Draft / Concept for a Pipeline that covers CI/CD of an iOS Mobile App Project

by Will Stefan

Introduction

For one thing, the meaning of the word “draft” as an idea and not a final concept has to be kept in mind here. Because the draft is more like a starting point due to the fact that that a good working testing process is iterative. That means it needs continuous improvement itself since every project is unique. Therefore the individual metrics differ. For instance how many requests are most likely sent to the app? Does every OS/device support it’s (core-)features and so on. Last but not least personal experience and research has shown that there is not such thing as a golden approach which underlines the statements in this paragraph.

Nevertheless a fundamental concept gets described in the following chapters that I am happy to discuss with the team.

This documentation also assumes that the development-/project management considers SCRUM.

Task

The task is to set Up a Draft / Concept for a Pipeline that covers CI/CD of an iOS Mobile App Project that includes a big UI Testing suit in a Nightly Regression. In this draft there will be also an explanation of the tool to use for this.

Theories to consider

This chapter gives a small overview of what theories, terms and their meaning need to be considered for this concept.

Continuous integration (CI)

According to Wikipedia, CI is the approach of merging every developer working copy to a shared one. This will be done many times a day.

It covers mainly the following questions to consider:

- Was it build right by a single developer?
- Did something break?

Continuous delivery (CD)

Looking it up on Wikipedia, CD deals with the feedback and covers delivery value to the user.

It considers mainly the following questions:

- Was the app build right by every developer?
- Did the changes break anything?

CI/CD for Mobile

In contrast to CI/CD for webapplications and cloud-solutions, a mobile app has to be shipped to a App Store.

Therefore the App needs to be delivered theoretically at all time even though not every build gets shipped.

To do so the following needs to be considered:

- Different OS versions
- App Store approval cycle which is time consuming

Concept

Key objectives and metrics

The following table shows the items/aspecst that testing needs to cover.

Item	Acceptance criteria
Coverage of test cases	> 80% coverage
Speed of the whole test	< 15 min
Reliability of test cases	> 90%
Bug limit	< 20 open bugs
Visual dashboard to have an overview about those metrics	Needs to be implemented/exist in the build pipeline of the project
GIT	Used by everyone as version control system

Implementaion of CI

One main branch in Git from which the app gets shipped will be the “master”-branch. Therefore this is the branch where the app works as expected.

The code review will be done in the “develop”-branch in which all the developers and testers merge their code. After the review states that everything works fine, the “develop”-branch will be merged into the “master”-branch.

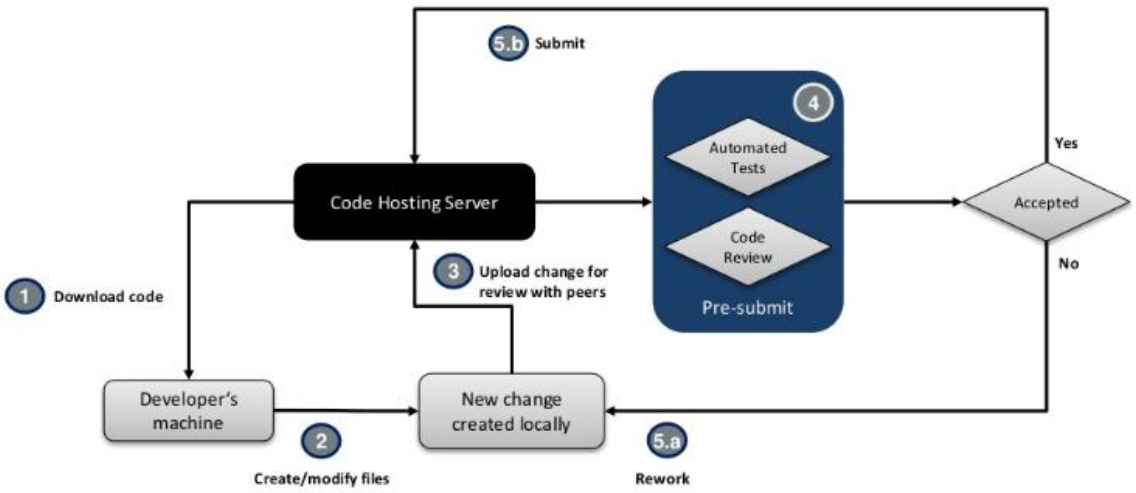

Image 1: Slide 14 on https://de.slideshare.net/stefanverhoeff/cicd-for-mobile-at-here

Pipeline-Setup

This pipeline-setup description contains the basic principals and not a guidance of how to do it with a special tool since this was not demanded.

Step 1 – Unit Test covering

The first step of the pipeline covers all the Unit-Tests before building the actual app since Unit-Tests are isolated and don't consider the environment but just the code itself.

Step 2 – Build the app

In the second, which only gets triggered if step 1 succeeded, the app will be built for further testing steps.

Step 3 – Automated tests

The automated tests after a successful software-build consider:

- **Component and Integration Tests:** After each and every component got tested by Unit Tests, now their interaction with each other gets tested according to a test plan individually designed according to the projects/software demands or specification.
- **API-Tests:** In this testing area all the REST-calls and interaction-calls with embedded components that the software talks to get tested. This may interfere with component and Integration Tests since components might talk to each other via the API.
- **GUI-Tests:** Just the necessary ones (consider the martin fowlers testing pyramide) since they are time consuming (high costs in the testing pipeline)

Step 4 – Ship the app

According to the release plan/schedule the shipping release will be triggered from the "master"-branch if the release is stable. Then the app will be pushed to the App Store after every sprint ideally.

It has to be mentioned, that the release plan/schedule contains the period of times in regards to shipping that needs to be discussed between the testing manager and the product owner.

Developers and testers also need to be included into the discussion since they fulfill the requirements.

Platform architecture and tooling

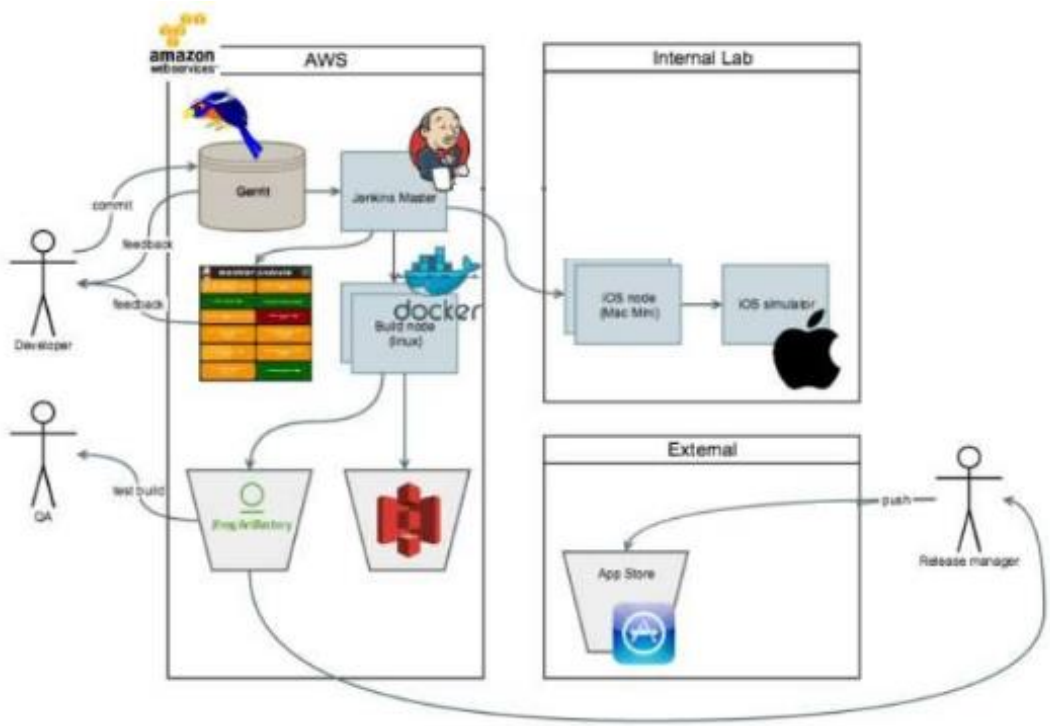

Image 2: Modified slide 17 on https://de.slideshare.net/stefanverhoeff/cicd-for-mobile-at-here

Testing

The testing process consists following steps:

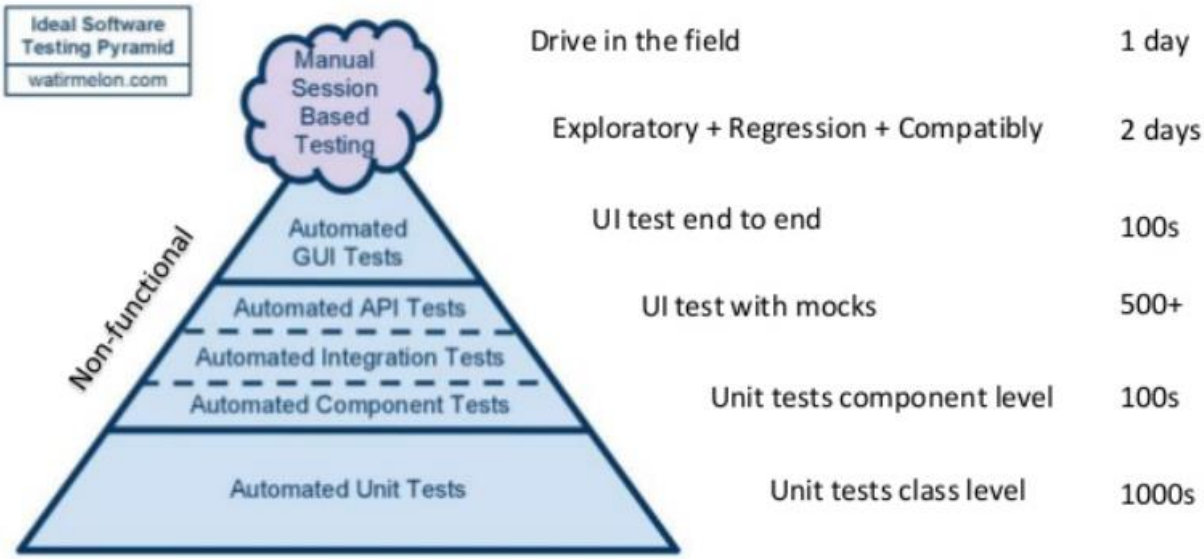

Image 3: : Modified slide 19 on https://de.slideshare.net/stefanverhoeff/cicd-for-mobile-at-here

The modified testing pyramide (in refers to martin fowler) from this source states it clearly and proofs the amount of coverage for each and every testing discipline.

Data Handling

Data Handling while testing is a controversy issue that also depends on the software architecture. Basically the issue is that mocking data is not authentic in regards to the actual data flow of the application but necessary for unit-testing. Considering data handling on a test server for instance, tests can create data which changes the initial test state. Therefore key is, while working with real data flow, that the tests that create data clean it up at the end of the test. That means that the initial testing state gets recovered in order to run the tests again with the same state or pre-conditions.

Pro Mocks

- Fast and in version control
- No need for a server, database-connection, backend- or GUI-implementation

Contra Mocks

- Not authentic in regards to the data flow/handling of the application
- Database structure doesn't get considered
- Since they are in version control, they also need to be maintained (up to date with the backend- and database-logic)

Dashboards

As integration tool Atlassian Jenkins gets used since it is for free and got a huge (support) community. Jenkins also supports various possibilities of making test results graphically visible to everyone involved into the testing process.

The dashboard needs to cover the following:

- Passed failed and skipped tests in a table overview as well as per single graph
- Performance graphs about the testing to get information if the automated API- and Components-test are slow/fast over a certain period of time of testing

Useful links

CI/CD for mobile:

https://de.slideshare.net/stefanverhoeff/cicd-for-mobile-at-here

Data-Handling:

https://www.micromata.de/blog/softwareentwicklung/data-mocking-moeglichkeiten-eines-fake-backends/

https://github.com/CarmenPopoviciu/protractor-styleguide

„Myths and Truths“ – Automated Testing

https://www.sencha.com/wp-content/uploads/2016/05/The-Evolving-Landscape-of-Web-Application-Testing-white-paper-screen.pdf