Sai Subhashree Int. Journal of Engineering Research and Application *www.ijera.com*
ISSN : 2248-9622, Vol. 8, Issue 4, (Part -3) April 2018, pp.126-134

RESEARCH ARTICLE OPEN ACCESS

The Impact of Context on Continuous Delivery

Mandakini Priyadrshini Behera, Sai Subhashree
Gandhi Institute of Excellent Technocrats, Bhubaneswar, India
Nigam Institute of Engineering and Technology, Bhubaneswar, Odisha, India

ABSTRACT
This paper will evaluate how the properties of production environment and software, which is continuously delivered, have influence on the implementation of Continuous Deliv- ery. The evaluation is based on three case studies from dif- ferent software development domains. The first case study deals with the way the software engineers at Etsyuse Con- tinuousIntegration for the delivery of their App. The second example is about Box's decision to introduce Continuous Deployment in order to continuously deploy their desktop software Box Sync to its customers. The last example is about the Hewlett-Packard LaserJet Firmware Team which implemented Continuous Delivery with great success.
These case studies will show that UI (user interface) com- plexity, the lack of control over the production environment and the quality of software simulators, which simulate the production environments, are properties or derived proper- tieswhichhaveimpactontheimplementationofContinuous Delivery.
Categories and Subject Descriptors
D.2.7 [**SoftwareEngineering**]: Distribution, Maintenance, andEnhancement
Keywords: Continuous Delivery, Production Environment

I. INTRODUCTION

Continuous Delivery is a software development discipline which enables "Reliable Software Releases through Build, Test, and DeploymentAutomation"[9].

This paper will evaluate the impact of context on Continu- ous Delivery implementation for software development do- mainswhicharedifferentfromtheclassicaldomai nsofnone UI heavy backend and web applications. Software and pro- duction environment properties present the context evalu- ated in this paper. The evaluation is based on threedifferent

.

case studies.

The structure of the paper is as follows. Section 2 provides all important definitions and a short introduction to Con- tinuous Delivery. Section 3 covers the three case studies, which are first described and then analyzed. The first case study is about the way Etsy, "often trotted out as a poster child for Devops" [3], introduced Continuous Delivery for their Apps (mobile applications). The second one comes from the company Box which recently implemented Contin- uous Deployment for their desktop software Box Sync. The last example is the very well documented case of the HP LaserJet Firmware Team, which increased their productiv- ity dramatically by implementing Continuous Delivery.
After the analysis and description of all three examples they are compared to each other in section 4 in order to identify similarities of properties which impact the Continuous De- livery implementation. The last section 5 will conclude the paper and provide ideas for future work.

TERMDEFINITIONS

In this section the most important terms are briefly intro- duced. More detailed information can be found in the cited sources.
Continuous Delivery and the Deployment Pipeline

The three terms Continuous Integration, Continuous Deliv- eryand Continuous Deployment and their coherences are easily confused and will therefore be defined in this section. The process of Continuous Delivery starts with continuously integratingcode.**ContinuousIntegration**"isaso ftware development practice where members of a team integrate their work frequently, [...].

Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible." **Continuous Delivery** goes one step further and ensures that"software is build in such a way that the software can be released to production at any time" and the software is deployable throughout its whole life cy- cle. The last step towards complete automation is **Contin- uous Deployment**. "Continuous Deployment means that every change goes through the pipeline and automatically gets put into production, resulting in many production de- ployments every day."[12]

The Core of a Continuous Delivery implementation is the **DeploymentPipeline**,whichmodelstheprocess ofget- ting "software from version control into the hands of your users" [7]. As figure 1, which illustrates a basicdeployment pipeline,showssomestagesofaDeploymentPipel ineareau-

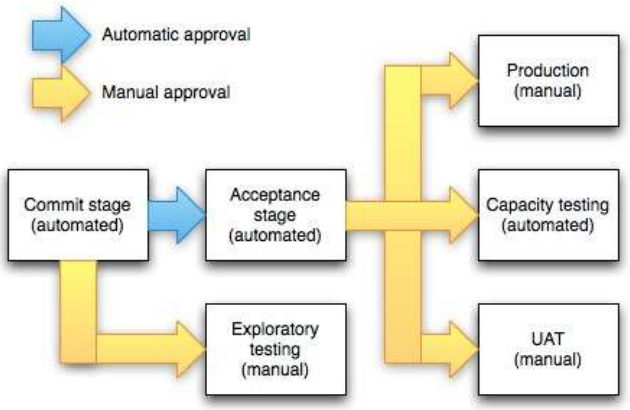

Figure 1: Basic Deployment Pipeline[8]

tomatedandsomeneedmanualinteraction.Thet hreecase studiesinsection3willshowthatthelevelofauto mationis differentforeachspecificimplementation.Thef irststageis thecommitstage.Thefollowingtwostages,auto matedac- ceptancetestsandamanualtestingstage,canbee xecuted inparallel.Thelastthreestagesareparallelagain andcon- sist of the UAT (user acceptance tests), capacity tests and going into production stage[8].

The term classical Continuous Delivery refers to Continu- ous Delivery for web or backend applications with none or very littleUI.

Environment

An **environment** in the context of software development is one specific combination of hardware properties and soft- ware properties which build a platform to run software on. Through the Continuous Delivery process the following en- vironments may occur.

In traditional software development there are four differ- ent environments for a software from development to pro- duction.Thefirstoneisthe**developmentenviron ment**, which represents the working environment of the developer. Afterthedevelopmentenvironmentcomesthe**int egration environment**wherethecodechangesfromalldev elopers are combined and integrated. For smaller projects the first two environments could be the same. The **staging envi- ronment**shouldbeassimilaraspossible,ideallyid entical, to the production environment. It is used to simulate pro- duction. The **production environment** is the environ- ment the software was developed for[14].

CASESTUDIES

This section covers the three case studies on which the evaluation in section 4 is based on. Each case is briefly de- scribedandfollowedbyananalysis.Thegoalofthe analysis istofindoutwhichpropertiesoftheproductionenvi ronment and the software had most impact on the specific implemen- tation of Continuous Delivery. The analysis part itself will focus on the following two questions:

Which properties of the production environment and the software have influence on Continuous Delivery?

How is this reflected in the implementation of Contin- uous Delivery?

The results of the analyses are

compared to each other in section 4 to identify similarities and determine what had the biggest impact regarding difficulties and unresolved prob- lems.

Mobile Application:Etsy

The first case study is taken from an article which de- scribes how Continuous Delivery is implemented for an App at Etsy. "Etsyis a marketplace where people around the worldconnect,bothonlineandoffline,tomake,sel landbuy unique goods." [4] The example was chosen since it points out the challenges of Continuous Delivery in the context of an App. The article mainly focuses on the iOS Continuous Delivery stack because at the time of writing the Android stack wasn't as well developed as its iOScounterpart[13].

Description

Etsy decided to use Continuous Integration since "through Continuous Integration, they can detect and fix major de- fects in the development and validation phase of the project, before they negatively impact user experience".

Automated Continuous Delivery at Etsy for mobile apps can be summarized in one sentence: "Every commit builds the mainline on special integration machines". So after every commit by a developer an integration server (Jenkins [10]) executesabuildplanwhichconsistsofmorethan15 jobsby using the integration machines and notifies the developers in case of a failure. There is also a simple homegrown dash- board which "communicates the current test status across all configurations" [15]. The whole CI infrastructure from Etsyis illustrated in figure2.

The biggest challenge was to setup an integration and test environment which covers all important devices. For iOS every build has to be tested on "seven different iPads, five iPhones and a few iPods" [15]. But for Android it is even worse because the number of Android devices to cover by tests is overwhelming. As integration and test en- vironmentthereisa"fleetofMacminis"whichare allnearly fully automatically provisioned. Additionally real devices in thecloudfromAWSdevicefarm[1]areusedfortest ing.The setup of the integration machines could not be fully auto- mated because of the "inability to automate the installation of some software dependencies". Especially the installation and setup of the iOSIDE Xcodestill needs some manual interaction.

With these integration machines the code is build and tested after each push to the repository to get immediate feedback. Regression test are run nightly on a broader range of real devices [15].

Since "most of the core logic of Etsy's Apps relies on the UI layer" the software engineers at Etsy focus on functional testing which mimics the steps of an actual user. The test includeactionslike"searchingforlistingsandshop s","regis- tering new accounts" and "purchasing an item with a credit card or a gift card". One example for a concrete functional tests is the checkout test. For this test a buyer and seller test account is created and a real credit card is used [15]. The test is asfollows:

1. "Signingintotheappwithatestbuyerac count."[15]
2. "Searchingforanitem(inthesellertestacc ountshop)." [15]

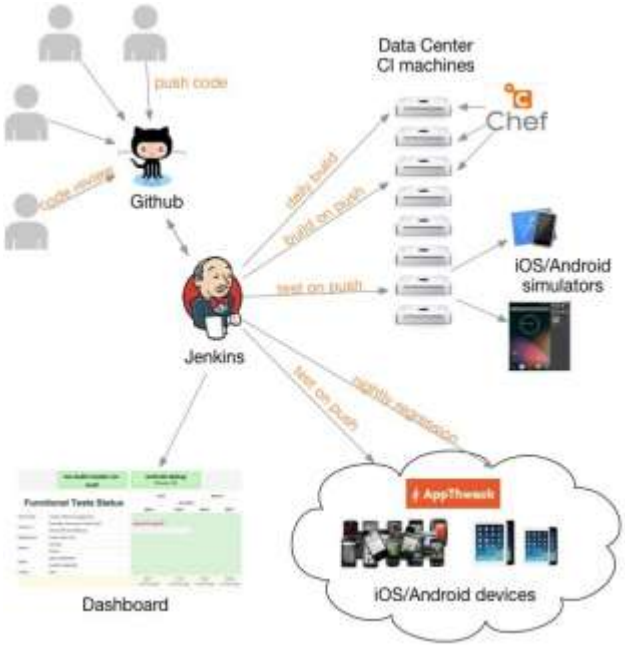

Figure 2: Etsy'sCI infrastructure overview[15]

3. "Adding it to the cart."[15]

4. "Paying for the item using the prepaid credit card." [15]

These tests run on simulators and real devices. Integration tests, executed after every push, are run in simulators on the integration machines of Etsy. Nightly Regression tests are outsourced to AWS device farm [1] (previously known as Appthwack), which provides the possibility to test Apps on a broad range of real mobile devices [15]. Since it is nearly impossible to test on all possible device configura- tions, devices are chosen based on Google Analytics data. Since the integration happened only recently there were still someproblemsrelatedtotestingonphysicaldevic esandthe challenges of aggregating and reporting test status from all the devices when the article was published[13].

In addition to the automatic integration testing there are layers of manual QA (quality assurance). An internal build is released daily which Etsy employees are encouraged to install on their devices [15]. Another manual test is called "app rotations": "Eight volunteers gather in a room, accom- paniedby a QA facilitator and a mix of devices. The goal is to find as many bugs as possible in a predefined timebox." [13]

After all automated and manual tests are passed the App is submitted to the App Store for approval which will take around five days [15]. So if a bug slips through all the tests and is discovered while the App is already running on the users devices, it takes a minimum five days to get an update to theusers.

Analysis

The goal of the software engineers at Etsywas to implement fully automated Continuous Delivery for their Apps. They automated the process as far as possible from pushing the code into the repository to submitting the App to the App Store. During the implementation of Continuous Delivery they were faced with two major challenges.

The first challenge was the setup of the environments for integration and testing. For testing the Android and iOS Apps, either a software simulators or real devices is neces- sary to run the Apps on. Simulators for Android and iOSdon'tmodelrealdevicescloselyenoughandpr ovedtobein- sufficient [17], thus simulators are only used for integration tests and real devices have to be used for regressiontests. So the first property with influence on the Continuous Delivery implementation is the inability to properly model mobilede- vices with softwaresimulators.

The need to use real devices for tests results in the next problem. Which devices should be used for the tests? The possible production environments are all iOSand Android devices

with an App Store or Google Play Store installed. While for iOSthere is a limited number of different devices and versions, the number of Android devices and versions is far to big to run tests on all of them, because that would lead to unmanageable number of devices to run tests on. Opensignalreports there are over "24,000 distinct Android devices seen in 2015" [16]. So it's impossible to cover all existing devices with tests and a specific set of iOSand An- droid has to be chosen for tests. Thus the second property with influence on the implementation of Continuous Deliv- eryis a to big variety of possible production environments which have to be covered withtests.

The second challenge was the automated testing of the UI. The functional tests used for this purpose will only discover basic bugs and App crashes. Additionally the aggregation and evaluation of the test data from the AWS device farm devices used for regression testing is still an unsolved prob- lem, which is again a result of the variety of possible pro- duction environments. These problems made it necessary to add the described manual QA stages. So one identified property of the production environment which has impact on Continuous Delivery is UI complexity. It results in the inability to fully automate tests, which makes manual qual- ity assurancenecessary.

Another property is control over the deployment process. The five days approval process of the App Store was one reason for an additional manual test stage. So the inability todeployahotfiximmediatelyresultsinevenmore accurate testing.

Desktop Application:Box

Box is a company which provides "secure content and on- line file sharing for businesses"[2]. One part of theirproduct is their desktop software Box Sync which syncs their cus- tomers desktop computers with Box 's online services. "In an effort to maintain the agility of our startup days and deliver the best software possible, Box has been moving to- wards Continuous Deployment". Since the software engi- neers at Box had huge success with Continuous Deployment and web development they decided to use their experience and knowledge and adapt it for their desktop software[18].

Description

"In order to do Continous Deployment you must be doing Continous Delivery" [12]. Therefore the team at Box imple- mented "automated acceptance testing" in a first step. Basicfunctionality of Box Sync, syncing files from one computer toanother,iseasytotestsincenetworkandfilesyste mcould easily be simulated [18]. They used standard best practice for webdevelopment:

5. "Every time a developer pushes a new commit, the ap- plication is built in its entirety ("continuous integra- tion") and the full suite of tests is run."[18]
6. "If a test fails, the build cannot be deployed and fur- ther commits are rejected until the test failure is fixed ("stop the line")."[18]

SincetheBoxSyncsoftware"islightonUIanditsba sicjob- ensuringtwosetsoffilesintwodifferentplaces match-isvery easy for a computer to verify" [18] there is full code cover- age through unit tests. They have three types of automatic integrationtests:

1. Full code coverage via unittests.[18]
2. "Themainsyncingalgorithmiscoveredbyint egration- style tests which simulate the network and file system called B to Y (the local file system is A, and the net- work is Z)."[18]
3. "Full-scale integration tests that launch the builtver- sionofSync(thefull.appor.exe,depending onplat- form),playwithfilesonthelocalharddriveo ronBox, andverifytherightthingsendupintherightpl aceat the end. They call this "chimp"."[18]

All of the described tests are run on each supported plat- form and operating system. Since the Box Sync software relies heavily on the Box web API, another suit of integra- tion tests called "chimp-staging" is run to ensure compati- bility. But "deploying client software is completely unlike deploying a web app, so their first goal was to make thepro- cess as consistent as possible, while respecting the different domain requirements and maintaining high user experience standards"[18].

As a result of Continuous Deployment of Box Sync all up- dates had to be backward compatible to a lot of prior ver- sions since the production environment of Box Sync are desktopcomputerswhichmightbeofflinefordays orweeks. The risk of rendering a client useless with a failed update is too high and therefore older versions of Box Sync are manu- ally updated with consecutive updates before the release of a new version[18].

To make sure that there are no problems during the auto- mated deployment the Box Sync clients are monitored re- motelyandbadthingslike“exceptions,errors,or warnings theclientsencounter”arereported.Butalsothin gslikeup- loads, downloads, and authentication session renewalsare monitoredtoassurethattheclientsdon’tstopwor kingcom- pletely.Allthedataisaggregatedbytheclientand sendto theserversinabandwidthsavingmanortopreve ntDenial- of-service attacks by the own client[18].
But there are still three problems to be solved for real Con- tinuous Deployment:
1. “Shipping a complete copy of the application multiple times a day would saturate bandwidth. Differential updates could solve this problem.”[18]
2. TheUIelementsofBoxSyncarestillchec kedmanually for each platform[18].
3. The reading of feedback from the clients is not auto- mated[18].

The author of the article summarizes the implementation of Continuous Deployment for Box Sync as follows: “One of the things we learned while building Box Sync is that even if we cannot reach true continuous deployment for technical reasons, having it as a goal makes a strong, positive impact on our culture and development practices.”[18]

Analysis

The Box team had a lot of experience with Continuous Deployment for web applications and tried to apply their knowledge to the delivery and deployment of their desktop software Box Sync. This worked out very well for the Con- tinuous Integration stages of their pipeline because the core functionality of Box Sync was easy to verify and the imple- mentation was very similar to an implementation for classi- calContinuous Integration.

The regression test stage, which was executed on “real com- puters”, however could only be partly automated. The core functionality was again easy to test since there was no com- plex UI and the result of an test could easily be verified automatically on real computers. UI tests in contrast were too complex for the Box Sync team to implement and there- fore the UI is tested manually before each release. So again the UI couldn’t be tested fully automated.

But the major challenge for the Box Sync team was to keep each release backward compatible to prior releases. This problem was a result of no control over the production en- vironment since it’s an decision of the customer when the client is online and can update itself. This is a big difference to web servers, the target environment for classical Contin- uous Delivery, which are fully owned and are mostly incre- mentally updated. This problem couldn’t be solved with automated tests instead they had to add a manual approval stage.

Embedded System: HP PrinterFirmware

The last case study is about the HP LaserJet Firmware Teamwhichmadetheirwayoutofacrisisandincrea sedpro- ductivityby implementing Continuous Delivery. The whole processisverywelldocumentedinthebook“APra cticalAp- proachtoLarge-ScaleAgileDevelopment”[6]bytheproject leader Gary Gruver, which I recommend for further details. This case study was selected since it’s completely different from the other two case studies and shows that fully auto- matedContinuousDeliveryispossibleforsoftwar edevelop- ment domains different from web andbackend.

When Gary Gruver joined the HP LaserJet Firmware Team theyspentonly5%oftheirresourcesondeveloping newfea- tures and the average time of one regression test cycle was six weeks. This is why they decided to implement Continu- ous Delivery and changed the architecture of their software. We will focus on the implementation of Continuous Integra- tionas described in chapter 6 of Gruver’s book[6].

Description

Before they implemented Continuous Delivery they had to change the structure of the code first. They reorganized theircodebaseandchangedfrommultiplebranche s,one

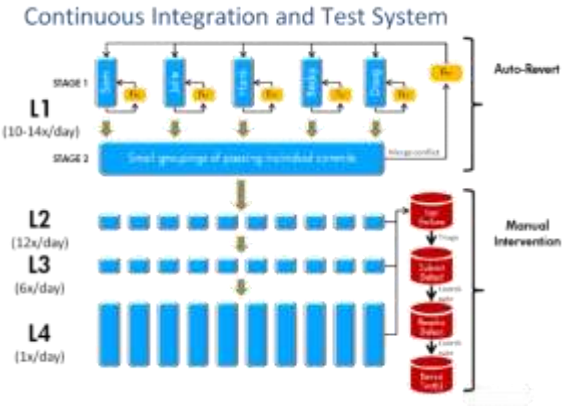

Figure 3: Continuous Delivery system at HP [6]

for each printer model, to one single branch. Instead of defining the specific capabilities for each printer with a C #ifdefdirective [5] they used XML configuration files for the definition of the capabilities. For the integration tests they developed their own printer simulators and deployed them on 2000 virtual servers. For the later test stages they used hardware emulators to get more accurate results [11]. Figure 3 shows the Continuous Integration and test setup of the HP LaserJet Firmware Team. The system has four different levels of testing[6]:

- **L1**: Is executed after each commit. If there is a failure it will be automatically reverted.
- **L2**: More detailed tests, which run every 2 hours and use last working commit from L1. If a test fails an email with everything needed to replicate the failure is sent to the developer who committed the code.
- **L3**: Same as L2 but runs on dedicated emulator hard- ware every 4 hours.
- **L4**: All automated tests are combined to a regression test suite and run daily around midnight. "Provides a complete view of the quality of the system" and is an indicator for the release readiness of the firmware.

With the introduction of Continuous Delivery the HP LaserJet Firmware Team could strikingly decrease the time and resources needed for code integration and tests. While the team spent 10 % of their resources for code integration before the changes now it's only 2 %. They could decrease the resources needed for testing from 15 % to 5 %. This allowed them to spend 40 % instead of 5 % on new features and innovations [6].

Analysis

The HP team also had the challenge of covering multiple production environments with tests. But in contrast to the other two case studies, their models of the simulators are very good and they have full control over each possible pro- duction environment. This way they could reach full cover- age of all possible production environments. Additionally, since there was no complex UI, all the test data could be evaluated automatically and for some tests there was also automated feedback to the developers. But there was also theproblemofsoftwaresimulatorsnotbeinggoodenough andthereforetheyusedhardwaresimulatorsfortheregres- sion teststage.

EVALUATION

This section sums up and evaluates the results of the anal- ysis parts in Section 3.

UI complexity

The first two case studies showed that the level of UI com- plexity of the software has a big influence on the degree of manual test stages required for Continuous Delivery. To reachfulltestcoverageforanUIheavysoftwareall possible input paths have to be covered and each result has to be ver- ified. User Input can be simulated with the help of scripts. The problem is the automated aggregation and evaluation of the test data from all devices. The software engineers at Etsycan only detect crashes and low level bugs with au- tomated UI tests. The UI of Box Sync is tested manually because implementing tests would be too complex. Thecase study from HP, in contrast, is a good example for software with very little UI and UI interaction of the user. As

a result they could completely automate theirtests.
Therefore UI complexity is one property of the software which has an impact on the implementation of Continuous delivery.

Lack of control over the production environment

The impact of lack of control over the production environ- ment showed itself in three different variants.

The first one comes from the Etsycase study which showed that if there are no constraints on the configuration of the production environments, that could lead to a fragmenta- tion of the production environment. This might result in an unmanageable number of possible productionenvironments. This again results in the problem how to implement the in- tegration and test environments, since it is impossible to run tests on all possible production environments. The HP case study in contrast shows that it is possible to cover all production environments with tests, even if there is a big number ofthem.

The second variant about lack of control over the production environments is the lack of control over when and how up- dates are deployed to the production environment. The Box Sync case shows that if you continuously deploy your soft- ware into production there might be problems because some clients skip updates and therefore updates have to be com- patible to all prior versions. So the lack of control over the production environment could lead to an additional manual approval stage

The last one is a result of no control over the deployment process. Bugs that slip into production can't be immedi- ately fixed with a hotfix. This makes it necessary to test several nightly builds manually before every release.
So the lack of control over the production environments has a lot of impact on all stages which are connected to tests.

Quality of software simulators

The HP and Etsycase studies showed that quality of soft- ware simulators, which simulate the production environ- ment,haveimpactontheteststagesofContinuous Delivery. Tests with simulators are mostly not sufficient since simu- lators are unable to properly imitate some properties of the production environment. Therefore the software running on simulators won't show the same behavior and performance as on the real devices. As a consequence, tests in simulators won't discover all bugs that are found with tests on real devices. For this reason in both case studies from HP and Etsysimulators are only used for early test stages. But with the use of hardware based simulators or real devices the ag- gregation and evaluation of test data is more complex. This results in more effort for the implementation of tests or even inability to process the dataautomatically.
So with decreasing quality of software emulators for an pro- duction environment the complexity of tests increases.

II. CONCLUSION & FUTUREWORK

The analysis of the case studies showed some of the pos- sible impacts of production environment and software prop- erties on the implementation of Continuous Delivery.
The following three properties were extracted from theanal- ysis of the casestudies:
With increasing **UI complexity** the test data from UI tests can't be processed automatically and manual test stages are necessary.
The **Lack of control** over parts or the whole of the productionenvironmentinfluencestheimplemen tation of the different teststages.
Qualityofsoftwaresimulatorsisconnectedtocom- plexity of the teststages.
Sincethispapercouldonlyevaluatealimitednumb erofcase studies examples from other software development domains shouldbeexaminedtoconfirmandexpandtheresu lts.
One of the consequence of the found properties is the need for additional test stages which require manual interaction. Especially the evaluation and feedback for UI tests are done manually for two of the three case studies. In order to solve this problem further investigation of UI testing is necessary to identify the reasons which prevent the full automation. The lack of control over the production environment com- bined with an unmanageable number of possible production environments made it impossible to reach full test coverage for them. There are two problems suitable for further inves- tigation. The first one is how to prevent the fragmentation of a production environment which leads to an uncontrol- lable number of possible production environments. And if it can'tbepreventedhowtomaximizethecoverageo frelevant productionenvironments.
The case studies also showed that if the production environ- ment is fully under control of the team and UI complexity is low it's

possible to implement fully automated Continuous Delivery.

REFERENCES

[1] Amazon. Aws device farm.http://aws.amazon.com/device-farm/?nc1=f_ls,2015. Retrieved December 03,2015.

[2] box. box.com. https://www.box.com, 2015. Retrieved December 06,2015.

[3] L. Chen. Continuous delivery: Huge benefits, but challenges too. Software, IEEE, 32(2):50–54, Mar 2015.

[4] Etsy. About etsy. https://www.etsy.com/de/about/, 2015. Retrieved December 06,2015.

[5] gnu.org. Ifdef, 2015. Retrieved December 16,2015.

[6] G. Gruver, M. Young, and P. Fulghum. A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmartFirmware. Addison-Wesley Professional, 1st edition,2012.

[7] J. Humble. Continuous delivery: Anatomy of the deployment pipeline. http://www.informit.com/articles/article.aspx?p=1621865, 2010. Retrieved December 11,2015.

[8] J. Humble. Deployment pipeline anti-patterns.http://continuousdelivery.com/2010/09/ deployment-pipeline-anti-patterns/, 2010. Retrieved December 12,2015.

[9] J. Humble and D. Farley. Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley Professional, 1st edition,2010.

[10] W. Jenkins. Meet jenkins, 2015. Retrieved December 15,2015.

[11] G. Kim. The amazing devops transformation of the hplaserjet firmware team (garygruver).http://itrevolution.com/the-amazing-devops- transformation-of-the-hp-laserjet-firmware- team-gary-gruver/, 2015. Retrieved December 06, 2015.

[12] F. Martin. Continuousdelivery. http:// martinfowler.com/bliki/ContinuousDelivery.html, 2013. Retrieved November 22,2015.

[13] J. Miranda. How etsy does continuous integration for mobile apps.http://www.infoq.com/news/2014/11/continuous-integration-mobile, 2014. Retrieved December 03, 2015.

[14] P. Murray. Traditional development/integration/staging/production practice for software development, 2006. Retrieved December 18,2015.

[15] K. Nassim. Etsy's journey to continuous integration for mobile apps.https://codeascraft.com/2014/02/28/etsys-journey-to- continuous-integration-for-mobile-apps/, 2014. Retrieved November 22, 2015.

[16] opensignal. Android fragmentation visualized (august 2015).http://opensignal.com/reports/2015/08/android-fragmentation/, 2015. Retrieved December 19,2015.

[17] M. Poschenrieder. Testing on emulators vs real devices, 2015. Retrieved December 17,2015.

[18] B. Smith. Continuous deployment in desktop software. https://www.box.com/blog/continuous-deployment-in-desktop-software/, 2013. Retrieved December 03,2015.