Mobile CI/CD

Lukas Baronyai, 01326526, lukas.baronyai@tuwien.ac.at

Abstract—How does an ideal CI/CD pipeline in the mobile (Android) world look like, what are existing best practises, what are possible organisationalen impacts

Index Terms—Mobile Continuous Integration, Mobile Continuous Delivery, Android.

1 Introduction

Here comes the introduction

2 Classical CI/CD

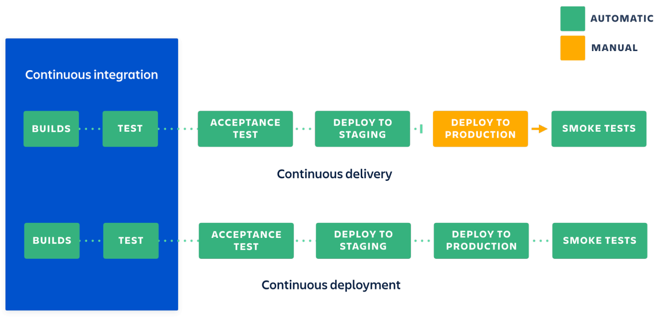

Fig. 1. CI & CD[1]

In order to look into mobile approaches to CI/CD, it is first necessary to have a quick look into the ”classical” theory. The original concept was heavily shaped by Kent Beck and Ron Jeffries in the context of their concept of Extreme Programming [1], [2]

2.1 Continuous Integration

In his text about Continuous Integration Martin Fowler defined it as follows:

> *”A software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integration per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.” [3]*

According to this, the typical workflow in a CI setup looks like the following. During the whole process there is always a high emphasis on monitoring the code state, checking (and preventing) broken code as often as possible.

1. https://wac-cdn.atlassian.com/dam/jcr:b2a6d1a7-1a60-4c77-aa30-f3eb675d6ad6/ci%20cd%20asset%20updates%20.007.png?cdnVersion=508

This process replaced the formerly used first-implement-then-integrate approach by integrating the code on a regular (daily) base.

- Checkout master branch of repository
- Create working copy
- Add changes
- Check local build
- Verify integrity with local tests
- Downmerge with master branch
- Upmerge with master branch
- Automated build at server
- Automated testing at server

Additional to this, he gave in the same text 9 practices to use CI. This list will be used in the following as a tool to analyze existing practices and especially to apply it for mobile CI/CD.

1) Maintain a Single Source Repository
2) Automate the Build
3) Make Your Build Self-Testing[2]
4) Everyone Commits Every Day
5) Every Commit Should Build the Mainline on an Integration Machine
6) Keep the Build Fast
7) Test in a Clone of the Production Environment
8) Make it Easy for Anyone to Get the Latest Executable
9) Everyone can see what’s happening

2.2 Continuous Delivery

Coming from CI, Continuous Delivery builds upon this and adds additionally a manual deployment step to the production environment. So after every test runs through positively, the setup is ready to deploy by a click on a button at any time. But it still has a final human instance in between the push to the repository and the release.

2.3 Continuous Deployment

Removing the last human component in this process then leads to Continuous Deployment, making the full process fully automated. Each push the the master branch then automatically leads to a release deployment if the whole pipeline does not detect any failed tests or broken builds.

2. https://martinfowler.com/bliki/SelfTestingCode.html

Disregarding if the last step is now fully automated or only partially with a final human component, I extend the mentioned checklist with:

10) Automate Deployment

3 Mobile CI/CD

Coming now from the "classical" CI/CD approach, it is not possible to instantly apply the known model to the mobile world, it needs some adoption. Before we do so, we have to consider the following points:

- **Mobile application have a high UI focus**
  UI tests are always more expensive to create than for "standard" code - and this is even more valid for mobile applications. Additionally from the known challenges from desktop applications like clickable fields, listener states, dependencies between views & co, a mobile device introduces complexity

- **Mobile applications have per default a build tool**
  Both iOS (XCode) and Android (AndroidStudio & gradle) come along with their own build system and IDE, making the question if the application is build manually or automated with a tool pointless.

- **There is no standard production environment**
  Due to the nature of the mobile device world, there is no standard device towards a deployment can happen. Especially but not only with Android there is a huge variety of screen sizes and operating system versions. Therefore automated testing has to be a compromise between covered variations and invested efforts.

- **Emulation is expensive**
  The consequence of the previous point is that the testing necessarily is connected with emulating the OS for each test - due to the high share of required UI tests, a majority of testing can not be done independent from the mobile OS. But emulating is expensive in terms of time and resources.

3.1 Adopted CI/CD model

Here comes the theory

3.2 Comparison

Here comes the comparison

3.3 Google Best Practices & tools

4 Mobile CI prototype

Due to the big differences in tools and languages, this paper will only focus on Android CI/CDs. Most of the principles can be used for iOS as well, but the tools will most likely differ

4.1 About Test Driven Developement in the mobile world

5 Mobile CI/CD example: AllAboutApps pipeline

How does the CI pipeline look like at AAA

5.1 Current state

Let's do the checklist:

1) **Maintain a Single Source Repository**
   ☑ Common practise in the company
2) **Automate the Build**
   ☑ Inherited due to the Gradle build system for Android
3) **Make Your Build Self-Testing**
   ☒ No automated testing
   ☒ No testing culture
   ☐ Bitrise would support it
4) **Everyone Commits Every Day**
   ☑ Common practise in the company
5) **Every Commit Should Build the Mainline on an Integration Machine**
   ☑ Covered by Bitrise
6) **Keep the Build Fast**
   ☑ Inherited due to the Gradle build system for Android
7) **Test in a Clone of the Production Environment**
   ☒ No automated testing
   ☒ production environment is hard to define
8) **Make it Easy for Anyone to Get the Latest Executable**
   ☑ Covered by Bitrise artifacts
9) **Everyone can see what's happening**
   ☑ Covered by Bitrise
10) **Automate Deployment**
   ☐ newly introduced practise to deploy to Google Play Alpha/Beta channel, but not in every project yet

5.2 Missing steps

We don't test :/

5.3 Change impact

Hui, nobody knows!

6 Conclusion

The conclusion goes here.

References

[1] K. Beck, "Extreme programming: A humanistic discipline of software development," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 1998, pp. 1–6.
[2] K. Beck and E. Gamma, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.
[3] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, vol. 122, p. 14, 2006.
[4] M. Virga and L. Clark. (2019, Aug.) Continuous integration and continuous delivery: Beyond the conveyor belt mentality. [Online]. Available: https://edge.siriuscom.com/digital/continuous-integration-and-continuous-delivery-beyond-the-conveyor-belt-mentality

2. https://www.bitrise.io/