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 - What's the motivation, what are the reasons of this

2 Classical CI/CD

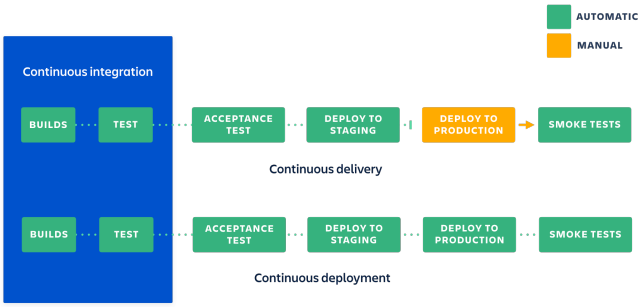

Fig. 1. CI & CD[1]

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. 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.

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

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 of 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 to be handled. Therefore automated testing has to be a compromise between covered variations and invested efforts. There are of course attempts to test with assuming the maximum coverage of variations, but it can never be sufficient as for e.g. a server software.

- **Emulation is expensive**
  Another consequence of the variety of device types, testing with real hardware is a pain and lead to the usage of emulators. But emulation is expensive in terms of time and resources, especially since due to the high share of required UI tests, a majority of testing can not be done independent from the (emulated) mobile OS.

3.1 Adopted CI/CD model

As consequence to the mentioned points, we will adopt the practise checklist of Fowler by completely removing 2) (inherited to the build tools) and changing 7) to "Test in a Emulated Environment".

3.2 Googles CI/CD tools

Google provides many frameworks and guidelines which can be pretty overwhelming - this and the next section is therefore dedicated to give an overview of these and summarize which options there are to implement a proper CI/CD pipeline within the Android world.

3.2.1 Android Emulator

Bundled within the Android SDK comes the Android Emulator[3], which allows the emulation of a virtual device on the local machine. It supports various images, called Android Virtual Devices (AVD), with different configuration options. Once launched it is considered as a real device by `adb` and can therefore used as deployment target. The tool further provides the option to launch without an UI and to be used for testing on a server.

3.2.2 Firebase Test Lab

With Firebase Test Lab[4] Google offers a cloud-based device farm with virtual and real devices. It supports two different testing methods: Instrumentation (see section 3.3.1) and Robo (basic test type that simulates real user interactions) Tests. Test results are provided with additional logs, videos and screenshots. Firebase Test Lab allows 10 tests per day for virtual devices and 5 for physical devices in it's free version ("Spark Plan") - but some CI/CD cloud services like Bitrise included the service within their contingents.

3.2.3 Google Play Deploy

Google offers via the Play Store console a REST API to deploy compiled APKs and AABs which can be used in an CD setup. The API is secured via the according Service account. Additionally to the (not prefered way) direct release track it is possible to deploy to the `alpha`, `beta` or `internal` tracks[5].

3.2.4 Available CI/CD Server

In order to properly utilize a mobile CI/CD pipeline there are many estabilshed server provider around - since it would exceed the scope of this paper to make a full comparison of these service providers, we limit it to a selection of the most commons:

- Bitrise[6]
- TeamCity[7]
- Travis CI[8]
- Jenkins[9]
- Bamboo[10]
- GitLab CI/CD[11]
- CircleCI[12]

3.3 Android testing

- **Small tests**
  70% unit tests, JUnit, Mockito, PowerMock
- **Medium tests**
  20% Integration Tests, Robolectric
- **Large tests**
  10% UI Test, Espresso, Robotium, UI Automator

3. https://developer.android.com/studio/run/emulator
4. https://firebase.google.com/docs/test-lab/android/overview
5. https://developers.google.com/android-publisher/tracks
6. https://www.bitrise.io/
7. https://www.jetbrains.com/teamcity/
8. https://travis-ci.org/
9. https://jenkins.io/
10. https://www.atlassian.com/software/bamboo
11. https://about.gitlab.com/product/continuous-integration/
12. https://circleci.com/
13. https://testing.googleblog.com/2010/12/test-sizes.html

Feature	Small	Medium	Large
Network access	No	localhost only	Yes
Database	No	Yes	Yes
File system access	No	Yes	Yes
Use external systems	No	Discouraged	Yes
Multiple threads	No	Yes	Yes
Sleep statements	No	Yes	Yes
System properties	No	Yes	Yes
Time limit (seconds)	60	300	900+

Fig. 2. Test Sizes[13]

3.3.1 *Local tests VS device tests*

`test` for unit tests, run on local machine/JVM
`androidTest` for instrumented tests, integration tests, UI tests

3.3.2 *AndroidJUnit4*

Android JUnit Runner
Android Test Orchestrator

3.3.3 *UI Automator*

3.3.4 *Espresso*

3.3.5 *Robolectric*

3.3.6 *Android Jetpack Test*

Note: Google recently (I/O18) announced "Project Nitrogen" which intents to remove any restrictions of tests to run time environment - but there are no more information available at time of writing this paper.

4 Mobile CI prototype

Note: 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 Development in the mobile world

5 Mobile CI/CD example: AllAboutApps pipeline

How does the CI pipeline look like at AAA

15. https://developer.android.com/images/training/testing/testing-workflow.png
17. https://developer.android.com/images/training/testing/pyramid.png

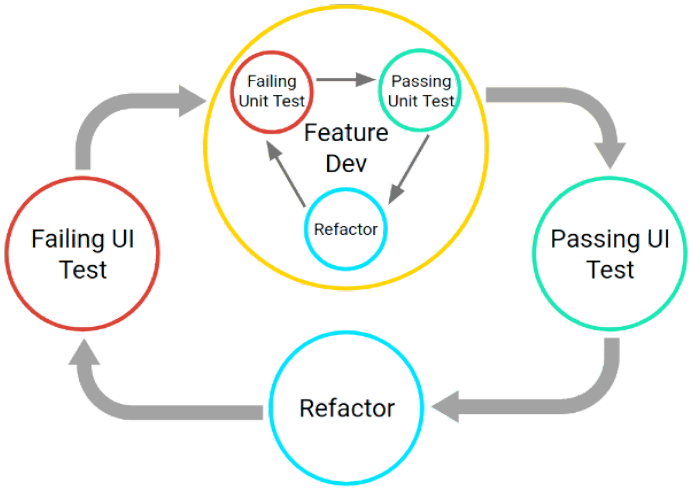

Fig. 3. The two cycles associated with iterative, test-driven development[15]

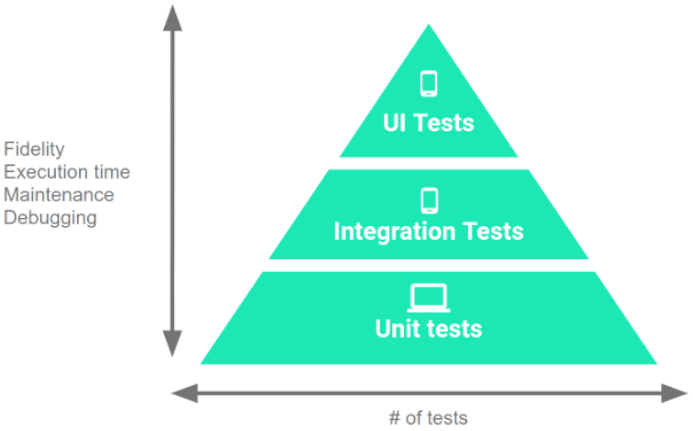

Fig. 4. The Testing Pyramid, showing the three categories of tests that should be included in an app's test suite[17]

5.1 Current state

Let's do the checklist:

1) **Maintain a Single Source Repository**
   ☑ Common practise in the company

2) **Make Your Build Self-Testing**
   ☒ No automated testing
   ☒ No testing culture
   ☐ Bitrise would support it

3) **Everyone Commits Every Day**
   ☑ Common practise in the company

4) **Every Commit Should Build the Mainline on an Integration Machine**
   ☑ Covered by Bitrise

5) **Keep the Build Fast**
   ☑ Inherited due to the Gradle build system for Android

6) **Test in an Emulated Environment**
   - ☒ No automated testing
   - ☒ production environment is hard to define

7) **Make it Easy for Anyone to Get the Latest Executable**
   - ☑ Covered by Bitrise artifacts

8) **Everyone can see what's happening**
   - ☑ Covered by Bitrise

9) **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