name of the adoptee.

C. Identifying Necessary Internet Artifacts

The next step is to determine if the collected artifacts contain necessary information regarding continuous deployment practices. This step is defined as 'Identify Artifacts' in Figure 1. From the search results we examined if the Internet artifact of interest describes one or many practices for that adoptee.

D. Extract Information to Identify Practices:

The first step towards information extraction from Internet Artifacts was identifying the continuous deployment practices of Facebook. We used Feitelson et al.'s paper [10] to identify Facebook's continuous deployment practices because this study is an academic literature that describes deployment and development practices of Facebook as a software company, and how these practices differ to that of traditional software engineering practices. From the study we identified a set of 11 software practices namely, *automated deployment, automated testing, code review*, *dark launching*, *end-user communication*, *feature flag*, *intercommunication*, *monitoring*, *repository use*, *shepherding changes,* and *staging*. We define each of these practices as a *continuous deployment practice*. From our study we found that Feitelson et al.'s paper was the most comprehensive Internet artifact because using this paper we were able to determine whether or not the 19 adoptees, including Facebook, use each of the eleven continuous deployment practices. The final step of information extraction was to identify if an adoptee has used any one of the 11 continuous deployment practices.

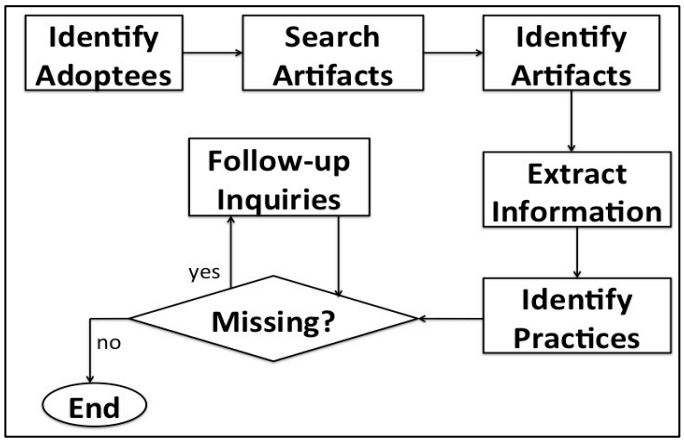

Fig. 1. Major Steps of Research Methodology

E. Follow-up Inquiries

In some cases after extracting information from Internet artifacts, we were not able to identify whether an adoptee performs any of the 11 continuous deployment practices. To complete our investigation we used follow-up inquiries to

obtain more information. To perform follow-up inquiries we first collected the author contacts of each Internet artifact. Next, we queried each author using e-mail and social networking websites. We ended this step when we were able to determine if an adoptee uses a continuous deployment practice. In some cases, our contacts did not respond to our inquiries and thus, we were not able to determine if a certain continuous deployment practice is used by an adoptee. For these cases, we identified that particular continuous deployment practice as an *unknown practice* for the corresponding adoptee.

IV. RESULTS

In this section we present evidence that answers our research question of interest:

How frequently are different software practices used by software companies that perform continuous deployment?

A. Identifying Adoptees

In total we studied the software practices of 19 adoptees. We present the names of the adoptees and the types of product they deploy in Table I. The references used to study the software practices is available online[1]. As shown in Table I, Atlassian is the only adoptee that uses continuous deployment to deploy software changes for desktop software as well as websites. Rest of the 18 adoptees deploys websites using continuous deployment. In Table I, each of the adoptee names is followed by an acronym that we use to map each continuous deployment practice to each of the 19 adoptees. For example, we use *FB* as an acronym to refer to Facebook.

B. Common Continuous Deployment Practices

Using Internet artifacts and follow-up inquiries we were not able to determine if all of the 19 adoptees use the 11 continuous deployment practices. As stated, for an adoptee if we were unable to determine whether it uses a continuous deployment practice, then we mark that practice as *unknown* for that adoptee. In Figure 2, we present three categories: 'Yes' refers to the adoptees who are using a certain a continuous deployment practice, 'No' refers to the adoptees who are not using a certain continuous deployment practice, and 'Unknown' refers to the adoptees for which we were unable to determine if they use a certain continuous deployment practice or not. We list the adoptees that are using and not using a continuous deployment practice as 'Adoptees Using', and 'Adoptees Not Using', respectively in the subsections where we summarize each of the continuous deployment practices. Figure 2 summarizes the use of continuous deployment practices amongst adoptees.

TABLE I: ADOPTEE PROFILE

Type of Product Deployed	Adoptee	Count
Desktop Software	Atlassian (AT)	1
E-Commerce Websites	Coolblue (CB), Etsy (E), Flipkart (FK), URLinkedUp (U), Wealthfront (WF)	5
Social Networking Websites	Facebook (FB), Flickr (FR), Github (G), IMVU (I), Quora (Q), Pinterest (P)	6
Review Websites	Yelp (Y)	1
Other Type of Websites	Atlassian (AT), Google Consumer Surveys (GCS), Kitchensurfing (K), Netflix (N), Outbrain (O), Rally Software (RS), Spreaker (S)	7

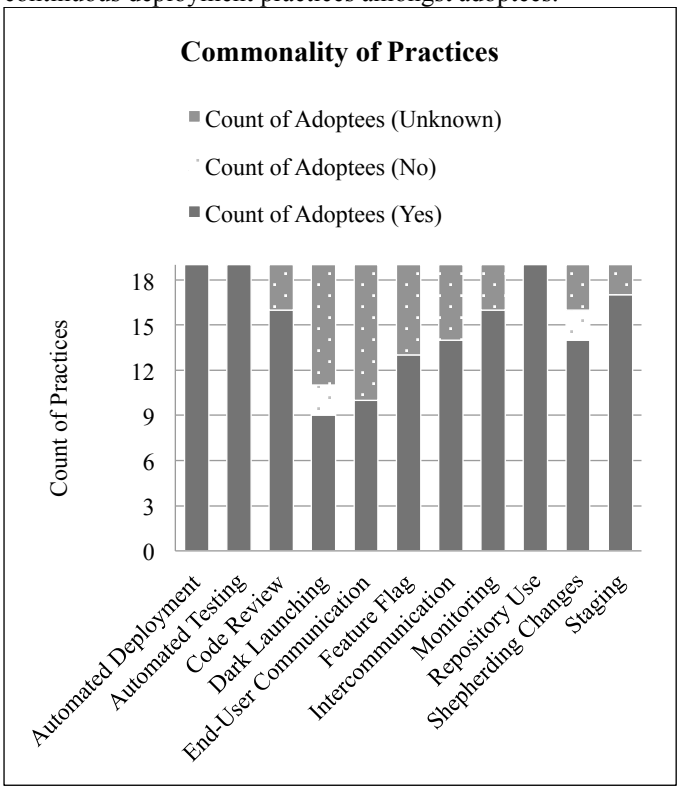

Fig. 2. Commonality of Continuous Deployment Practices

C. Use of Internet Artifacts

In our study we studied 45 Internet artifacts in total that included blog posts, video presentations, academic articles, InfoQ presentations etc. to identify the continuous deployment practices of 19 adoptees, as shown in Figure 3. 55.55% of these Internet artifacts were blog posts.

D. Use of Follow-up Inquiries

Figure 4 presents how we used Internet artifacts and follow-up inquiries in tandem to analyze the continuous deployment practices of the 19 adoptees. According to Figure 4, Internet artifacts were the primary source of information.

[1] http://www.realsearchgroup.org/realsearch/agile-software-development/references/

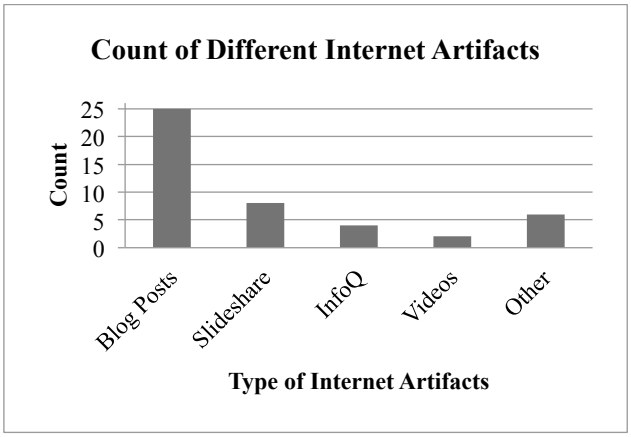

Fig. 3. Types of Internet Artifacts Used in the Study

V. CONITNUOUS DEPLOYMENT PRACTICES

We use this section to describe each of the continuous deployment practices with necessary details.

A. Automated Deployment

Automated deployment refers to the practice of making software available to end-users automatically; this practice is conducted in between software acquisition and software execution without manual effort [19]. The practice of automated deployment facilitates in rapid delivery of software changes to end-users [19].

Adoptees have used a wide range of automated tools to implement the practice of automated deployment including Bit Torrent, automated scripts and Codeship. BitTorrent is a peer-to-peer file sharing mechanism to download, upload and distribute large files across servers [8]. Codeship[2] is another tool that facilitates continuous deployment by automating the code deployment procedure from developer machines to production servers. Table II presents the techniques that adoptees have used to implement automated deployment, along with the adoptees that are using them.

Adoptees Using (19): AT, CB, E, FB, FK, FR, G, GCS, I, K, N, O, P, Q, RS, S, U, WF, Y

B. Automated Testing

Automated testing refers to the practice of automated techniques to perform various testing activities, such as test case management, test monitor and control, test data generation, test case generation, and test case execution [23].

TABLE II: TECHNIQUES USED TO IMPLEMENT AUTOMATED DEPLOYMENT

Technique Used	Adoptee	Count
BitTorrent	FB, P	2
Codeship	K	1
Scripting	FK, GCS, I, RS	4
Other Tools	AT, CB, E, FR, G, N, O, Q, S, U, WF, Y	12

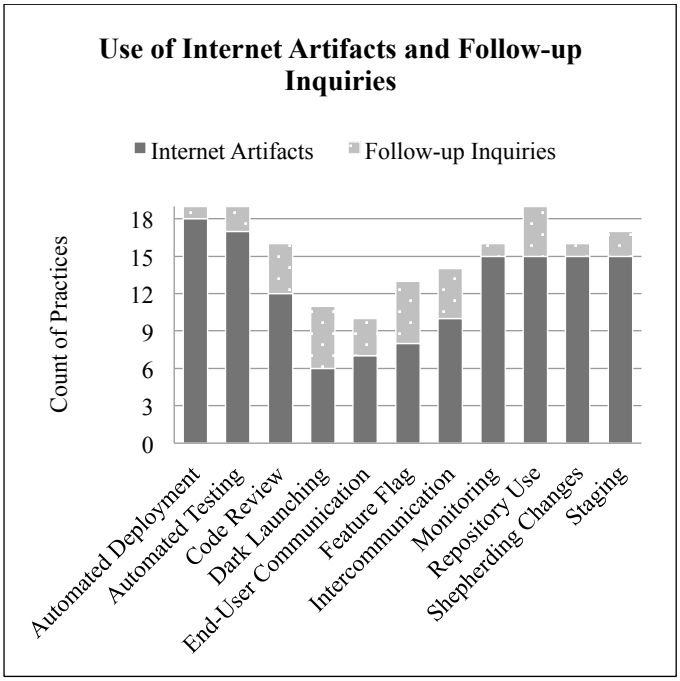

Fig. 4. Use of Internet Artifacts and Follow-up Inquiries in the Study. Internet Artifacts were the Major Source of Information.

Similar to automated deployment, the practice of automated testing facilitates in rapid delivery of software changes to end-users, along with early detection of software defects [19].

In this subsection, we briefly describe different aspects that are related to the practice of automated testing of the adoptees. First, we summarize the types of testing each adoptee is doing as part of their continuous deployment practices. Unit testing refers to testing of individual software components [22]. Integration testing refers to gradual testing amongst different components of the software to test the functionality of the complete software [22]. In alpha beta testing or A/B testing in short, a specific version of the software is deployed to a limited number of end-users to get feedback [22]. In functional testing functional requirements of the software is tested with a specific set of input, ignoring the internals of the software [22]. Acceptance testing is used to test whether or not the software fulfills the contractual requirements of the end-user [22]. Regression testing is performed to check if a software change has adversely affected the functionality of the whole software [22]. Perceptual testing is a testing mechanism that automatically compares two different front-end designs of a user interface[3]. Table III presents the adoptees and different types of testing they are using.

A testing environment is defined as the facilities, hardware, software, firmware, procedure and documentation used in testing of software [23]. To perform automated testing adoptees have used their own testing environments to execute their tests automatically that are alternatively referred as *testing suites* or *testing servers* by adoptees. Implementation

[2] https://codeship.com/

[3] http://www.thoughtworks.com/insights/blog/perceptual-testing

of these testing environments varied amongst adoptees. For example, Pinterest uses a Jenkins-based[4] testing suite, IMVU

TABLE III: DIFFERENT TYPES OF TESTING USED BY ADOPTEES

Type of Testing	Adoptee	Count
Unit Testing	AT, CB, E, FB, G, GCS, I, N, O, P, Q, RS, S, U, WF	15
Integration Testing	AT, E, G, GCS, I, K, N, O, P, Q, RS, W, Y	13
A/B Testing	CB, E, FB, FR, GCS, N, P, RS, S	9
Functional Testing	CB, E, G, GCS, I, N, RS, S	8
Acceptance Testing	AT, N, Y, U, WF	5
Regression Testing	E, CB, FB, WF, Y	5
Perceptual Testing	GCS	1

uses Buildbot[5], and Facebook uses its own testing suites to perform automated testing [10].

The team responsible for performing testing and maintaining quality of software is defined as a testing team [37]. Atlassian and Netflix use separate testing teams as well as automated testing suites to implement continuous deployment. The other 17 adoptees do not have a separate testing team to perform continuous deployment.

Adoptees Using (19): AT, CB, E, FB, FK, FR, G, GCS, I, K, N, O, P, Q, RS, S, U, WF, Y

C. Code Review

Code review is the practice that requires developers to present software changes for comment and approval [22]. Benefits of performing code review on software changes include defect detection, sharing of knowledge amongst software team members, and discovering alternative solutions [2].

To perform code review, developers use manual inspection performed by other team members [22], as well as automated software tools [2]. Adoptees use different techniques to perform code review including automated software tools such as, Gerrit [15], Github tools[6,7], and Phabricator[8]. Static analysis is the technique of evaluating a software component based on its structure, organization, or content [22]. Coding convention is a programming principle where developers have to abide by a set of rules that are specific to a programming language [37]. Table IV summarizes the techniques adoptees have used to perform code review.

Adoptees Using (16): AT, CB, E, FB, G, GCS, I, K, N, P, Q, RS, S, U, WF, Y

TABLE IV: DIFFERENT TECHNIQUES USED TO IMPLEMENT CODE REVIEW

Technique Used	Adoptee	Count
Coding Convention	CB	1
Github Tools	AT, E, G, I, P, RS, Y	7
Manual Inspection	AT, FB, G, Q, U, Y	6
Phabricator	FB, Q	2
Static Analysis	CB	1
Other	CB, GCS, KS, N, S, U, WF	7

D. Dark Launching

Dark launching is the practice of deploying software changes by keeping the functional aspects of the software changes hidden to end-users [18]. The motivation of using this technique is to get early feedback on the quality and performance of the software changes without letting the end-users know [18].

Implementation of dark launches varies amongst adoptees. For example, Facebook uses a tool called Gatekeeper that controls which software changes will go to which portion of the end-users [10].

Adoptees Using (9): CB, E, FB, FR, GCS, N, P, WF, RS
Adoptees Not Using (2): I, S

E. End-User Communication

End-user is defined to be the individual who uses the software [22]. We define the *practice of end-user communication* as the practice of communicating with end-users in order to receive feedback and gather requirements about the software of interest. Beck and Andres in their book identified communication with end-users to be essential to achieve successful results in software development [4]. Adoptees have used different techniques to communicate with end-users including use of official forums, phone calls, web seminars and social networking websites such as Twitter, and Facebook.

Table V presents the techniques used by adoptees to realize the practice of end-user communication.

TABLE V: TECHNIQUES USED TO IMPLEMENT END-USER COMMUNICATION

Technique Used	Adoptee	Count
Official Forums	E, FB, GCS, I, N, P	6
Phone Calls	WF	1
Social Networking Websites	G, S	2
Web Seminars	RS	1

Adoptees Using (10): CB, E, FB, G, GCS, N, P, RS, S, WF

F. Feature Flag

Feature flag, also known as *feature toggle* or *feature flipper* is a technique that facilitates in triggering a specific branch amongst several branches of the software code [13]. If the condition is satisfied then the corresponding branch of the code will be executed. If a portion of the deployed software changes is malfunctioning, then that portion can be switched off [13]. Feature toggles help adoptees to switch between different portions of the software repository and monitor

[4] https://jenkins-ci.org/
[5] http://buildbot.net/
[6] https://github.com/blog/1872-improved-audit-log
[7] https://help.github.com/articles/using-pull-requests/
[8] http://phabricator.org/

overall performance of the system for a certain portion of the software repository [31].

To implement feature flags adoptees have used conditional logic and configuration flags. For example, Rally Software uses a conditional framework [31], whereas Etsy uses configuration flags that switch on and off specific portions of the software code [5].

Adoptees Using (13): CB, E, FB, FR, G, GCS, K, N, O, P, RS, S, WF

G. Intercommunication

We define the practice of sharing all necessary development and deployment information amongst software team members as the *practice of intercommunication*. Sharing of development and deployment information helps software companies to achieve efficiency [32].

Our findings state that continuous deployment adoptees use a wide range of tools, including Chatops, conversation bots, and Gerrit. Gerrit is a web-based code review management system that also enables communication between software team members [15]. Conversation bots are automated tools that facilitate instant messaging using a certain protocol such as, Internet relay communication (IRC), instant messaging (IM), and XMPP. These conversation bots include XMPP bots, IRC bots, and IM bots. Chatops is a tool that execute scripts and other tools based on the typed command [40].

Adoptees Using (14): CB, E, FB, FR, G, GCS, K, N, O, P, RS, S, U, Y

H. Monitoring

Monitoring is the practice of collecting deployment related information, producing appropriate performance metrics, and reporting them in an appropriate format [22]. Monitoring is also referred as telemetry [16]. In a continuous deployment process, monitoring helps to identify the sources of errors in development and deployment quickly [19]. Monitoring also helps adoptees to get quick feedback on their deployment strategies [19] and the adoption/use of new features.

A wide range of automated tools is available as commercial and open source products, such as Graphite, Nagios, Splunk etc. [19, 42]. For example, IMVU uses a modified version of a real-time graphing software called Graphite[9].

Adoptees Using (16): CB, E, FB, FK, FR, G, GCS, I, K, N, O, P, RS, S, WF, Y

I. Repository Use

A software repository is termed as a software library that contains all the necessary software artifacts [22]. A collection of software artifacts that are derived from the software repository and is continuously changed is called a *branch* [22]. A branch corresponds to a specific software file version and is subject to deployment [22]. *Trunk* is defined as the main line of development of the software that is used to create branches [22]. We define the technique of pushing software changes using trunk as *trunk shipment*. Another way of pushing software changes to production is to create separate branches from the trunk, perform necessary changes and tests on that branch, and deploy that branch to end-users. We define this practice as *branch shipment*. We collectively define the practice of trunk shipment and branch shipment as the *practice of repository use*. The practice of repository use facilitates better management and easier backup of software artifacts [19].

We note that adoptees have used Git and SVN to implement the practice of repository use. Table VI presents the adoptees that are using the techniques of branch shipment and trunk shipment to implement repository use. This finding is in congruence with Humble et al.'s [20] recommendation to maintain one single, stable trunk and ship that trunk to production servers.

Adoptees Using (19): AT, CB, E, FB, FK, FR, G, GCS, I, KS, N, O, P, Q, RS, S, U, WF, Y

J. Shepherding Changes

We define *shepherding changes* as the practice of developers making software changes and being responsible for those software changes throughout the whole deployment process. The main motto of shepherding changes is getting involved in all the steps of the continuous deployment which includes writing software changes, running different tests on the software, deploying software changes into production, and fixing problems that arise after deployment. Use of this practice enhances developer responsibility and ensures delivering quality software changes to end-users without requiring a dedicated quality assurance team [10].

To implement the practice of shepherding changes adoptees use different techniques. For example, Facebook arranges boot camps for incoming developers so that they get habituated to Facebook's deployment and development practices [10]. Etsy, Github, IMVU, Netflix, and Wealthfront use an on call policy that requires software team members to fix any deployment problem for which they are responsible at the earliest possible time.

Adoptees Using (14): AT, CB, E, FB, FR, G, GCS, I, K, N, RS, S, WF, Y

Adoptees Not Using (2): P, Q

TABLE VI: TECHNIQUES USED TO IMPLEMENT REPOSITORY USE

Technique Used	Adoptee	Count
Branch Shipment	AT, N, Y	3
Trunk Shipment	CB, E, FB, FR, G, GCS, KS, O, P, Q, RS, S, U, WF	14
Unknown	FK, I	2

K. Staging

We define the *practice of staging* as the practice of executing a specific set of techniques by the adoptee after software changes are written, tested, and before software changes are deployed to end-users. We identify two techniques namely *dogfooding*, and *gradual rollout*. The benefit of using the practice of staging is to get early feedback on the software changes that are subject to deployment [19].

[9] http://graphite.wikidot.com/

Dogfooding is the technique when a software team uses its own software as part of their software development process [17]. One method to implement dogfooding is to use production servers that are accessible to the software team only and deploy software changes. Members of the software team will test out the software changes as an end-user. To perform dogfooding, Facebook makes the software changes available in their internal production servers using an internal web link[10].

Gradual rollout is the step-by-step process of deploying software changes to fractions of end-users [19]. For example, Facebook deploys software changes that are available in the internal production servers. If no error occurs then software changes are delivered to 1% of its end-users. If no further problem arises then the software changes are made available to all of Facebook's users [10].

Table VII presents the adoptees and the techniques they have used to implement staging.

Adoptees Using (17): AT, CB, E, FB, FR, G, GCS, I, K, N, O, P, RS, S, U, WF, Y

TABLE VII: TECHNIQUES ADOPTEES HAVE USED TO IMPLEMENT STAGING

Technique Used	Adoptee	Count
Dogfooding	AT, CB, E, FB, FR, G, GCS, I, N, P, RS, WF	12
Gradual Rollout	AT, CB, E, FB, FR, G, GCS, I, K, N, O, P, RS, S, U, WF, Y	17

VI. DISCUSSION

A. Common Continuous Deployment Practices

According to our findings, all 19 adoptees use the practices of automated deployment, automated testing, and repository use. This finding re-instates the necessity of applying these three practices mentioned above to implement continuous deployment. From our findings we also observe that as a software process continuous deployment necessitates the consistent use of sound software practices such as automated deployment, automated testing, code review, monitoring, and repository use that are strongly recommended by software practitioners.

B. Automated Testing

Swartout [42] along with Humble and Farley [19] described the importance of automated testing suites in a continuous deployment pipeline. All of the 19 adoptees use automated testing as a part of their development process. We observed that the implementation of automated testing suites, and types of testing used in development, varied from one adoptee to another.

Adoptees have used different types of testing as part of their deployment process such as unit testing, integration testing, functional testing, and A/B testing. Google Consumer Surveys uses an emerging type of testing called perceptual testing. According to a practitioner from Google Consumer Surveys, errors related to perceptual testing can lead to customer dissatisfaction and detecting these types of errors are non-trivial as they are often missed by the human eye, the automated tests, and monitoring graphs[11]. Perceptual testing can detect these errors automatically and release the burden of the software team[9].

Google Consumer Surveys performs perceptual testing as following: first they take snapshots of two versions of a web page, then they pair them by url path and finally they calculate the visual differences in pixels. To execute these steps they use PhantomJS[12].

Adoptee experience related to automated testing is also worth mentioning. For example, some adoptees have emphasized on the importance of speedy testing [11, 31]. Neely and Stolt stated that developers prefer fast running tests, and they recommended to minimize long running tests by breaking them up, and parallelizing them [31]. Kitchensurfing runs their test suite in more or less than 15 minutes [25]. IMVU uses 30-40 machines to run all their tests and each of their test run takes approximately nine minutes [11].

Some of the adoptees has also considered test coverage, the measure to which a single test or a set of tests satisfy all specified requirements for software components, to be an important aspect in the practice of automated testing [22]. For example, IMVU considers test coverage to be an important criterion to implement continuous deployment. IMVU emphasizes on 'reliable testing' which means 'that tests must not fail more often than once in a million test runs' [11]. According to Neely and Stolt, test suites that have good coverage help new members of the team to adopt continuous deployment quickly and deploy their own software changes with confidence [31].

C. Code Review

According to our findings, code review is widely used practice amongst adoptees. However, the motivation of using code review as a practice remains unknown for most of the adoptees. Facebook is one of the few adoptees that have described the reasons for using code review. Facebook uses the practice of code review to facilitate 'high quality code', 'find defects', 'suggest alternatives', and 'spreading general knowledge about coding practices' [10]. Mary and Tom Poppendieck discouraged the use of code review for 'finding defects' and recommended to use this practice for other uses such as ensuring simplicity in written software, complexity analysis, and ensuring absence of repetition [36].

D. End-User Communication

According to our study, the number of adoptees using this practice is 10. We were unable to identify if the other ten adoptees have used this practice as a part of their deployment.

[10] http://www.infoq.com/presentations/Facebook-Release-Process

[11] https://www.youtube.com/watch?v=1wHr-O6gEfc

[12] http://phantomjs.org/

E. Intercommunication

According to our study, adoptees are more relying on automated tools to share necessary development and deployment information amongst team members. We identify this finding as a major shift from the traditional approach where software teams have relied on team meetings, scrum meetings etc. [6].

F. Shepherding Changes

Our findings state that 14 of the 19 adoptees use the practice of shepherding changes. The practice of shepherding software changes takes a different stance than traditional software engineering where developers are only responsible to write and test their software changes and another team is responsible for deploying those software changes to production [37]. This finding also implies that developers are getting more involved in operations, supporting the basic principle of DevOps that encourages co-ordination among team members of the development team and the operations team [42].

VII. LIMITATIONS

A. Identifying Adoptees

We used two criteria to identify the 19 adoptees that we used in our study. These two criteria may generate false positives and false negatives.

B. Identifying Internet Artifacts

We relied on two search strings to search and identify Internet artifacts. We used a keyword-based approach to identify continuous deployment practices and in the process we might have missed Internet artifacts that describe the practices of more adoptees.

C. Extracting Information from Internet Artifacts

We did not use any automated tool or technique to extract information from the Internet artifacts and our methodology required manual book keeping. We leave the scope of applying any automated technique to extract information from Internet artifacts as future work.

In many cases, the Internet artifacts used provided challenges. For example, software practitioners write blog posts informally and extracting necessary information about practices became challenging. To overcome this limitation of Internet artifacts we used follow-up inquiries.

D. Identified Continuous Deployment Practices

We used Feitelson et al.'s [10] paper to identify the set of continuous deployment practices. We cannot claim that this set is complete. We have observed that none of the other 18 software companies prevalently used software practices that are not included in the set of 11 continuous deployment practices.

E. Importance of the Identified Continuous Deployment Practices in Continuous Deployment

We were unable to determine why the 11 continuous deployment practices are important in continuous deployment due to lack of empirical evidence. Understanding how these 11 practices help adoptees to achieve continuous deployment successfully can be beneficial to software engineering practitioners and we leave this topic as future work.

F. Follow-up Inquiries

The limitation of using follow-up inquiries is we might not find the contact of interest or the contact of interest might not reply. This is a limitation of our methodology and as a result we were unable to determine if some adoptees have used some of the continuous deployment practices. The combination of Internet artifacts and follow-up inquiries was not enough to discover the use of all continuous deployment practices by the 19 adoptees along with the techniques to implement them.

VIII. CONCLUSION

Systematic analysis of software practices used in continuous deployment can not only help software practitioners to understand continuous deployment as a software process but also facilitate them in adopting continuous deployment. This paper summarizes the software practices used in industry to implement continuous deployment, and the adoptees that are using these practices. Instead of walking the painful path of searching and learning from Internet artifacts, software practitioners can use this paper to learn the industry practices used in continuous deployment. We observe that all the 19 software companies use automated testing and automated deployment, the two pre-requisite software practices for continuous deployment. In our paper, we also have observed that continuous deployment necessitates the consistent use of sound software engineering practices, such as automated deployment, automated testing, and code review. Future research in continuous deployment might identify the appropriate reasoning that will explain this observation. We also identify the scope of future research that will provide guidelines on how software companies can use the identified 11 software practices to implement continuous deployment effectively.

ACKNOWLEDGMENT

We thank all the software practitioners who responded to our questions and provided us with valuable information related to their company's continuous deployment practices. We also thank the Realsearch research group for providing helpful feedback on this paper.

REFERENCES

[1] G. Azizyan, M. K. Magarian, and M. Kajko-Mattson, "Survey of Agile Tool Usage and Needs," in *Proceedngs of the Agile Conference*, 2014

[2] A. Bacchelli, and C. Bird, "Expectations, oucomes, and challenges of modern code review," in *Proceedngs of the International Conference on Software Engineering*, 2013

[3] K. Beck, M.Beedle, A. Bennekum, A. Cockburn, W.Cunningham, M. Fowler, J. Grenning, J.Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas (2001, February 13). The agile manifesto [Online]. Avaialable: http://agilemanifesto.org/

[4] K. Beck, and C. Andres, "Extreme Proramming Explained: Embrace Change," 2nd Ed. Addison-Wesley, 2004

[5] M. Brittain (2013, January 29). Continuous Deployment: The Dirty Details[Online].Available:http://www.slideshare.net/mikebrittain/mbrittain-continuous-deploymentalm3public?related=1

[6] T. Chau, and F. Maurer, "Knowledge Sharing in Agile Software Teams," in *Logic versus Approximation*, vol. 3075, pp. 173-183, January, 2004

[7] G. G. Claps, R. B. Svensson, and A. Aurum, "On the journey to continuous deployment: Technical and social challenges along the way," in *Information and Software Technology*, vol. 57, pp. 21-31, January, 2015

[8] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003

[9] DevOpsGuys (2013, March 13). Continuous Delivery Adoption Barriers | DevOpsGuys [Online]. Available: http://blog.devopsguys.com/2013/03/13/continuous-delivery-adoption-barriers/

[10] D. G. Feitelson, E. Frachtenburg, and K. L.Beck, "Development and Deployment at Facebook," in *IEEE Internet Computing*, vol. 17, pp. 8-17, July –August, 2013

[11] T. Fitz (February 10, 2009). Continuous Deployment at IMVU: doing the impossible fifty times a day [Online]. Available: http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

[12] M. Fowler (2013, May 30), Continuous Delivery [Online]. Available: http://martinfowler.com/bliki/ContinuousDelivery.html

[13] M. Fowler (2010, October 29), Feature Toggle [Online]. Available: http://martinfowler.com/bliki/FeatureToggle.html

[14] Gartner (October 08, 2014). Gartner Identifies the Top 10 Strategic Technology Trends for 2015 [Online]. Available: http://www.gartner.com/newsroom/id/2867917

[15] Google Project Hosting (2015, March 27). Gerrit - Gerrit Code Review – Google Project Hosting [Online]. Available: https://code.google.com/p/gerrit/

[16] K. C. Gross, A. Urmanov, L. G. Votta, S. McMaster, and A. Porter, "Towards Dependability in Everyday Software Using Software Telemetry," in *Proceedings of the Third IEEE International Workshop on Engineering of Autonomic and Autonomous Systems*, 2006

[17] W. Harrison, "Eating Your Own Dog Food," in *IEEE Software*, vol. 23, pp. 5-7, May-June, 2006

[18] J. Humble (2012, Februrary 16). Principle 2: Decouple Deployment and Release[Online].Available:http://www.informit.com/articles/article.aspx?p=1833567&seqNum=2

[19] J. Humble, and D. Farley, "Continuous Delivery," 1st Ed. Addison-Wesley, 2011

[20] J. Humble, J. Molesky, and B. O' Reilly, "Lean Enterprise," 1st Ed. O' Reilly Media Inc., 2015

[21] J. Humble, C. Read, and D. North, "The Deployment Production Line," in *Proceedngs of the Agile Conference*, 2006

[22] IEEE Standards Association (2010, December 15). IEEE SA – 24765 – 2010 – Systems and software engineering – vocabulary [Online]. Available: https://standards.ieee.org/findstds/standard/24765-2010.html

[23] ISO/IEC/IEEE (2013, September 01). ISO/IEC/IEEE 29119-1:2013 Software and systems engineering -- Software testing - Part 1: Concepts and definitions [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=45142

[24] N. Kerzazi, and F. Khomh, "Factors Impacting Rapid Releases: An Industrial Case Study," in *Proceedngs of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014

[25] L. Kluge (2013, July 17). Continuous Deployment at Kitchensurfing [Online]. Available: http://www.slideshare.net/LarsKluge/1-24352219

[26] L. Lagerberg, T. Skude, P. Emanuelsson, and D. Stahl, "The impact of principles and practices on large-scale software development projects – A multiple-case study of two projects at Ericsson," in *Proceedngs of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013

[27] M. Leppanen, S. Makinen, M. Pagels, V. Eloranta, J. Itkonen, M. Mantyla, V. Mika, and T. Mannisto, "The Highways and Country Roads to Continuous Deployment," in *IEEE Software*, vol. 32, pp. 64-72, March-April, 2015

[28] E. Murphy-Hill, "The Future of Social Learning in Software Engineering", in *Computer*, vol.47, pp. 48-54, January, 2014

[29] G. A. Moore, "Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Customers," Revised Ed. Collins Business Essentials, 2002.

[30] S. Narayanan (2013, Novermber 18). The Netflix Tech Blog: Prepapring the Netflix API for Deployment [Online]. Available: http://techblog.netflix.com/2013/11/preparing-netflix-api-for-deployment.html

[31] S. Neely, and S. Stolt, "Continuous Delivery ? Easy ! Just Change Everything (well, maybe it is not that easy)," in *Proceedngs of the Agile Conference*, 2013

[32] M. Olofsson, "Managing knowledge sharing in software development organizations," M. S. thesis, Linkopping University, Linköping, Sweden, 2012.

[33] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the 'Stairway to Heaven' – A multiple-case study exploring barriers in the transition from agile development towrds continuous deployment of software," in *Proceedngs of the Euromicro Conference on Software Engineering and Advanced Applications*, 2013

[34] M. Paasivaara, B. Behm, C. Lassenius, and M. Hallikainen, "Towards Rapid Releases in Large-Scale XaaS Development at Ericsson," in *Proceedngs of the International Conference on Global Software Engineering*, 2014

[35] C. Passos, D. S. Cruzes, A. Hayne, and M. Mendonca, "Recommendations to the Adoption of new Software Practices-A Case Study of Team Intention and Behavior in Three Software Companies," in *Proceedngs of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013

[36] M. Poppendieck, and T. Poppendieck, "Implementing Lean Software Development – From Concept To Cash," 1st Ed. Addison-Wesley, 2007

[37] R. S. Pressman, "Software Engineering – A Practitioner's Approach," 2nd Ed. McGrawHill, 2010

[38] P. Rodriguez, J. Markkula, M. Oivvo, and K. Turula, "Survey on Agile and Lean Usage in Finnish Software Industry," in *Proceedngs of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2012

[39] R. Sanheim (September 18, 2014). Continuous Delivery at Github [Online]. Available: https://speakerdeck.com/rsanheim/continuous-delivery-at-github

[40] E. Sigler (2014, December 02). What is ChatOps ? And How Do I Get Started ? [Online]. Available: http://www.pagerduty.com/blog/what-is-chatops/

[41] S. Smith (April 17, 2014). Practical continuous deployment [Online].Available: http://blogs.atlassian.com/2014/04/practical-continuous-deployment/

[42] P. Swartout, "Continuous Delivery and DevOps: A Quickstart Guide," 1st Ed. Packt Publishing, 2012

[43] N. F. Velasquez, G. Kim, N. Kersten, and J. Humble, "2014 State of DevOps Report," https://puppetlabs.com/sites/default/files/2014-state-of-devops-report.pdf, 2014.