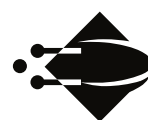


# Automating Deployments in Mobile App Development

## Advanced Software Engineering (F2015)

Kristian S. M. Andersen

Supervisor: Yvonne Dittrich  
Submitted: May 13, 2015



**IT University**  
of Copenhagen

# Abstract

This is an abstract

# 1 Introduction

In the development of large scale web-infrastructures it is common to use Continuous Delivery techniques to facilitate small rapid development cycles. These techniques are not as widely adopted when it comes to smaller products such as Apps installed on mobile devices. Many of the challenges are the same though the tools differ a bit.

I work in a small danish company that develops Apps for the iPhone and iPad. The company has 17 Apps in production which is developed and maintained by 7 developers. Most of the apps in production needs both weekly and daily builds. This constitutes a problem since every app has a different build process and none of the processes are automated. A build can take anything from 10 minutes to 2 hours.[2]

## 1.0.1 Definitions

**Continuous Integration** The practice of merging all developer working copies of a project into a mainline as first proposed by Grady Booch.

**Continuous Delivery** The process of delivering a software product directly to the customer in small rapid software development cycles.

**Apps** Applications developed for mobile devices such as the iPhone or iPad.

## 1.0.2 Structure of the Paper

In section 2 I will further refine the problem and highlight three key aspects that needs to be addresses by the solution. In section 3 I will relate the problem to the literature around the subject. In section 4 I will propose a solution to solve the problem. In section 5 I describe how to

evaluate the proposed solution against the problem. Finally in section 6 I will reflect on and conclude the paper.

## 2 Refine the problem

I work in a small company, called Robocat, where we develop Apps for iOS and Mac OS X. We have around 17 actively maintained products which are developed by 7 developers. At any given moment each developer may be involved with 2 or 3 different products. Each week the entire team and any potential clients needs to receive a weekly build of the updated products. Builds are also requested ad-hoc several times during the week. This is a process that usually takes between 10 minutes to 2 hours depending on the product. It is all done manually.

This constitutes a problem for many different reasons. We will outline the different aspects of the problem here.

### 2.1 Build Documentation & Streamlining

As mentioned earlier the company has 17 active apps currently in production. All these 17 products are actively maintained and require several updates every year just for maintenance. The most problematic issue is that every product has a separate build process. All products are build with the same tools provided by Apple but the individual build steps for each app differ in subtle ways. They have wildly different environment variables, runtime setups, libraries, etc. Some apps depend on many 3rd party libraries that needs to be updated with each new build. This means there is absolutely no streamlining of the build process. Each app is build separately in different ways.

To make matters worse there is seldomly any documentation available in the projects for how to build and deploy the apps. Important knowledge pertaining to individual build steps for products often resides with the lead developer of a product. If the developer is sick or on

vacation the build may take longer to produce or can't be produced at all.

## 2.2 Deployments break the workflow

Builds can take a lot of time to produce. Some apps only take 10 minutes to build and deploy while others can take several hours. Some apps are build once a month while others are build and deployed daily. Every time an app needs a new build it requires that a developer steps away from what they is currently working on to produce a build. These interruptions are annoying for the developer and degrades the productivity of the team. Even more so when frequent builds are requested by a customer.

## 2.3 The consequences of faulty builds

Due to the way the App Store on iOS works, bad deployments have much greater consequences than other deployments like web services. Once a build is ready for production it needs to be submitted to Apple for review. This process takes an average of 6-9 days but sometimes take as long as 3 weeks. Once Apple has reviewed the app it is either released into production or rejected (in which case a new build must be submitted). After an app is released into production it cannot be rolled back. If an app with severe defects is rolled into production they only thing that can be done is submit a new build and wait through app review once again.

This aspect severely amplifies the problem that there is no streamlined and/or documented process around building and deploying apps into production. It also contributes to a great deal of developer anxiety around deploying apps into production.

### 3 Literature Review

In *Continuous Delivery: Huge Benefits, but Challenges Too* [1], Chen covers the implementation of Continuous Delivery (CD) techniques at a large organization, the huge benefits and challenges involved. By implementing CD [1] experienced accelerated time to market with more frequent releases, faster and more usable user feedback, improved productivity and efficiency among developers, more reliable releases and decreased anxiety around production releases.

Osterweil [4] describes in his article *Processes are Software too* how humans have an innate facility for indirect problem solving through process specification. He suggest that we not only document our processes through process descriptions but also program them and use them directly as software.

Much in the way that Osterweil describes it, CD as a field has turned to scripting languages, which specify our build and deployment processes to automate these processes.

Humble, Read and North describes in [2] how to fully automate the testing and deployment process by using a multi-stage automated workflow. They note many of the same issues that are experienced by Robocat like lack of documentation around deployments.

"[...] often the documentation is incomplete or out-of-date. In some cases, the information needed to deploy resides in the heads of several key members of staff who need to come together to perform the deployment." [2]

[2] enumerates 4 key principles and some practices they motivate in order to adress the most common challenges facing automation of the

build and deployment process.



## 4 Proposed Solution: Continuous Delivery

Motivated by Osterweil [4] I have identified a solution that through process descriptions used as software aims to automate the build and deployment process.

Continuous Delivery (CD) is a software engineering approach in which a software development team keeps producing software in short cycles and ensure that the software can be reliably released at any time.

### 4.1 Automating Builds

This first part of CD is the process of automating the build phase of a software product. All parts of the process must be documented through the process of automation [4] until a binary build can be produced by a single click of a button. The implementation of this automation empowers all users of a team to generate new builds on the fly without any single team member with key knowledge to be present.

It does however take some effort to automate the build process. The cost of implementing the automation up front should be insignificant relative to the gain of faster builds. With the build automation in place, the development team will spend less time on making builds thus being more productive. This is also one of the noted benefits by [1].

### 4.2 Automating Deployments

The second and most important part of implementing CD is to automate the process of delivering software. The steps involved in this part of the

process varies somewhat depending on the software type and obviously how it is deployed to customer.

The practice is commonly used in web architectures because changes can be instantly deployed to users within minutes or even seconds. In app development for Apple's platform this is not the case.

### 4.3 Continuous Integration

- Automating build steps to produce binaries - Automating deployments to deliver software - Running integration testing (Continuous Integration)  
A three-fold solution.

First part is to streamline and document build processes. Second part is to automate build processes and deployments. Third part is to auto-trigger weekly and daily builds.

At least build automation and documentation. In more general use Continuous Delivery techniques. Automate builds entirely. Let them be invocable manually and automate weekly builds. Possibly use continuous integration techniques to verify the validity of builds before deploying to the app store. Less developer anxiety and more confidence.

Quantitative measuring development hours lost/gained. Qualitative interviews with developers and team leaders.

### 4.4 Using fastlane.tools

fastlane.tools is an Open Source suite of development tools to automate builds and deployments for iOS. [3]

## 5 Evaluation

There are several things that can be measured. Number of bad builds deployed into production. Number of development hours used for manual deployments vs. number of development hours used for setting up automated builds + invoking builds. In the qualitative department there is developer satisfaction in terms of fewer disruptions in dev workflow.

## **6 Conclusion**

# References

- [1] Chen, L. Continuous delivery: Huge benefits, but challenges too. *Software, IEEE* 32, 2 (Mar 2015), 50–54.
- [2] Humble, J., Read, C., and North, D. The deployment production line. In *Agile Conference, 2006* (July 2006), 6 pp.–118.
- [3] Krause, F. Automating your ios release process using fastlane, March 2015.
- [4] Osterweil, L. Software processes are software too, revisited: An invited talk on the most influential paper of icse 9. In *Software Engineering, 1997., Proceedings of the 1997 International Conference on* (May 1997), 540–548.