Stefan Painhapp, BSc

Continuous Integration for iOS Apps

A practical approach for Pocket Code

Master's Thesis

to achieve the university degree of
Master of Science

submitted to
Graz University of Technology

Supervisor
Univ.-Prof. Dipl-Ing. Dr.techn. Wolfgang Slany

Institute for Software Technology

Graz, May 2019

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Abstract

Continuous Integration (CI) has been increasingly gaining importance at Software Development workflows during the course of recent years. The benefits of automated processes and feedback are facilitating the agility of developers and improving productivity. In this thesis the concepts of Continuous Integration are explained in general, moreover how they can be specifically used at iOS Development.

A practical example of a Continuous Integration pipeline is shown with the introduction of CI at the development of Catrobat's Pocket Code for iOS. Additionally, the tools in use for this practical setup of CI and Continuous Delivery (CD) for iOS are explained in detail. At the end a short summary shows the findings of the practical work and an outlook of future developments of CI.

Kurzfassung

Die Bedeutung von Kontinuierlicher Integration (Continuous Integration) hat in der Softwareentwicklung in den vergangenen Jahren immer mehr zugewonnen. Die Vorteile von automatisierten Prozessen und Rückmeldungen fördern die Agilität der Entwickler und erhöhen die Produktivität. In dieser Arbeit werden die Konzepte von Kontinuierlicher Integration im Allgemeinen, sowie dessen Eigenheiten in Bezug auf iOS Entwicklung, erklärt.

Ein praktisches Beispiel eines Kontinuierlichen Integrations-Ablaufs wird anhand der Einführung des Prozesses bei Catrobat's Pocket Code für iOS vorgestellt. Weiters werden die Werkzeuge für die Verwendung von Kontinuierlicher Integration und Kontinuierlicher Bereitstellung (Continuous Delivery) für iOS genauer erläutert. Schlussendlich werden die Ergebnisse der praktischen Arbeit zusammengefasst und ein Ausblick für Kontinuierliche Integration und Kontinuierliche Bereitstellung präsentiert.

Contents

List of Figures

Listings

1. Introduction

Continuous Integration (CI) is a practice at Software Development that resulted from the drawbacks of developers working on a set of features for a long time span. After the completion of changes, merging the code was a cumbersome task. The phrase "Continuous Integration" has been used in literature before, but became popular with the publication of *Extreme Programming* (XP) by Kent Beck's demand:

> *"Continuous integration — Integrate and build the system many times a day, every time a task is completed"* , Beck (1999)

Many XP teams started to adopt to this behaviour. In 2000, Fowler and Foemmel published their findings about using a basic CI setup. Wider adoption came with the software CruiseControl[1] in 2001, which was one of the first tools that provided a set of features to tackle the challenges of CI.

Since then, CI development has come a long way and several different software solutions are offered. However, the principles, challenges and benefits stated in original publications, further discussed in chapter 3, are still valid. Enhancement to CI, like Continuous Delivery (CD) are additionally automating reoccurring tasks of the Software Development cycle.

Over the course of this work, CI will be discussed with the special focus on iOS development (2). The setup is including several challenges, not only the ones that come with developing smartphone apps itself, but also with a very restrictive runtime and development environment of Apple, that is increasing the difficulty of CI introduction for iOS App Development.
A practical implementation of a CI system is shown with the help of Catrobat's Pocket Code.

[1]`http://cruisecontrol.sourceforge.net/`, visited on 05/15/2019

1.1. Catrobat

Catrobat was founded by Wolfgang Slany at Graz University of Technology. It started with the vision of bringing Visual Programming in the education sector to the emerging market of smartphones.
Inspired by the existing work of Scratch[2] from MIT an Android app "Pocket Code" was created (team name: *Catroid*). It uses the same concept of graphical scripting bricks to develop apps and games directly on phones.

This approach came not only with new difficulties - compared to the existing web- and computer-based products - by using smaller screen sizes and touchscreen input, but also with new possibilities to enable the creativity of the users. The capabilities of using cameras, accelerometer, location data and many other functionalities of smartphones are implemented within Pocket Code to be utilised when developing projects.

All products developed by Catrobat are FOSS projects dependant on the work of volunteers. Numerous developers, translators and other project members are participating to constantly improve the functionalities by realizing new ideas.

Several teams are working on different projects. Besides Catroid (Android), there are Catty (iOS - presented in the following chapter), Jenkins (CI environment), Catroweb (website and sharing), Paintroid (drawing app for Android) and special teams for Design, Drones, Music, Lego Robots, Phiro Robots, as well as working groups to research new technologies (example: Catblocks - Google Blockly[3]).

Some projects were discontinued, such as Catrobat's HTML5 player or the Windows Phone app. Especially after Microsoft announced that they are stopping the development of Windows Phone, the smartphone market became a duopoly and providing an iOS version of Pocket Code became crucial for supporting most of the devices on the market.

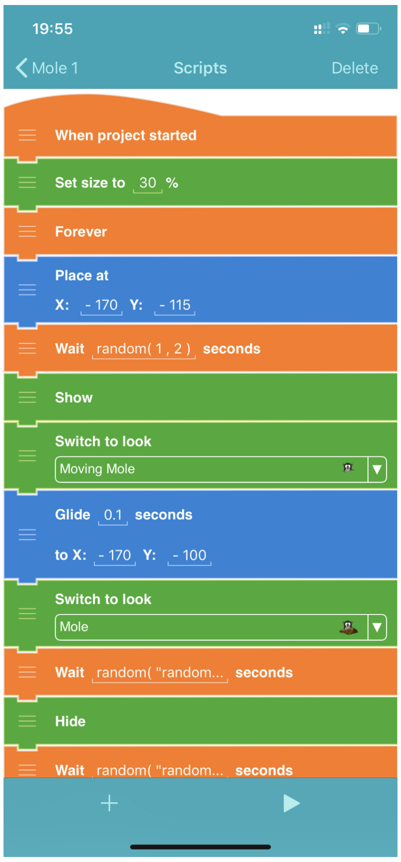
[2]`https://scratch.mit.edu`, visited on 10/05/2019
[3]`https://developers.google.com/blockly/`, visited on 05/15/2019

Figure 1.1.: Pocket Code for iOS app icon.

1.2. Catty - Pocket Code for iOS

The idea for an iOS implementation of Pocket Code was already established in 2012. Under the team name "Catty" (reference to "Pocket Code for iOS"), a prototype was realized to evaluate the potential. The project did not develop as rapidly as the Android counterpart, due to the lack of iOS developers at Graz University of Technology and other factors. At the beginning of 2018 a feature-stop was coordinated, and the decision was made to publish the app with a lower Catrobat language version (0.991) than currently released in other projects. The language version of Catrobat defines the standard, which includes featured bricks, their functionality and how projects are stored in XML files and the corresponding file structure. After eradicating misbehaviour and preparing the app for release, Pocket Code went live on the Apple App Store on 18/01/2019. The app icon is shown in figure 1.1.

Catty is a fully native implementation using iOS frameworks (for more information about iOS Development, see chapter 2), sharing the same backend with other Catrobat projects. As a result, Catrobat projects can be shared across platforms and different operating systems.



```
When project started
Set size to 30 %
Forever
Place at
X: - 170 Y: - 115
Wait random( 1 , 2 ) seconds
Show
Switch to look
Moving Mole
Glide 0.1 seconds
to X: - 170 Y: - 100
Switch to look
Mole
Wait random( "random... seconds
Hide
Wait random( "random... seconds
```

Figure 1.2.: Visual Programming with bricks in Pocket Code.

One of the challenges at implementing Catty was to provide the same user experience as on other Catrobat platforms, while also following Apple's set of rules for iOS apps. All iOS apps distributed on the Apple App Store have to follow the Human Interface Guidelines[4], which are checked by Apple during a review process (see chapter 7.4).

Figure 1.2 illustrates the visual editor for developing projects within Catty. iOS UI patterns are used for an intuitive user experience and mitigating rejections during App Review. On the other hand it is showing a familiar UI for users, who are already using other Catrobat products.

[4]https://developer.apple.com/design/human-interface-guidelines/ios/, visited on 05/15/2019

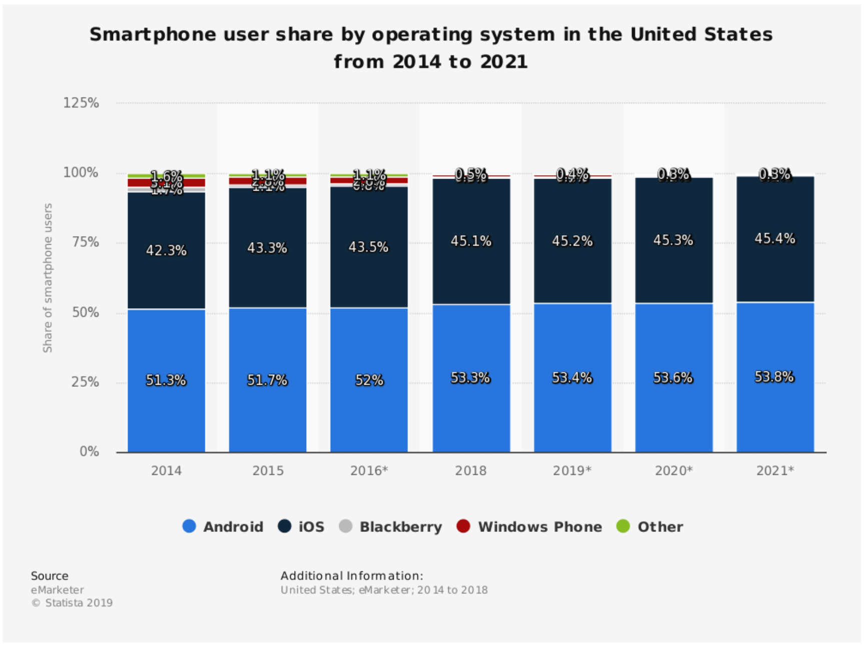


Figure 1.3.: Smartphone user share by operating system in the United States from 2014 to 2021, from Statista, eMarketer, 2019.

1.2.1. iOS market for Pocket Code

Catrobat's philosophy is pursuing the goal to teach Programming in a playful way to any interested person, whereat kids and teenagers are especially targeted. Pocket Code for Android is also popular in developing countries, as the smartphone market is notably surpassing computers. In terms of iOS importance, the market is globally smaller in contrast to Android.

In some countries - like the US market (shown in figure 1.3), but also in specific countries in Asia (South Korea, Japan) - iOS has a substantial market share. With the release of Pocket Code for iOS, these phone owners can also be served, thus helping the vision of Catrobat by offering cross-device sharing of created content.

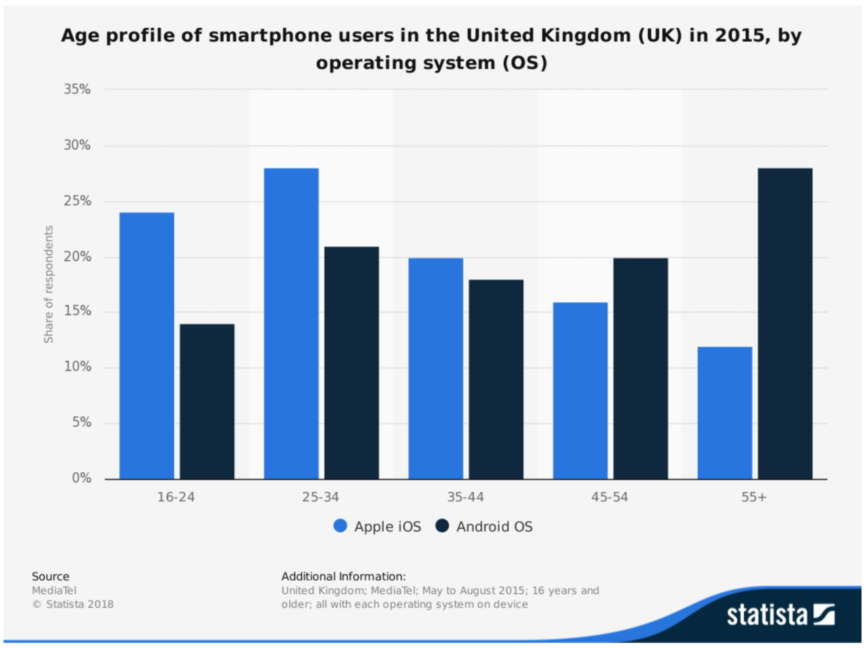


Figure 1.4.: Age profile of smartphone users in the United Kingdom (UK) in 2015, by operating system (OS), from Statista, MediaTel, 2015.

In figure 1.4 the target group of teenagers in the United Kingdom, 2015 is specified by operating system. There is a noticeable trend of younger age groups using iOS, whereas Android is more popular at older age groups.

Part I
Theoretical

2. iOS Development

In this chapter, iOS development characteristics related to Pocket Code are discussed. Catty is using a variety of iOS frameworks, such as:

SpriteKit Framework for creating 2D games, supporting physics effects. Core functionality for graphical interpretation, when running Catty projects

CoreMotion Framework for accessing accelerometer, gyroscope and pedometer data. The data can be used when executing projects at runtime.

AVFoundation Framework for audio playback and camera support, making Media content available within Catty projects.

These frameworks will not be covered in detail. Instead, the respective information necessary for CI and CD should be pointed out.

2.1. FOSS in the iOS ecosystem

Due to the nature of Apple's philosophy of distributing a closed, proprietary system, FOSS is less common in contrast to Android. Especially in terms of Software Distribution, developers are tied to the App Store Guidelines, making it less attractive for open source projects to provide iOS versions, as their single distribution channel can be revoked by Apple at all times. Unlike Android, alternative markets for iOS Software Distribution (example: Cydia, after jailbreaking devices) are exceptions and used by a minority.

However, renowned FOSS projects like Firefox, owncloud/nextcloud, VLC and messengers (Signal, Telegram) among others are also providing iOS versions.

In 2014, Apple announced the new programming language Swift. One year later (2015), there was a paradigm shift by Apple, making Swift fully open source and utilising the community engagement. That brings us to the next section.

2.2. Programming Languages (Objective-C, Swift)

Until the introduction of Swift, Objective-C was the only officially supported language for iOS Development by Apple. As for iOS software updates from user perspective, the adoption rate at the development community has been tremendous to adjust to Swift. It is specifically owed to the low popularity of Objective-C, which has also been proved at Stack Overflow Insights, 2019. In this study, nearly 90,000 developers participated to rate their most and least favourite programming languages. As illustrated in 2.1, the development community really embraced the new software development concepts introduced by Swift.

Some of the new features are:

No Pre-Processor Objective-C is based on C. At Swift no pre-processor is needed, the source code consists of a single file (no header files).

Optionals Improved handling of non-existing values.

Protocols Acting as blueprint for requirements (methods, properties) that can be assigned to classes.

Syntax Appearance is more common to modern programming languages. Objective-C's square brackets syntax is obsolete.

Other features like Range- and Overflow Operators, Lazy Stored Properties and Closures are pointed out in the work of Garcia et al., 2015.

By May 2019, Catty's code base consists of 59.2 % Objective-C, 37.1 % Swift and 3.7 % other source code. Over the course of this thesis, several modules have been migrated to Swift during refactoring, as well as newly introduced concepts like Linting (see 6.2.2) have been realised with Swift.

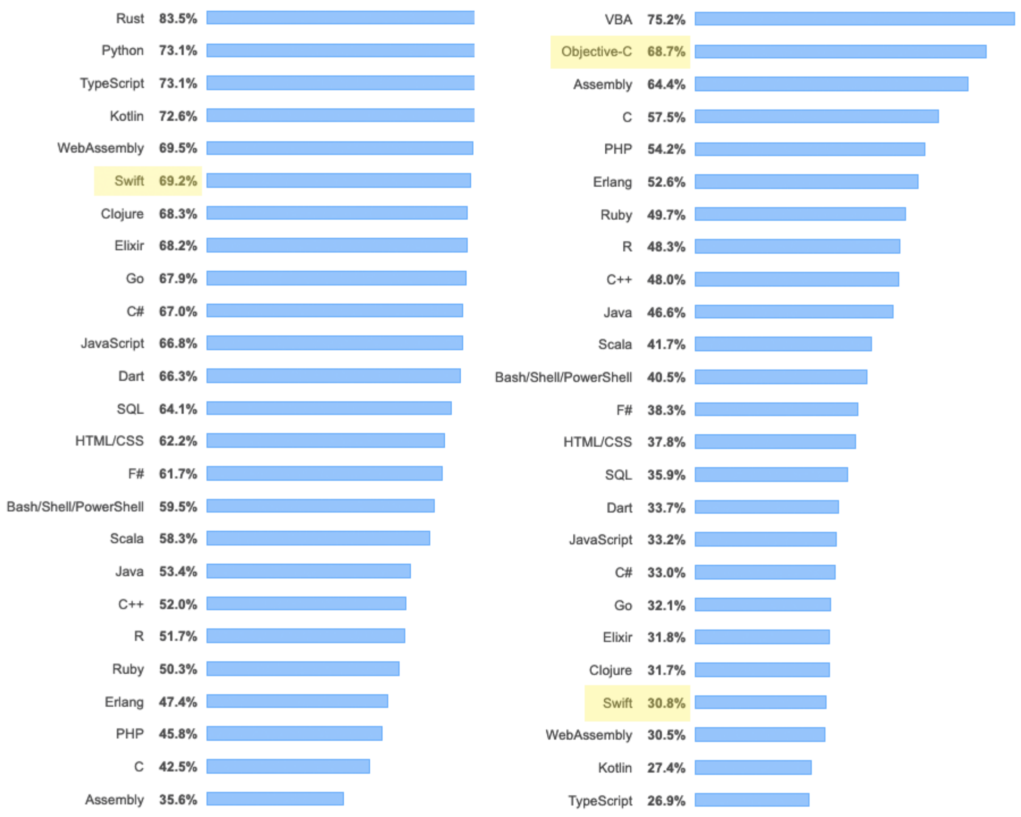
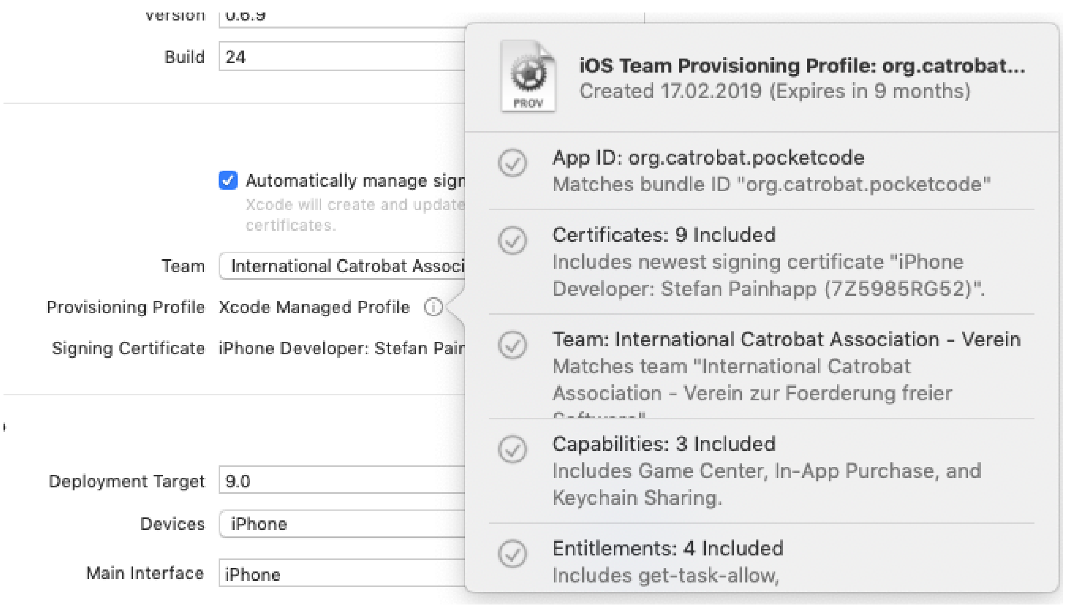


Figure 2.1.: Loved (left) and dreaded (right) programming languages in percentage, from Stack Overflow Insights, 2019

Rebouças et al. (2016) claimed in an empirical study on the usage of Swift that it can be easily adopted by developers. The majority of the questions during the interviews have not been about Swift, but more about libraries and frameworks in general. This was also observed at Catrobat; team members that were new to iOS Development showed a drastically decreased learning curve than when confronted with Objective-C code.

Swift itself is using CI at development for PR integration testing on macOS, Linux (Ubunutu) and iOS (Simulator) via Jenkins server[1].

[1]`https://ci.swift.org/`, visited on 05/15/2019



Version 0.6.9

Build 24

☑ Automatically manage sign

Xcode will create and update

certificates.

Team International Catrobat Associ

Provisioning Profile Xcode Managed Profile

Signing Certificate iPhone Developer: Stefan Pair

Deployment Target 9.0

Devices iPhone

Main Interface iPhone

iOS Team Provisioning Profile: org.catrobat...
Created 17.02.2019 (Expires in 9 months)

✓ App ID: org.catrobat.pocketcode
Matches bundle ID "org.catrobat.pocketcode"

✓ Certificates: 9 Included
Includes newest signing certificate "iPhone Developer: Stefan Painhapp (7Z5985RG52)".

✓ Team: International Catrobat Association - Verein
Matches team "International Catrobat Association - Verein zur Foerderung freier

✓ Capabilities: 3 Included
Includes Game Center, In-App Purchase, and Keychain Sharing.

✓ Entitlements: 4 Included
Includes get-task-allow,

Figure 2.2.: Provisioning Profiles at Xcode 10.

2.3. Code Signing

A requirement for running iOS apps on physical devices is code signing. To execute builds on a physical device, a signing certificate has to be requested from Apple. This is done by creating a Certificate Signing Requests (CSR) on the implementation machine and uploading the CSR to Apple. After that, the provided certificate has to be added to the local Keychain of the workstation. Meanwhile, Xcode is providing functionality to automatically handle these requests for the developer. The integration into Xcode is illustrated in figure 2.2.

Signing Certifcates

Signing certificates can be classified into two categories: Development and Distribution. Development certificates are bound to the Developer Identifier, whereas Distribution certificates are attached to the Team Identifier.

The following signing certificates are examined:

Development Used by team members for development (builds).

Ad-hoc (Distribution) Enables over-the-air distribution for a limited number of devices (100). The devices have to be registered with their UUID in the developer portal.

App Store (Distribution) Required signing method for builds that are uploaded to the App Store infrastructure.

Enterprise (Distribution) Propagation of builds outside of the App Store.

Development, Ad-hoc and App Store signing is included in the Apple Developer Membership for iOS. Enterprise certificates are exclusive for publishers that are not using the App Store. A seperate developer membership is needed and can not be combined with an existing App Store membership.

Enterprise distribution outside the App Store is uncommon. Zheng et al., 2015 analysed the state of the market and their opposing security risks, finding several malicious apps.

Provisioning Profiles

Provisioning Profiles act as container that include all necessary components of code signing, including certificates:

App Information General information about the app: Bundle-Id, Team Id and the list of Entitlements (Push Notifications, Siri, HomeKit, ...).

Certificates Intended signing certificate.

Devices A list of devices by name and UUID, which are allowed to install the signed build.

These profiles are used for signing a build. Common errors at build time are caused by corrupt, expired or incorrect Provisioning Profiles.

3. Continuous Integration

Continuous Integration was introduced due to a common problem area at software development teams: Every developer is working on an assigned set of features until completion, until finally everything is tied together in an integration phase. This can not be accomplished without difficulties in most cases. As a result, integration becomes an uncertainty in development cycles. If problems occur, the origin is not easily identifiable and it results in a lengthy search for errors and bugs.

The idea of CI is to avoid these incidents by integrating as often as possible. As manual integration can be an extensive task, it needs to be automated wherever feasible. Duvall, Matyasand, and Glover (2007) accurately summed it up:

> *"Developers spend most of their time automating processes for their users, yet don't always see ways to automate their own development processes."*

Then again, CI is more than simple scripts tied together. Stakeholders of the development process have to adhere to new rules and principles, that are showing additional benefits to the sole concept of automation.

3.1. Practices

For CI to work, some behaviours and processes have to be adopted by the developers. CI should not interfere with other software development concepts, that might already be in use, like Agile, Scrum, XP or TDD among others. The success factor of a CI system, however, is dependant on improving the *habits* of developers, especially when working in teams.

Fowler and Foemmel, 2006 defined the following key practices for CI:

Maintain a Single Source Repository Every developer involved needs to work on a single code base. All necessary parts should be included and accessible by all members – only third-party components can be referenced by dependecy management.

Automate the build Builds are automatically executed as first step of the CI pipeline. Xcode provides command line tools for building without the need of in IDE at iOS Development.

Make your Build Self-Testing Successful compilation of builds in the previous step are not guaranteeing correct functionality. All created tests should be run additionally to verify the changes.

Everyone Commits To the Mainline Every Day Daily commits act as a rule of thumb. Later literature (Duvall, Matyasand, and Glover, 2007) even suggests frequent commits, many times a day – the more often, the better.

Every Commit Should Build the Mainline on an Integration Machine CI jobs at every commit ensure finding possible errors in a timely manner. When introducing CI, nightly builds are a good start.

Keep the Build Fast Crucial for the productivity of the developers. Fast responses mean, that they can focus on other tasks quickly, instead of waiting for the CI status.

Test in a Clone of the Production Environment Tests should match the live environment. iOS testing on physical devices might be more beneficial than using iOS simulators in some cases.

Make it Easy for Anyone to Get the Latest Executable Getting builds for specific changes is especially important for helping other team members during problems or at code review.

Everyone can see what is happening Information about all CI runs should be shared with the rest of the team. Modern CI tools are offering valuable information to all members.

Automate Deployment This principle of Fowler and Foemmel resulted in further deployment improvements, Continuous Delivery (see: 3.5).

3.2. Requirements

CI itself does not require particular tools, although the following parts are found in most CI setups:

Version Control System A central access point for developers. Example: a source control system like Git.

Automation Tools A collection of tools to automate the processes, necessary for CI (build, test, analyse). It is usually handled by the CI server and its features.

Software Development Some preconditions at software development processes are necessary, such as:
a) Tests should already be present, if CI is introduced in existing development teams. More precisely, it is crucial that the tests are meaningful for providing valuable feedback of CI. Pouclet, 2014 stated that the tests should help the product and should not only be created for the sake of testing.
b) Quality measures should be defined in the form of coding standards, code coverage analysis and other actions.

3.3. Feedback

One of the main components of a CI system is the ability to provide more detailed information about the development of software products. CI feedback can help various stakeholders: *Managers* are able to have better insights on the current state of developed features. They can quickly react to bottlenecks at occurring issues by adapting workforce and priorities. *Developers* are able to react faster to broken builds. The corresponding changes of a CI job are included in the reports, which makes it easier to find bugs.

Most feedback systems of CI servers are providing additional functionalities. For example: Emails, notifications or alerts can immediately report to specific groups of interest, if certain cases occur.

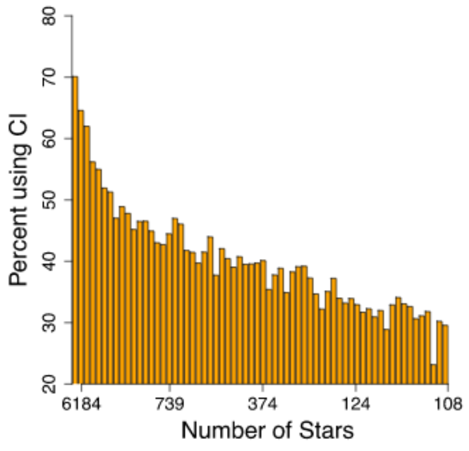


Figure 3.1.: CI usage of projects in GitHub. Projects are sorted by popularity (number of stars), taken from Hilton et al., 2016

3.4. Benefits

During the introduction of CI, multiple problems have to be tackled, besides the technical challenges. The processes have to be adopted from everyone within the team, barriers have to be removed and previous habits changed. After successfully transitioning to this organizational behaviour, several benefits can be observed.

Hilton et al., 2016 analysed open source projects that are using CI and found out that FOSS projects with CI release more often, not only than before using CI, but also in comparison to other similar projects. In figure 3.1 the correlation of popularity (number of stars of a repository) and CI usage is pointed out.

The main advantage is a reduced risk of software errors, because of the early detection of bad code. Thanks to automation, developer spend less time on integration and in general less time on all processes, that have been automated by CI tools. Response times to broken code, builds and tests are shortened by immediate feedback to the developers. Publishing releases is involving less stress of the team, which results in more frequent updates. In more advanced setups of CI, this step is automated as well.

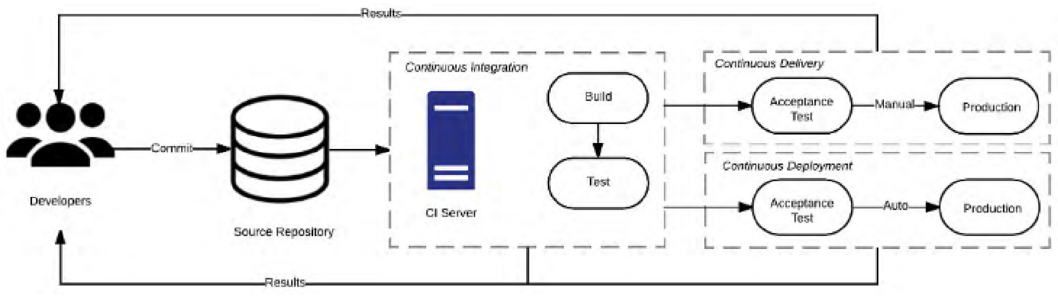


Figure 3.2.: The relationship between Continuous Integration, Delivery and Deployment, taken from Shahin, Babar, and Zhu, 2017.

3.5. Continuous Delivery

Extending the concepts and philosophies of CI to the area of Software Publishing, results in the topics of "Continuous Delivery" (CD) and "Continuous Deployment". Shahin, Babar, and Zhu, 2017 explain that there are debates about the exact definitions. As seen in figure 3.2, a commonly used distinction is, that Continuous Delivery is using a manual approach for publishing, whereas Continuous Deployment is automatically pushing all successful changes automatically to the production environment. This process is difficult to achieve at iOS Development, as iOS Apps have to pass an App Review (see, 7.4). Rossi et al., 2016 describe in their paper how Continuous Deployment at Facebook's iOS app is handled.

After a successful CI job, CD is dealing with the general problems of delivering software; this includes Configuration- and Release-Management. Besides software testing, acceptance tests ensure that non-functional requirements are fulfilled. Humble and Farley, 2010 describe the challenges of Automated Acceptance Tests and how to handle them.

In regards to iOS development, CD has to additionally handle App-Store specific tasks (Distribution signing, Metadata, Screenshots) during the deployment.

Part II
Practical

4. Motivation

Continuous Integration has already been successfully introduced at other Catrobat teams (Catroid, Paintroid). Findings and benefits for Catroid's CI were published by Luhana, Schindler, and Slany, 2018.

To take advantage of the automated workflows and improvements at Catty, CI should be adopted at iOS development of Pocket Code as well. With an increasing number of contributors and a matured state of Catty, CI is becoming a necessary part for substantial product growth. Where possible, the running infrastructure should be utilised.

In contrast to other teams, there are particular challenges regarding the iOS software development process. Other subjects are comparatively simpler to Android. There is less segmentation because the amount of supported hardware and devices is limited, which makes integration testing even more compelling.

If feasible, limitations that are bound to the Apple ecosystem should be avoided or abstracted. Examplarily, as some product owners (PO) of Catty do neither own a Mac nor an iPhone, PO pull request (PR) reviews have not been possible. An isolated environment of iOS in respect to other Catrobat teams should be prevented.

4.1. Situation before CI introduction

All required actions for integrating new source code were performed manually by a reviewer. Provision of builds for PRs and other useful processes described in chapter 3 were not available.

Human error ratio is a substantial part of software failures emerging from fully manual PR reviews. The reviewer had to first pull the code changes submitted in the PR from the VCS. Then the code is checked on local machines for validity, coding style and other standards. After that, the integration tests are run within the local Xcode installation. As UI tests are a lengthy process (for a detailed explanation, see 6.3.1), the whole development process is slowed down. Furthermore, the motivation of the stakeholders is decreased because of the tedious adoption of new features and code changes.

To improve overall understanding of the source code, PR reviews should have been conducted by every team member, including junior members. Due to the difficulty of this setup, this demand was hardly ever achieved.

Furthermore, the distribution of new release builds was manually executed from senior project members.

4.2. Challenges at setting up CI

Xcode - the IDE for iOS Development - only works on Apple's macOS. As a result, a dedicated machine had to be installed to cover this task. According to the requirements a Mac mini was integrated into the existing network of the CI infrastructure.

Due to the already present infrastructure along with the knowledge and experience of the Catrobat Jenkins team, the decision was made to use Jenkins as CI server for Catty. At the beginning, other solutions besides Jenkins have been evaluated (Circle CI[1], Travis CI[2] and Xcode Server).

Travis CI and Circle CI are popular choices at FOSS projects as they offer free plans for the open source community. Xcode Server is Apple's official approach for iOS Continuous Integration. It is using Bots (comparable to Jobs at Jenkins) to map the workflow. Besides traditional CI tasks (run tests, build), it is limited in functionality; being more widespread at smaller teams

[1]`https://circleci.com/`, visited on 05/15/2019
[2]`https://travis-ci.com/`, visited on 05/15/2019

and individual developers. After previous research of another Catrobat project member, using Xcode Server was not further pursued.

The chosen Jenkins setup is a standalone solution, which results in less dependencies and full control over the process, but also in administration overhead. Other reasons for choosing Jenkins are the wide range of plugins/third-party integrations and the usage of a fully FOSS CI solution.

Moreover, in-depth knowledge to various topics had to be acquired, that are automatically handled by Xcode when executed manually. This includes issues like code signing for builds (development, ad-hoc, App Store - already discussed in 2.3), as well as communicating with the App Store Connect API (see 7.2.1) among others.

5. Infrastructure and Tools

For the implementation of Continuous Integration, several tools and different software were used. In the next subchapters the major cornerstones are explained in detail and how they have been integrated in the CI pipeline. Also, other third-party tools and frameworks (SwiftLint - covered in 6.2.2, Carthage - covered in 6.2.1) were facilitated that are covered in the corresponding topics in chapter 6.

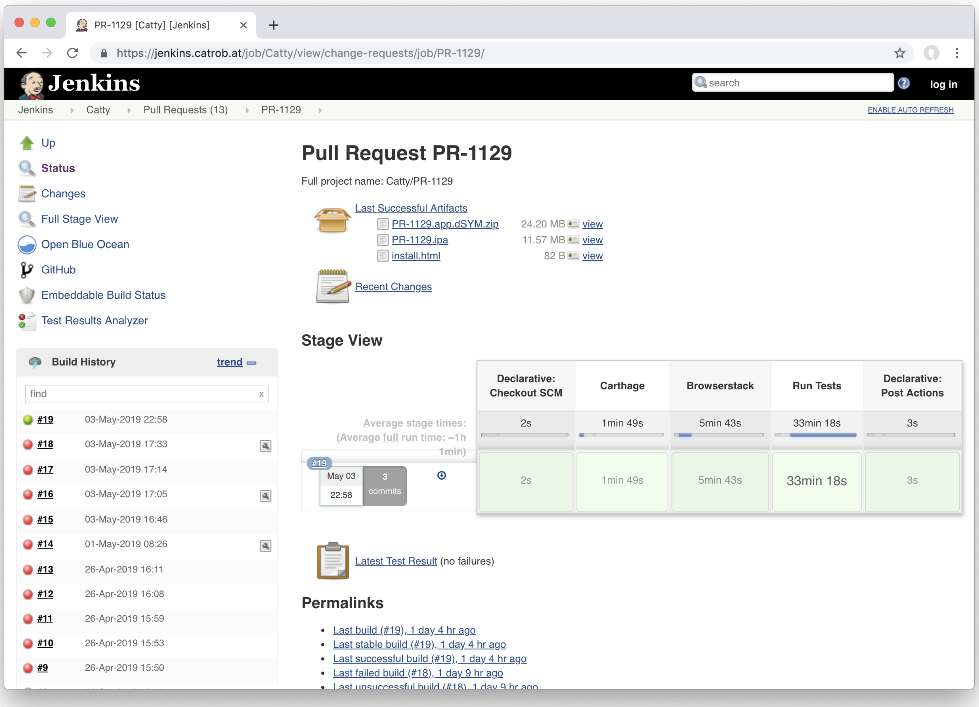
5.1. Automation Server - Jenkins

The development of Jenkins started in 2004. Back then, it was known as "Hudson", engineered and owned by Sun Microsystems. Jenkins is written in Java and functioning as CI Automation Server at Catrobat.

The present Jenkins infrastructure consists of the following Jenkins nodes:

- master
  - Slave1-HardwareSensorBox
  - Slave2_emulator
  - Slave3_emulator
  - Slave4
  - *SlaveMAC_Mini*

The master server is handling all requests, Slave1 to Slave4 are Linux nodes used for CI at Android-related projects of Catrobat. For this thesis, the macOS "SlaveMAC_Mini" was put into operation. The communication between the master and the node is accomplished via SSH connections authenticated by SSH keys. The requirement for setting up a Jenkins node at macOS is that the Java Development Kit (JDK) is installed beforehand.



PR-1129 [Catty] [Jenkins]

https://jenkins.catrob.at/job/Catty/view/change-requests/job/PR-1129/

Jenkins

search

log in

Jenkins › Catty › Pull Requests (13) › PR-1129 ›

ENABLE AUTO REFRESH

- Up
- Status
- Changes
- Full Stage View
- Open Blue Ocean
- GitHub
- Embeddable Build Status
- Test Results Analyzer

Build History trend

find

#19	03-May-2019 22:58
#18	03-May-2019 17:33
#17	03-May-2019 17:14
#16	03-May-2019 17:05
#15	03-May-2019 16:46
#14	01-May-2019 08:26
#13	26-Apr-2019 16:11
#12	26-Apr-2019 16:08
#11	26-Apr-2019 15:59
#10	26-Apr-2019 15:53
#9	26-Apr-2019 15:50

Pull Request PR-1129

Full project name: Catty/PR-1129

Last Successful Artifacts

PR-1129.app.dSYM.zip	24.20 MB	view
PR-1129.ipa	11.57 MB	view
install.html	82 B	view

Recent Changes

Stage View

	Declarative: Checkout SCM	Carthage	Browserstack	Run Tests	Declarative: Post Actions
Average stage times: (Average full run time: ~1h 1min)	2s	1min 49s	5min 43s	33min 18s	3s
#19 May 03 22:58 3 commits	2s	1min 49s	5min 43s	33min 18s	3s

Latest Test Result (no failures)

Permalinks

- Last build (#19), 1 day 4 hr ago
- Last stable build (#19), 1 day 4 hr ago
- Last successful build (#19), 1 day 4 hr ago
- Last failed build (#18), 1 day 9 hr ago
- Last unsuccessful build (#18), 1 day 9 hr ago

Figure 5.1.: Jenkins - PR pipeline

A screenshot of a finished PR job is illustrated in 5.1. Most information about running and finished Jenkins jobs is publicly accessible under: `https://jenkins.catrob.at/`

The configuration of Catty's Jenkins job is directly integrated in the VCS. Changes to the workflow can be handled as any other source code change. The complete process, known as CI pipeline, is divided into different stages. Each stage has its own responsibility and is aborting when a problem occurs or at failures. As a result, the problem area is evident at a glance.

The current configuration file can be found here[1] or in the following listing (5.1):

[1] `https://github.com/Catrobat/Catty/blob/master/Jenkinsfile/`, visited on 05/15/2019

Listing 5.1: Jenkinsfile - Pipeline configuration

```
#!/usr/bin/env groovy
pipeline {
 agent {
  label 'MAC'
 }

 options {
  timeout(time: 2, unit: 'HOURS')
  timestamps()
  buildDiscarder(logRotator(numToKeepStr: '30',
                  artifactNumToKeepStr: '30'))
 }

 stages {
   stage('Carthage') {
    steps {
    sh 'make init'
   }
  }
  stage('Browserstack') {
   steps {
    sh 'cd src && fastlane po_review'
   }
  }
  stage('Run Tests') {
   steps {
    sh 'cd src && fastlane tests'
   }
  }
 }

 post {
  always {
   junit
    testResults:
      'src/fastlane/test_output/TestSummaries.xml',
      allowEmptyResults: true
    archiveArtifacts(
     artifacts: 'src/fastlane/builds/',
     allowEmptyArchive: true)
    archiveArtifacts(artifacts:
     'src/fastlane/install.html',
     allowEmptyArchive: true)
  }
 }
}
```

As the pipeline shows, the Jenkins stages are essentially functioning as scheduler for the automation tool Fastlane, which brings us to the following chapter.

5.2. App Automation Tool - Fastlane

Fastlane[2] is a tool by the Austrian developer Felix Krause, which started as a university project. It was later acquired by Fabric (Twitter) in 2015 and Fabric ultimately by Google in 2017. Nonetheless it is an open source project driven by community engagement. It originated as solution to handle the complex task of software delivery for iOS apps by providing a set of command line tools. By now, it is also supporting Android Development.

With Fastlane, reoccurring tasks can be defined as "lanes". These lanes, written in the scripting language Ruby (a Swift version is currently in Beta), are more than a configuration file with full OOP support. This comes with the benefit of reusing lanes, combining lanes and improved readabilty of the code.

[2]https://github.com/fastlane/fastlane/, visited on 05/15/2019

Figure 5.2.: Fastlane features, taken from Krause, 2019.

The cornerstones of fastlane are illustrated in figure 5.2 and consist of the following actions:

deliver Upload screenshots, metadata and binaries to ASC. The app version can also be automatically submitted for App Review

pem Helper for Push certificate handling. Signing requests are uploaded to Apple, certificates are downloaded.

produce Creation and Modification of iOS apps on ASC. Supports enabling Apple-related services (Gamecenter, HomeKit, Siri, ...)

snapshot Producing localized screenshots for different languages and device sizes, configurable with a "Snapfile" (see chapter 7.2.2) and processable by the "deliver" action.

sigh Handling Provisioning Profiles.

gym Building and signing IPA files - helper for a command line build of Xcode (xcodebuild).

frameit Improving the appearance of screenshots by adding device frames or insertion of tag lines.

cert Handling code signing certificates.

scan Running tests on simulator or connected hardware and producing reports (Junit, Code coverage).

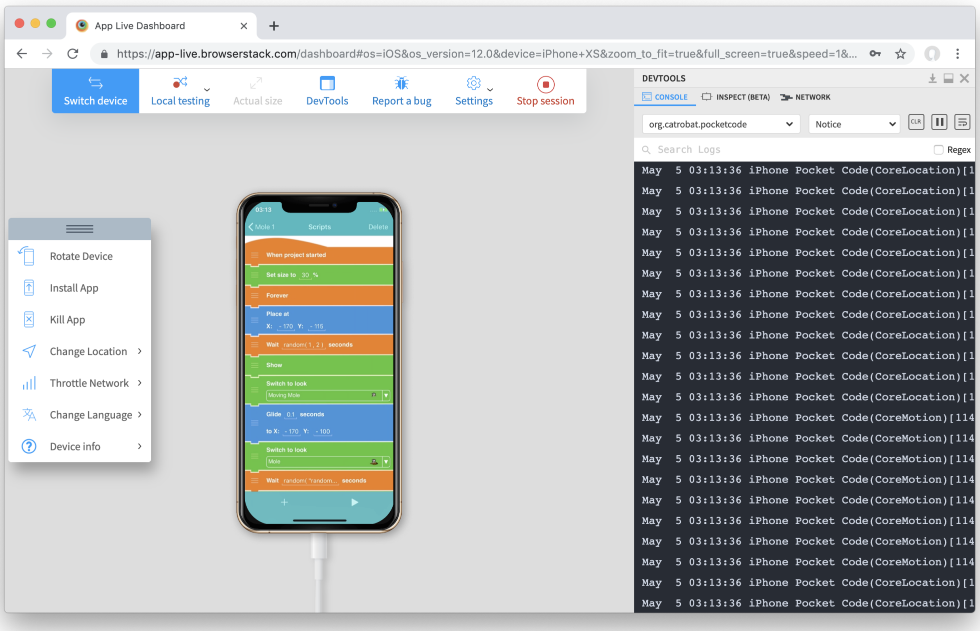
Listing 5.2: Extract of Fastfile

```
...
desc "Upload Development Build to Browserstack"
lane :upload_to_browserstack do
  upload_to_browserstack_app_live(
    browserstack_username: ENV["BS_USERNAME"],
    browserstack_access_key: ENV["BS_ACCESS_KEY"],
    file_path: $build_dir+$branch_name+".ipa"
  )
end

...

desc "Prepare for PO Review"
lane :po_review do
  cert
  sigh(adhoc: true)
  create_build scheme:$catty_schemes["release"],
               method:"ad-hoc"
  upload_to_browserstack
end
...
```

Listing 5.2 is an example lane triggered in a stage of the Jenkins integration job. In the “po_review” lane, provisioning profiles and certificates are updated at first. Then a build is created and uploaded to Browserstack (handled on the following page). Moreover, the build is archived as Artifact of Jenkins, so that iOS devices can also install the build when visiting the Jenkins portal on their smartphone. The necessary usernames and access tokens are obfuscated with environment variables and stored at the Jenkins node configuration.



https://app-live.browserstack.com/dashboard#os=iOS&os_version=12.0&device=iPhone+XS&zoom_to_fit=true&full_screen=true&speed=1&...

org.catrobat.pocketcode — Notice

```
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreMotion)[114
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
May  5 03:13:36 iPhone Pocket Code(CoreLocation)[1
```

Figure 5.3.: In-Browser-testing with Browserstack.

5.3. In-Browser App Testing - Browserstack

Browserstack is a commercial product, offering the ability to run iOS apps within the browser. Catrobat was provided with 5 user accounts of Browserstack for being an educational open source project.

Builds can be uploaded on the website or by using an API. The interface is even accepting Ad-hoc builds, as Provisioning Profiles are replaced. To be executable on their simulators, the app is re-signed by Browserstack. As visible in figure 5.3, the simulator is fully operable and a useful tool not only for product owners, but also for other Catrobat teams who have got no permanent access to iOS devices.

Log output is visualized on the right side of the screen. Besides test cases for which sensors (accelerometer, location, camera) are needed, Catty has no limitations running on this web service.

5.4. Internationalization & Localization - Crowdin

Internationalization (i18n) and Localization (l20n) at all Catrobat projects is handled by Crowdin[3]. Crowdin is a collaboration web tool for Translation Management. Catrobat projects are uploading their translation resource files in English, after that these get transcribed by the community of translators. Different file formats are supported, like .xml, .csv, .txt and the format used by iOS: .strings

All necessary strings (App- and App Store-specific) are stored on Crowdin with the following structure:

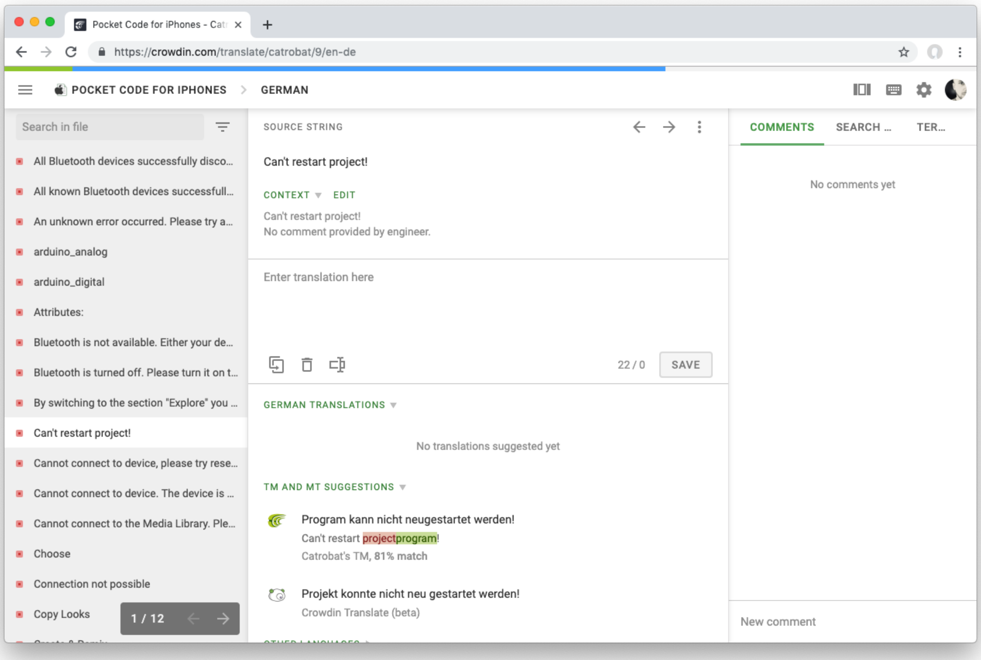
```
catty
├── App
│   └── Locale: en.lproj, pt_BR.lproj, ru.lproj, zh-Hans.lproj...
│       └── Localizable.strings
└── AppStore
    └── Locale: en, de, ru, ...
        ├── description.txt
        ├── keywords.txt
        └── subtitle.txt
```

Listing 5.3: Crowdin - Export settings

```
App:
/catty/%osx_code%/%original_file_name%
App Store:
/catty/AppStore/%osx_locale%/%original_file_name%
```

Listing 5.3 shows the export settings of Catty translations. With the placeholder "%osx_code%" (provided by Crowdin), a usable folder structure for Xcode - with the format [language designator]-[script designator]_[region designator] - is automatically created.

[3]https://crowdin.com/project/catrobat, visited on 05/15/2019



All Bluetooth devices successfully disco...
All known Bluetooth devices successfull...
An unknown error occurred. Please try a...
arduino_analog
arduino_digital
Attributes:
Bluetooth is not available. Either your de...
Bluetooth is turned off. Please turn it on t...
By switching to the section "Explore" you ...
Can't restart project!
Cannot connect to device, please try rese...
Cannot connect to device. The device is ...
Cannot connect to the Media Library. Ple...
Choose
Connection not possible
Copy Looks

Can't restart project!

Can't restart project!
No comment provided by engineer.

No translations suggested yet

Program kann nicht neugestartet werden!
Can't restart projectprogram!
Catrobat's TM, 81% match

Projekt konnte nicht neu gestartet werden!
Crowdin Translate (beta)

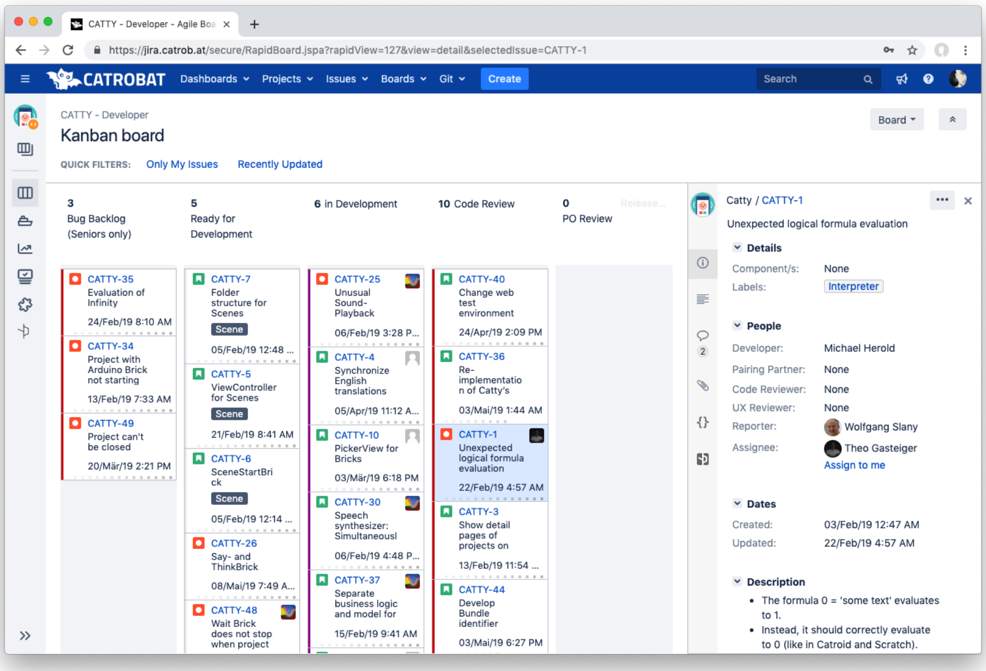
No comments yet

Figure 5.4.: Crowdin - translation view.

The GUI for translation input is displayed in figure 5.4. As indicated, Crowdin offers suggestions for translators to use already existing translations from a translation memory of all combined Catrobat projects, as well as independent translation suggestions from Crowdin (in beta).

Untranslated items are identifiable by a red status indicator on the left side-menu. In the main menu all available languages can be selected. The current progress of translations by languages is displayed - broken down to the Catrobat platform.

As Translation Management is an ongoing process, it should become a non-factor for developers. The CI sequence is automatically handling all synchronization activity from a software developer point of view. This includes uploading new or modified data, but also downloading and integrating all language data from Crowdin.


CATTY - Developer - Agile Boa
https://jira.catrob.at/secure/RapidBoard.jspa?rapidView=127&view=detail&selectedIssue=CATTY-1
CATROBAT
Dashboards
Projects
Issues
Boards
Git
Create
Search
Board

CATTY - Developer

Kanban board

QUICK FILTERS: Only My Issues Recently Updated

3 Bug Backlog (Seniors only)	5 Ready for Development	6 in Development	10 Code Review	0 PO Review
CATTY-35 Evaluation of Infinity 24/Feb/19 8:10 AM	CATTY-7 Folder structure for Scenes Scene 05/Feb/19 12:48 ...	CATTY-25 Unusual Sound-Playback 06/Feb/19 3:28 P...	CATTY-40 Change web test environment 24/Apr/19 2:09 PM	
CATTY-34 Project with Arduino Brick not starting 13/Feb/19 7:33 AM	CATTY-5 ViewController for Scenes Scene 21/Feb/19 8:41 AM	CATTY-4 Synchronize English translations 05/Apr/19 11:12 A...	CATTY-36 Re-implementation of Catty's 03/Mai/19 1:44 AM	
CATTY-49 Project can't be closed 20/Mär/19 2:21 PM	CATTY-6 SceneStartBrick Scene 05/Feb/19 12:14 ...	CATTY-10 PickerView for Bricks 03/Mär/19 6:18 PM	CATTY-1 Unexpected logical formula evaluation 22/Feb/19 4:57 AM	
	CATTY-26 Say- and ThinkBrick 08/Mai/19 7:49 A...	CATTY-30 Speech synthesizer: Simultaneousl 06/Feb/19 4:48 P...	CATTY-3 Show detail pages of projects on 13/Feb/19 11:54 ...	
	CATTY-48 Wait Brick does not stop when project	CATTY-37 Separate business logic and model for 15/Feb/19 9:41 AM	CATTY-44 Develop Bundle identifier 03/Mai/19 6:27 PM	

Catty / CATTY-1

Unexpected logical formula evaluation

Details

Component/s:	None
Labels:	Interpreter

People

Developer:	Michael Herold
Pairing Partner:	None
Code Reviewer:	None
UX Reviewer:	None
Reporter:	Wolfgang Slany
Assignee:	Theo Gasteiger Assign to me

Dates

Created:	03/Feb/19 12:47 AM
Updated:	22/Feb/19 4:57 AM

Description

- The formula 0 = 'some text' evaluates to 1.
- Instead, it should correctly evaluate to 0 (like in Catroid and Scratch).

Figure 5.5.: Jira - Kanban Board of Catty, taken 10/05/2019.

5.5. Project Management - Jira

Jira is an online tool used for Project Management of the Agile software process at Catrobat. Occurring issues, bugs and new features from planning games are categorized in Kanban Boards. Anderson, 2010 is showing priorization approaches, when using the Kanban method. Product Owners and Project Coordinators are able to prioritize and control the process of current development.

Items with the status “Ready for Development” are free for any Developer to implement (example in 5.5). When starting to work on an issue, it needs to be moved to “in Development”. With a comment function, changes can be discussed within the team, but also questions to the reporter can be clarified. Jira has an integration to Github (see 6.1) for direct linking of associated branches, commits and PRs.

6. Continuous Integration

During the practical part, the theoretical knowledge from chapter 3 was realized for Catty. The upcoming sections cover the corresponding parts of a Continuous Integration system.

As development for Catty was already ongoing, CI was progressively introduced to prevent any potential issues whilst adoption.

6.1. Version Control System - Github

As Duvall, Matyasand, and Glover, 2007 pointed out, version control systems are a basic requirement for CI. Even without using CI, source code should be handled within version control - whether it is Git or another type (SVN, Mercurial).

Catrobat is using Github, as numerous other open source projects. The Catty repository is available at: `https://github.com/catrobat/catty/`

An overview of how Github is integrated into the CI process is illustrated in 6.1. The representation is adopted to reflect the environment of Catty.

With the help of webhooks, Github is communicating PR- and branch-changes to the CI master server. The Jenkins master is scheduling a job to the Mac node, running the consecutive stages of the CI pipeline. After the job has finished the results are propagated back to Github to reflect the status of the task.

The interaction possibilities for developers with the VCS are extended by CI feedback, giving additional information on PRs by providing results of CI jobs (more in 6.1.2)

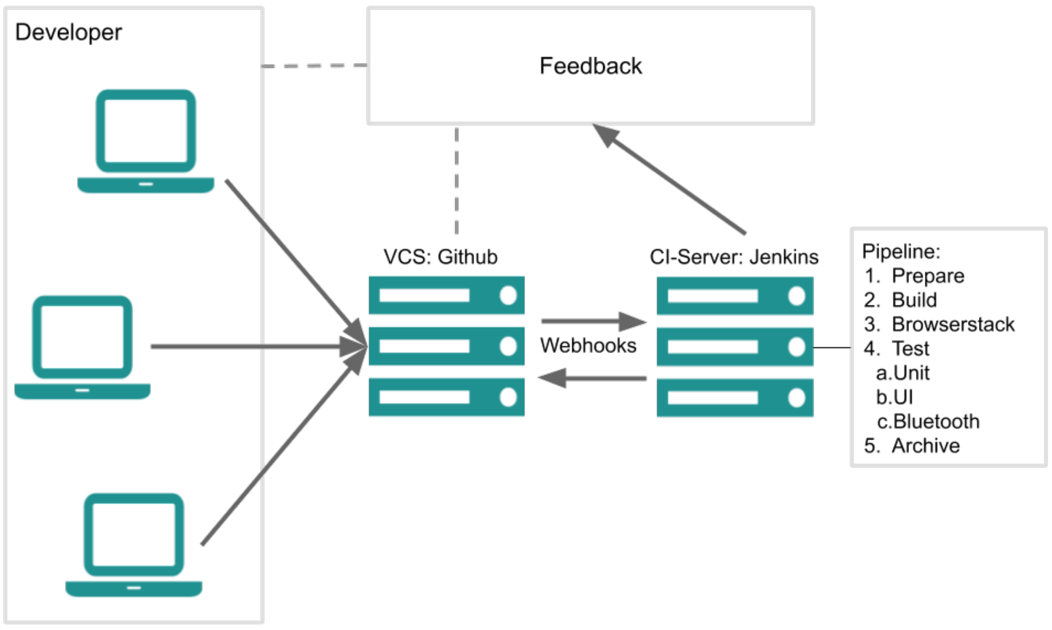


Figure 6.1.: CI Overview, adopted from Duvall, Matyasand, and Glover, 2007, p.5.

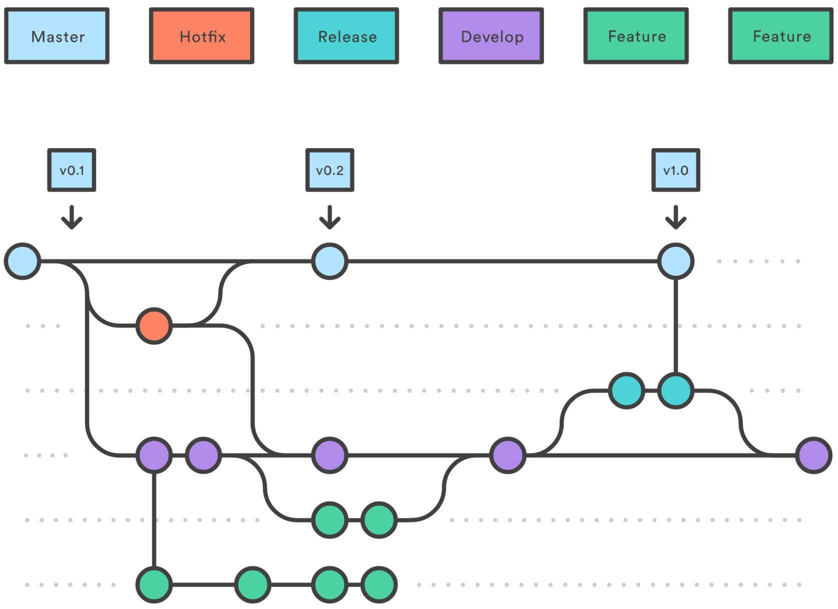


Figure 6.2.: Gitflow - example illustration, taken from Atlassian, 2019.

6.1.1. Gitflow for Pocket Code

In the course of this thesis Catrobat's workflow for code integration was applied to Catty. The process is based on Gitflow by Atlassian, 2019 with slight modifications. The presented example in 6.2 is divided into the following branches (with the distinction of feature branches at Catrobat):

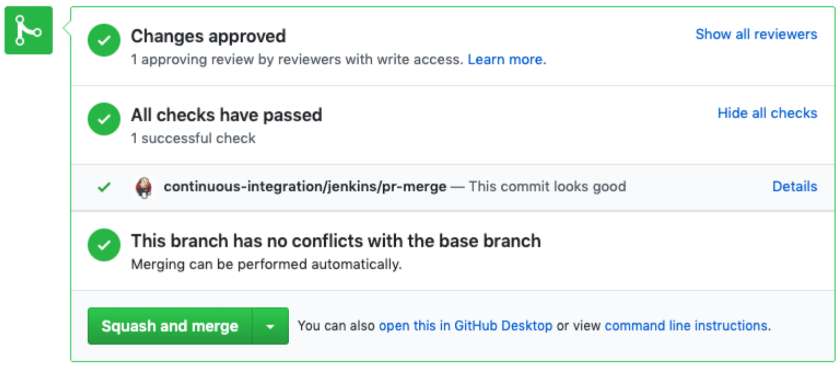
master reflects the currently released version.

hotfix is used to patch bugs in production code. It is the only fork allowed from master. Respective patches are also adopted in the develop branch.

release branches are created whenever a new version is about to be deployed.

develop reflects the current state of changes. It acts as base for feature branches and is used for PR creation. It represents the mainline of the "Maintain a Single Source Repository" principle of Fowler and Foemmel.

feature Catrobat: branches are maintained at the user forks of the mainline.



Changes approved
1 approving review by reviewers with write access. Learn more.
Show all reviewers

All checks have passed
1 successful check
Hide all checks

continuous-integration/jenkins/pr-merge — This commit looks good
Details

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Figure 6.3.: Github - Jenkins integration.

6.1.2. Pull Request Integration

The credo at development and review of Pull Requests (PR) is to "Keep it green". Every time a PR is created, a CI integration job is automatically scheduled. It is helping the stakeholders to follow the build and test status of the changes. Whenever new commits are pushed to the branch, associated with the PR, the process is repeated.

The *master* and *develop* branch, mentioned in 6.1.1, are branch-protected, which means that it is only allowed to merge changes if at least one reviewer from the Catrobat team has verified and approved the performed changes. The possibility to only merge code, that has successfully passed the CI workflow, is available - however not activated. It is up to the reviewer to check if the reason for failed CI checks are comprehensible and not resulting in potential issues.

Figure 6.3 shows the feedback of Github for a PR ready to merge, after changes were approved by a reviewer and the CI job was succeeding.

6.2. Build Process

The first stage of the CI workflow verifies that the changes are successfully buildable. Before every build process, caches (iOS: derived data) are deleted and simulators are reset to prevent interference with previous runs. Successful builds are a precondition for every other step of the CI job and should hamper the "It works on my machine" phenomenon of development. Before every build the dependencies of Catty are handled, covered in the next chapter.

6.2.1. Dependency Management - Carthage

Dependency management is necessary to improve maintenance of third-party tools and frameworks. There are two solutions with widespread usage at iOS development: Cocoapods, and Carthage
(Note: A similar tool used at Android Development is Gradle.)

Cocoapods

Xcode workspaces (.xcworkspace) are required to use Cocoapods. Cocoapods creates a workspace with the project and all the dependencies listed in a corresponding configuration file (Podfile). The dependencies are built at the same process as the app itself. Catty has previously been using Cocoapods, but migrated to Carthage.

Carthage

Carthage, however, follows the approach of pre-building the dependencies and linking the frameworks within the existing project file. This is resulting in faster build times and less overhead within Xcode. The configuration file (Cartfile) is comparable to Cocoapods. With a simple command "carthage bootstrap" the dependencies are automatically fetched and pre-built. Moreover, Carthage is written in Swift which makes it easier to extend for iOS developer, compared to Cocoapods written in Ruby. Running Carthage is the first step of the CI pipeline after the source code is fetched from the repository.

6.2.2. Coding Standard - Linting

To improve overall code quality at Catty, SwiftLint[1] was introduced. SwiftLint is used to define rules for styling and coding conventions, based on Github's Swift Style Guide. The tool is producing additional warnings and errors - depending on the severity of the violation - already at the development stage.

Development at Catty is following the principles of Clean Code by Martin, 2008, which is explaining the problem of Code Smells. Habchi et al., 2017 found 6 Code Smells specific to iOS. They also observed that the quality metrics of Objective-C and Swift apps are very differntial.

Additional to the integrated warnings and errors of Xcode, 99 more rules have been activated during the course of this thesis. The existing code was refactored to satisfy the conditions.

Examples for styling checks: (descriptions from *SwiftLint Rules*):

colon Colons should be next to the identifier when specifying a type and next to the key in dictionary literals.
operator_usage_whitespace Operators should be surrounded by a single whitespace when they are being used.
trailing_semicolon Lines should not have trailing semicolons.

Examples for coding convention checks (descriptions from *SwiftLint Rules*):

anyobject_protocol Prefer using AnyObject over class for class-only protocols.
empty_count Prefer checking isEmpty over comparing count to zero.
discouraged_object_literal Prefer initializers over object literals.

Some violations are able to be auto-corrected by SwiftLint, but only trailing whitespaces are automatically removed. This feature helps to standardize the code appearance across the code base. For educational purposes of all contributing developers the warnings should be taken notice of instead of being automatically processed at other violations.

[1] `https://github.com/realm/SwiftLint/`, visited on 05/15/2019

6.3. Running Automated Tests

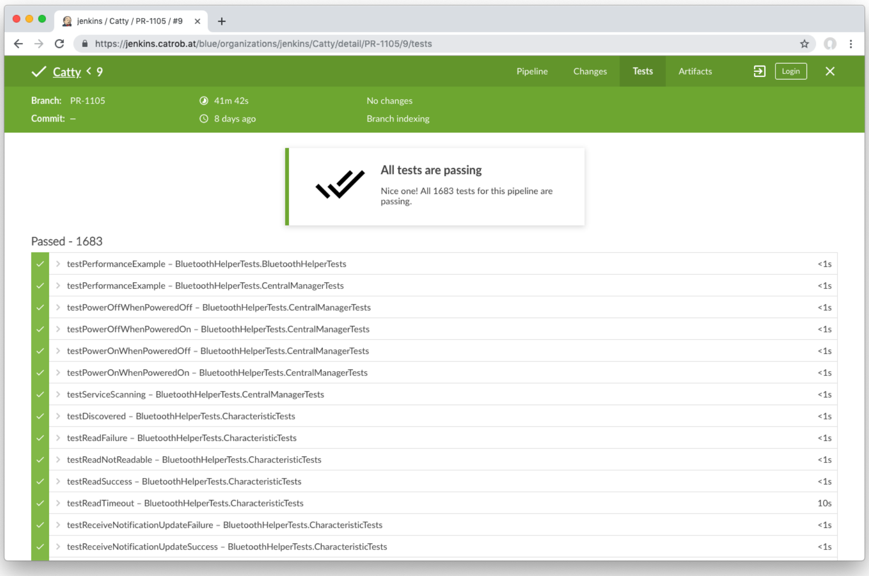
Catty is using the method of Test Driven Development (TDD) in practice. Hauser, 2017 presented in his work about TDD for iOS the method of "Red", "Green", "Refactor", in allusion to the status colour of a test run. Red means that a test failed, green indicates that a test has passed. Hereby, when working on an issue, a test is written first that is validating the desired outcome of the changes. After the testing code is ready, the actual implementation is realized. In the next stage, the developer is programming the actual changes until the it reaches the "Green" stage by any means. Finally, the code is getting refactored to meet the Coding Standards of Catrobat.

As a result of the TDD approach, testing is getting special attention within the Catty team. The overall test suite is providing meaningful feedback to the developer, if the changes do any harm to the functionality of the software. By the time of writing (May, 2019) almost 1700 tests are implemented that can be divided into the following categories:

- (1526) Unit Tests
- (84) Bluetooth Unit Tests (introduced during the work of Slavec, 2016)
- (73) UI Tests

The testing framework XCTest is available since Xcode 5 to simplify testing iOS Apps. It is providing a rich set of features for both Unit, and UI tests. Previous to CI implementation, all testing was executed by all participants involved in a Pull Request separately. Tests had to be run manually on the local machines for verification purposes.

With the introduction of CI, test runs are automatically processed on a separate build machine (Mac Jenkins node). It is acting as neutral judge for running the tests. The test run is started by a fastlane plugin on a defined target device running with a configurable iOS version. In contrast to Xcode the test run is executed headless with the help of mapped command line tools. At the end the results of the different testing schemes are concatenated as Junit reports, which are processable by Jenkins to generate a rich UI for visual feedback.


jenkins / Catty / PR-1105 / #9
https://jenkins.catrob.at/blue/organizations/jenkins/Catty/detail/PR-1105/9/tests
Catty < 9
Pipeline
Changes
Tests
Artifacts
Login

Branch: PR-1105
Commit: –
41m 42s
8 days ago
No changes
Branch indexing

All tests are passing

Nice one! All 1683 tests for this pipeline are passing.

Passed - 1683

✓	testPerformanceExample – BluetoothHelperTests.BluetoothHelperTests	<1s
✓	testPerformanceExample – BluetoothHelperTests.CentralManagerTests	<1s
✓	testPowerOffWhenPoweredOff – BluetoothHelperTests.CentralManagerTests	<1s
✓	testPowerOffWhenPoweredOn – BluetoothHelperTests.CentralManagerTests	<1s
✓	testPowerOnWhenPoweredOff – BluetoothHelperTests.CentralManagerTests	<1s
✓	testPowerOnWhenPoweredOn – BluetoothHelperTests.CentralManagerTests	<1s
✓	testServiceScanning – BluetoothHelperTests.CentralManagerTests	<1s
✓	testDiscovered – BluetoothHelperTests.CharacteristicTests	<1s
✓	testReadFailure – BluetoothHelperTests.CharacteristicTests	<1s
✓	testReadNotReadable – BluetoothHelperTests.CharacteristicTests	<1s
✓	testReadSuccess – BluetoothHelperTests.CharacteristicTests	<1s
✓	testReadTimeout – BluetoothHelperTests.CharacteristicTests	10s
✓	testReceiveNotificationUpdateFailure – BluetoothHelperTests.CharacteristicTests	<1s
✓	testReceiveNotificationUpdateSuccess – BluetoothHelperTests.CharacteristicTests	<1s

Figure 6.4.: Successful test result page of a Jenkins job.

The feedback of a successful test run is shown in figure 6.4. Reports include all testcase status and the corresponding execution time. Other information about the CI job - like performed changes, pipeline command line output and resulting Artifacts - can be easily accessed via the provided navigation.

6.3.1. Problems & Limitations

Several problems and limitations had to be tackled during the introduction of automated testing. In the sense of the “Keep the Build Fast” CI principle, the overall runtime of the tests was not entirely satisfying. Mainly due to slow UI testing in comparison to Unit testing. This is caused by the nature of UI tests, simulating how a user would operate in a real-world situation by mimicking clicks and producing view transitions with UI animations.

To decrease UI test runtimes, the animation speed was increased. However, this was only marginally an imporevement. In general, Unit tests should be preferred in future development in many areas, solely for speed reasons. Then again, UI tests remain an important part of the test process and support acceptance testing in a possible Continuous Deployment environment.

UI tests are popular at developers new to Software Testing, as the tests act in a natural, human way. At unit testing deeper knowledge is required: Some data that is only available at runtime has to be simulated, for instance by Mocking. Nolan, 2016 is presenting a showcase of how Mocking in Swift can be achieved with the Mocking framework Cuckoo[2].

Flaky UI Tests

When running automated tests, "flaky" UI tests were observed. Flaky means, that the test is not reliable and failing randomly (Mascheroni and Irrazabal, 2018). 3 to 4 test cases have been identified that showed this behaviour increasingly. Test cases dependent on animations and alerts are especially concerned. Some cases were able to be limited by using *Expectations* (telling the test case to wait for UI to catch up), but it was not reliably preventing all occurences.

To mitigate this behaviour Fastlane's testing action has been extended to allow re-running of failed test up to *n* times (production setting: 3). In the repeated test run only failed tests are restarted. In ideal circumstances this solution would not be necessary as the tested workflow should pass every time. However, it is the current solution in use to deal with this problem of nondeterminism. Flaky tests can be identified by the feedback of the CI system and should be addressed to be more stable.

[2]`https://github.com/Brightify/Cuckoo`, visited on 05/15/2019

6.4. Distribution

The distribution of the integration builds is supporting the following two categories and use cases:

- Abstract: Browser-based App Testing
- Physical: Signed Ad-hoc builds

For both cases a signed ad-hoc build is used. The browser-based solution has been shown in 5.3 by using the "App Live" Browserstack service. For installation on hardware devices the build is archived as Artifact on Jenkins. The necessary steps for this task are described in the following chapter.

6.4.1. Provision of Ad-hoc Builds

Executable IPA files (Android-counterpart: APK) do not offer one-click installation like Android. Providing that the device is included in the provisioning profile, with which the build was signed, the app can be installed in two ways.

a) After an IPA file is downloaded on a workstation it can be installed with the software iTunes. This process is not really appealing as it includes too much manual effort.

b) iOS is providing an URL-scheme to trigger app installations (itms-services://). An example hyperlink for a PR build installation is shown in 6.1. The ITMS scheme is called with a Manifest file containing information of the app (bundle identifier, version number, title) and a link to the corresponding IPA file that should be installed, 6.2. With this setup it is possible to provide installation links alongside the IPA files on Jenkins' Artifacts to enable simple installation on smartphones without the need of additional equipment.

Listing 6.1: Ad-Hoc Installation Link

```
<a href="itms-services://?action=download-manifest&
   url=https://jenkins.catrob.at/.../PR147-1.plist">
 Install PR on your device</a>
```

Listing 6.2: Ad-Hoc Manifest Installation File

```
<plist version="1.0">
 <dict>
  <key>items</key>
  <array>
   <dict>
    <key>assets</key>
    <array>
     <dict>
      <key>kind</key>
      <string>software-package</string>
      <key>url</key>
      <string>
       https://jenkins.catrob.at/.../PR147-1.ipa
      </string>
     </dict>
    </array>
    <key>metadata</key>
    <dict>
     <key>bundle-identifier</key>
     <string>org.catrobat.pocketcode.PR</string>
     <key>bundle-version</key>
     <string>0.6.9</string>
     <key>kind</key>
     <string>software</string>
     <key>title</key>
     <string>PR147#1</string>
    </dict>
   </dict>
  </array>
 </dict>
</plist>
```

7. Continuous Delivery

Several preparations for Continuous Delivery have been realized, that are discussed in this chapter. In the sense of XP, fast deployment of new features, reducing integration risks and other reasons, Catty is aiming for short release cycles.
Hereby, CD is not only important for less time spent on administrative tasks, but also to give Product Owners the possibility to deploy Catty without profound knowledge of the manual Xcode deployment workflow.

7.1. Pre-Release Actions

Whenever the decision is made to start a release process, a few required steps have to be handled before.

Firstly, according to the adopted Gitflow (6.1.1) a *release* branch is forked from the *develop* branch. The naming convention is release-[ReleaseNumber] (example: release-0.6.9). This also acts as feature stop, so that only bugfixes that occur during the beta test will be merged into the version.
Secondly, a fastlane action was developed to upload the currently active output strings to the Crowdin (5.4) platform for translators to interpret. This is accomplished by running the custom command "fastlane upload_crowdin". It is necessary as the Translation Service is only updating the data after running a build. However, the build process can neither be triggered by an API call nor with the official command-line tool provided by the publisher. Therefore, the build has to be executed via the Crowdin website. Generated translations are downloaded later during Deployment (7.2) and imported into the project.

Although the used tools are capable of automating this process as well, the pre-release process is currently manual by choice, as it can be carried out with simple command-line instructions. To minimize introduction problems, this workflow will be continuously transformed to automation.

7.2. Release Deployment

After the creation of a release branch, the CI integration pipeline is running automatically. If the integration tests are passing and the version looks ready, the distribution can be started. The deployment itself is handled by a separate Jenkins job, and can only be started with required user rights on the Jenkins server.

The release pipeline is running the following actions:

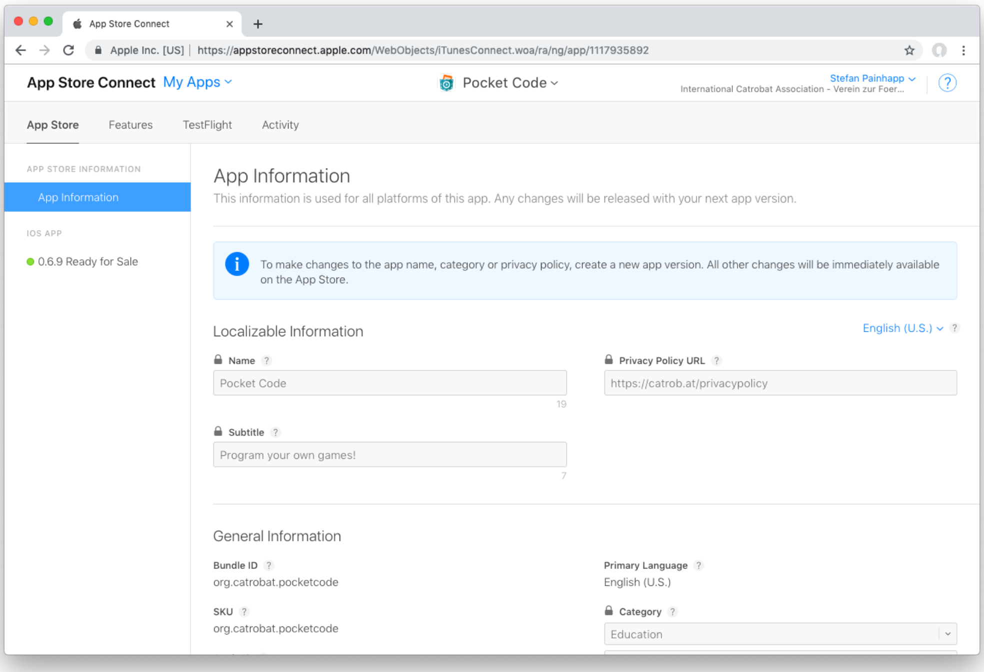
- Increase build number (and version number if necessary)
- Download translation data from Crowdin
- Prepare App Story entry with metadata from Crowdin (see 7.2.2)
- Run Snapshot scheme to update localized screenshots (see 7.2.2)
- Create Build with "App Store" certificate
- Upload data to ASC (build, metadata, screenshots)

After every upload of a binary to App Store Connect (see 7.2.1), the build number has to be increased as it is only allowed to use build numbers once. If the build number is not adapted, the process fails.

Depending every step above was successful and the job has successfully finished, the app is entering the next stage: beta testing (see 7.3).

In the event of occuring bugfixes in the release branch the deployment job has to be repeated. On the case of a successful beta test, the app will be submitted to Apple for App Review (see 7.4). App Review can either be started on the ASC platform manually or automatically within the release pipeline by a configuration setting.

The final production release needs to be treated with caution as reverting changes is not supported on the App Store. Hence, it is not possible to roll back to a previous, working, lower version. Even if serious issues occur a new release process has to be run through.


App Store Connect
https://appstoreconnect.apple.com/WebObjects/iTunesConnect.woa/ra/ng/app/1117935892
App Store Connect My Apps
Pocket Code
Stefan Painhapp
International Catrobat Association - Verein zur Foer...
App Store
Features
TestFlight
Activity
APP STORE INFORMATION
App Information
IOS APP
0.6.9 Ready for Sale
English (U.S.)

App Information

This information is used for all platforms of this app. Any changes will be released with your next app version.

To make changes to the app name, category or privacy policy, create a new app version. All other changes will be immediately available on the App Store.

Localizable Information

Name: Pocket Code

Privacy Policy URL: https://catrob.at/privacypolicy

Subtitle: Program your own games!

General Information

Bundle ID: org.catrobat.pocketcode

Primary Language: English (U.S.)

SKU: org.catrobat.pocketcode

Category: Education

Figure 7.1.: App Store Connect portal.

7.2.1. App Store Connect

App Store Connect[1] (previously known as iTunes Connect) is the app administration portal of iOS. All interactions with the App Store are managed on this platform.

Features of ASC include:

App Management Create, modify and delete listings on the App Store. Respond to user ratings and reviews.

Sales Statistics and trends of downloaded app units.

Analytics Reports on the usage: origin countries, active sessions, crashes and more.

Users Assign roles (Developer, App Manager, Marketer) to project members

[1] https://appstoreconnect.apple.com/, visited on 05/15/2019

with Apple-IDs.

Agreements Signing developer agreements and managing distributor data (company name, bank account).

Test device management (UUIDs), Signing certificates and provisioning profiles are not handled by ASC, but with the help of the Apple Developer portal[2].

7.2.2. Localized Metadata & Screenshots

The most time consuming part of manual publishing of iOS versions is a) preparing the metadata in different languages, and even more b) the production of localized preview images for different device sizes. The data has to be captured language by language for every intended change on the App Store Connect platform, if done by hand.

At Catty, it is possible to run this process automatically, because fastlane's "deliver" and "snapshot" actions are used in combination with Crowdin's translation data.

Metadata

Metadata, used at the "deliver" action, is autonomous from the build process. The "AppStore" folder structure mentioned in chapter 5.4 has to be downloaded, merged with the already existing English information and saved at a predefined path. The folder structure is using simple language identfiers in contrast to the .lproj structure used by Xcode.

During the upload of the new version the translated files - description.txt, keywords.txt, subtitle.txt - are transmitted to the App Store server in all supported languages.

[2] `https://developer.apple.com/account`, visited on 05/15/2019

Screenshots

A new build scheme has to be added to the project for using snapshots. Snapshots are realized as special form of UI Tests created in a separate target to not interfere with integration UI tests. Fastlane is providing a helper class “SnapshotHelper.swift” that deals with the image logic. A simplified example implementation is shown in 7.1. Whenever the simulator is navigated to a desired screen at the UI test, the “snapshot()” function is saving a preview image.

Listing 7.1: Snapshot UI Test - Code snippet

```
class CattyUISnapshots: XCTestCase {
    let app = XCUIApplication()

    override func setUp() {
        continueAfterFailure = false
        setupSnapshot(app)
        app.launch()
    }

    func appStoreScreenshots() {
        ... //navigate to first screenshot
        snapshot("0Launch")
        ... //navigate to next screenshot
        snapshot("1Scripts")
        ... //navigate to next screenshot
        snapshot("4Paint")
        ... //navigate to next screenshot
        snapshot("3MediaLibrary")
        ... //navigate to next screenshot
        snapshot("2Explore")
    }
}
```

Listing 7.2: Snapfile - Snapshot configuration

```
devices([
  "iPhone 8",
  "iPhone SE",
  "iPhone XS",
  "iPhone XR",
])

languages([
  "en-US",
  "de-DE",
  "it-IT",
  "ru",
  #add additional languages here
])

# The name of the scheme which contains the UI Tests
scheme("Snapshots")

# Where should the resulting screenshots be stored?
output_directory("./fastlane/screenshots")

clear_previous_screenshots(true)
```

The recommended usage of fastlane's "snapshot" action is by using a "Snapfile" configuration. An example of a Snapfile is indicated in listing 7.2. Devices and languages can be added to the configuration statically on demand or can be generated dynamically by evaluating the existing translations reported by Crowdin.

The result of the snapshot process is shown in figure 7.2. The images are only showing a specific device type, respectively screen size, for demonstration purposes.

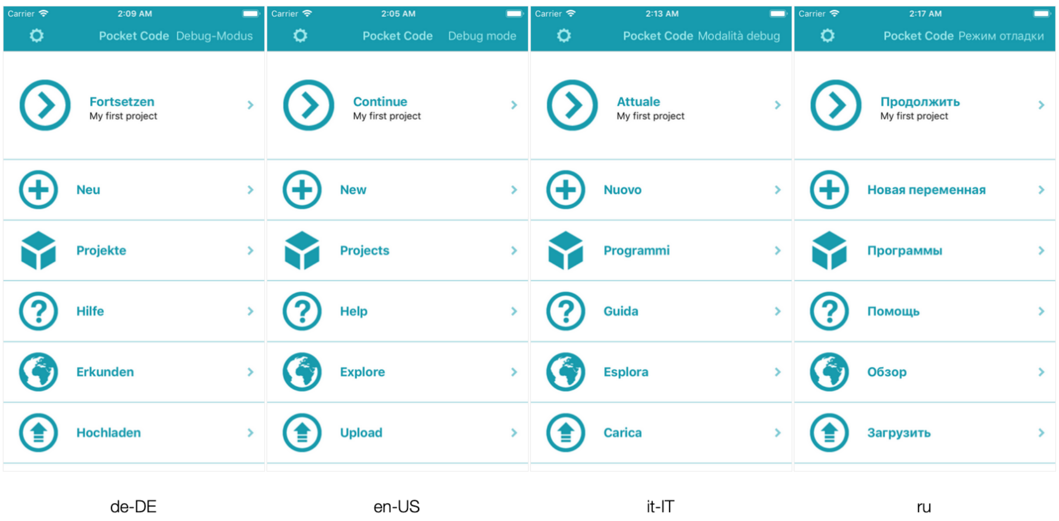
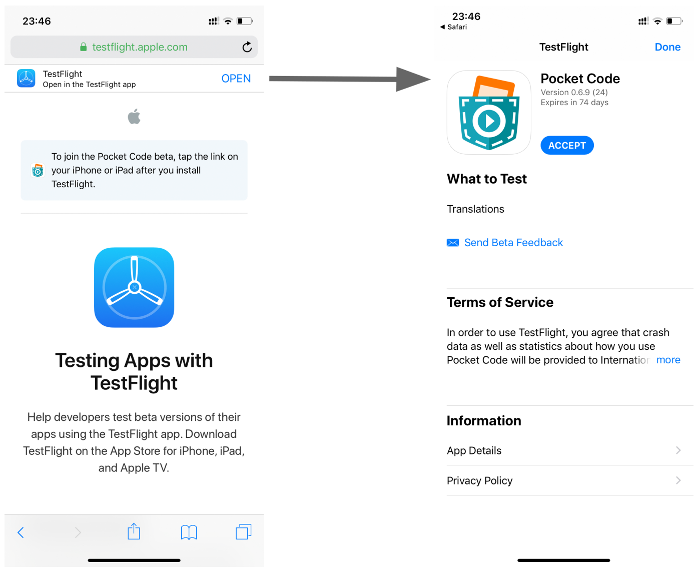


Figure 7.2.: Localized Screenshots


23:46
testflight.apple.com
TestFlight
Open in the TestFlight app
OPEN

To join the Pocket Code beta, tap the link on your iPhone or iPad after you install TestFlight.

Testing Apps with TestFlight

Help developers test beta versions of their apps using the TestFlight app. Download TestFlight on the App Store for iPhone, iPad, and Apple TV.

23:46
◂ Safari
TestFlight
Done
Pocket Code
Version 0.6.9 (24)
Expires in 74 days
ACCEPT

What to Test

Translations

Send Beta Feedback

Terms of Service

In order to use TestFlight, you agree that crash data as well as statistics about how you use Pocket Code will be provided to Internatio more

Information

App Details

Privacy Policy

Figure 7.3.: Public beta acceptance process.

7.3. Beta Testing

After uploading a build to the App Store infrastructure, it is available for publication to beta testers. The distribution is not handled by the "App Store" app on end devices, but with the separate "Testflight" app. Participants can be divided into internal and external testers:

internal Up to 25 members of the team, who have at least one of the following roles assigned: Admin, Technical, App Manager, Developer or Marketer.

external Up to 10,000 arbitrary, registered testers.

Internal testers are immediately able to start testing after the build has finished processing on the App Store server. For external testing to start, the version has to first run through Beta Review of Apple.

Since 26/09/2018 the public beta registration process is simplified by providing a web form for enrolment. Previously testers had to be captured manually at the App Store Connect (see 7.2.1) portal. The process of accepting an external testing invite for Pocket Code is shown in 7.3.

7.4. App Store Review

Apps have to pass a review process to be listed on Apple's App Store. The software is not only checked automatically for anomalies and misusage, but also manually to fulfil Apple's guidelines. Precise information about this process is not communicated to software publishers. If an infringement of a guideline occurs, the submission is getting rejected with a listing of reasons as response.

Common rejection reasons include:

- Crashes and/or Bugs
- Unfinished Content (Placeholder)
- Improper UI for mobile usage or touchscreens
- Metadata and/or Screenshot objections

Prior to the first Publication of Pocket Code on the App Store, the app was rejected because of software crashes. After eradicating the misbehaviour Pocket Code was rejected again due to accusation of running executable code from shared, downloaded *.catrobat files, which is prohibited at iOS. Accordingly, an appeal to the App Review Board was expressed, and the misconception could be clarified.

"The App Review Board provides the opportunity to appeal the rejection of an app if you believe that the functionality or technical implementation was misunderstood. You can submit additional details to the App Review Board to help them determine if your app should be reconsidered."[3]

Reoccurring issues for new app releases can not be fully mitigated. Emerging rejections need to be handled case by case in the future. This may prevent an otherwise possible, completely automated deployment during these incidents.

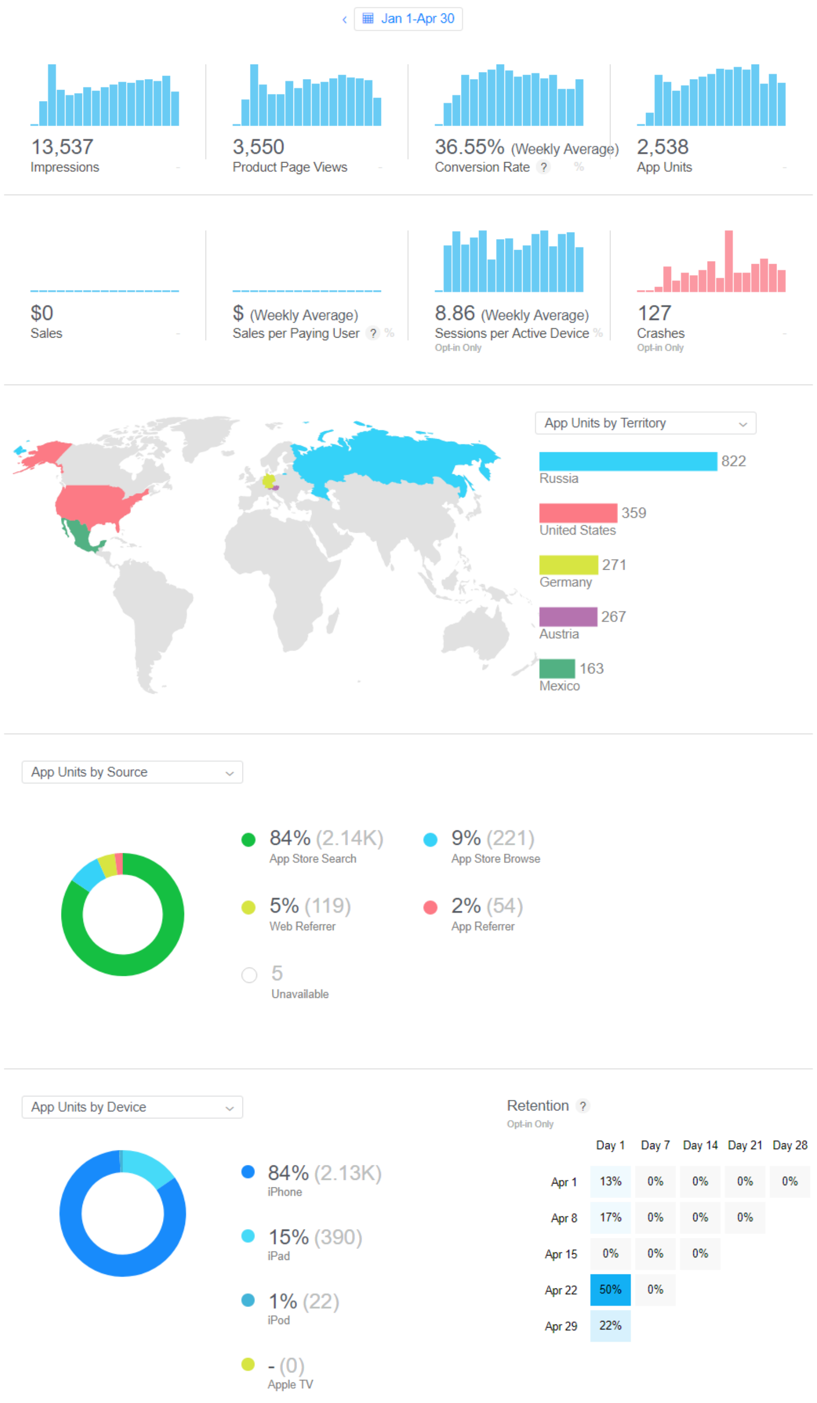
[3]https://developer.apple.com/app-store/review/, visited on 05/15/2019

7.5. App Store Optimization

An important part for user acquisition is a relatively new subject, called App Store Optimization (ASO). The goal is, similar to Search Engine Optimization (SEO), to improve the download rate within the smartphone stores (primarily Google Play Store and Apple's App Store). The App Store is using several key metrics for ranking, including ratings, title, short title, description and keywords. Keywords can be of 100 characters length divided by commas and should emphasize the use case of the app. They are used for search rankings within the App Store. The search algorithms are transforming rapidly and have not been analysed during this work. However, as Jung, Baek, and Lee, 2012 show, it is important for apps to be listed in the charts as it has a big impact on the performance of download numbers.

Commercial ASO products suggest to publish regular updates at least every 4 to 6 weeks to improve app visibility on the app store. During the timeframe of January and April three releases of Pocket Code were published. After every update the app was listed in the charts of the "Education" category. Though this is too little information to statistically proof the positive impact, regular updates enhanced by Continuous Delivery should be targeted to investigate further.

Figure 7.4 on the following page shows some statistical information of Pocket Code at the App Store during the timeframe of January to April 2019.



	Day 1	Day 7	Day 14	Day 21	Day 28
Apr 1	13%	0%	0%	0%	0%
Apr 8	17%	0%	0%	0%	
Apr 15	0%	0%	0%		
Apr 22	50%	0%			
Apr 29	22%				

Figure 7.4.: App Store Statistics for Pocket Code, Jan 2019 - Apr 2019.

Part III
Findings

8. Conclusion

In this thesis the principles of Continuous integration have been shown by the example of Pocket Code for iOS. The introduction of CI has proven to amplify philosophies of Test-Driven Development and certain CI principles like "Don't break the build".

Holck and Jørgensen, 2003 claimed that even the basic approach of Continuous Integration at FreeBSD and Mozilla in 2003 had a positive impact on their engineering process. For contributors of Catty the integration suite has turned out to be a valuable tool during development. Contrary, Vasilescu et al., 2014 discovered by analyzing (Java, Python, Ruby) Github repositories that many projects introduced CI, but are not using it.

Hence, it is important that the feedback of the CI system is fast and meaningful to programmers. On the basis of a Continuous Integration system, CD can help iOS projects to achieve short release cycles by implementing a set of tools presented in this work.

8.1. Outlook

Due to the complexity of setting up Continuous Integration structures, many projects are starting to use managed, hosted services. Also, for teams that would not fully utilise hardware, but would still like to use CI, it is a good starting point. Furthermore, administration overhead is cut down at managed solutions.

Apple's solution of Xcode Server Bots for CI at iOS development is not satisfying for large projects. With the acquisition of buddybuild, they are trying catch up to the competition, like they did with the beta testing issue and Testflight integration. No further information about the future development of CI at Apple is communicated to iOS developers by the time of writing. However, it is expected that Apple will improve this topic within their Software environment.

8.2. CI at Pocket Code for iOS

After the practical implementation of this thesis an existing CI system is used in production. Nevertheless, it should be the goal to always improve this process (continuos integration of new Continuous Integration features). Regarding Continuous Delivery, a necessary set of tools is provided that should be automated further as much as possible. New technologies like Xcode's support of parallelizing test runs on multiple simulators could be evaluated and introduced if integrable in Catty's testing environment. Developers are able to examine overall test coverage with a set of tools used by CI to further improve the testing ability. Moreover, the following areas should get addressed particularly.

Runtime

Due to the amount of UI tests in use at Catty, the runtime of integration tests is relatively lengthy in comparison to suggested timeframes by literature. UI tests are reset to the initial condition of the app in every test case via simulated actions on the User Interface. The process of resetting could be verified in a separate test and simulated with program logic in every other run.

Physical devices

Pocket Code is using many device sensors, that are unavailable in iOS simulators. Mockings for sensors in test cases are not entirely assuring correct functionality for un-simulated program execution. Hardware-tests with physical devices are already in use at Catroid and would be a good extension of Catty's CI suite. During an evaluation of this thesis, the CI integration job in use was tested and verified to work as well with hardware devices.

Appendix

Bibliography

Anderson, David J (2010). *Kanban: successful evolutionary change for your technology business*. Blue Hole Press (cit. on p. 36).

Atlassian (2019). URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (visited on 05/07/2019) (cit. on p. 39).

Beck, Kent (Oct. 1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional. ISBN: 9780201616415 (cit. on p. 1).

buddybuild (Jan. 2018). *The buddybuild team is now part of Apple!* URL: https://www.buddybuild.com/blog/buddybuild-is-now-part-of-apple (visited on 05/12/2019) (cit. on p. 63).

Duvall, Paul M., Steve Matyasand, and Andrew Glover (July 2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional. ISBN: 9780321336385 (cit. on pp. 15, 16, 37, 38).

Fowler, Martin and Matthew Foemmel (2006). "Continuous integration." In: URL: https://martinfowler.com/articles/ContinuousIntegration.html (visited on 04/20/2019) (cit. on pp. 1, 16, 39).

Garcia, Cristian Gonzalez et al. (2015). "Swift vs. objective-c: A new programming language." In: *IJIMAI* 3.3, pp. 74–81 (cit. on p. 10).

Habchi, Sarra et al. (2017). "Code Smells in iOS Apps: How do they compare to Android?" In: *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, pp. 110–121 (cit. on p. 42).

Hauser, Dominik (Oct. 2017). *Test-Driven iOS Development with Swift 4 - Third Edition: Write Swift code that is maintainable, flexible, and easily extensible*. Packt Publishing. ISBN: 9781788475709 (cit. on p. 43).

Hilton, Michael et al. (2016). "Usage, costs, and benefits of continuous integration in open-source projects." In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, pp. 426–437 (cit. on p. 18).

Holck, Jesper and Niels Jørgensen (2003). “Continuous integration and quality assurance: A case study of two open source projects.” In: *Australasian Journal of Information Systems* 11.1 (cit. on p. 63).

Humble, Jez and David Farley (July 2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional. ISBN: 9780321601919 (cit. on p. 19).

Jung, Euy-Young, Chulwoo Baek, and Jeong-Dong Lee (2012). “Product survival analysis for the App Store.” In: *Marketing Letters* 23.4, pp. 929–941 (cit. on p. 58).

Krause, Felix (2019). *fastlane is now part of Fabric*. URL: `https://krausefx.com/blog/fastlane-is-now-part-of-fabric` (visited on 05/10/2019) (cit. on p. 31).

Luhana, Kirshan Kumar, Christian Schindler, and Wolfgang Slany (May 2018). “Streamlining mobile app deployment with Jenkins and Fastlane in the case of Catrobat’s pocket code.” In: *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pp. 1–6. DOI: `10.1109/ICIRD.2018.8376296` (cit. on p. 23).

Martin, Robert C. (Aug. 2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. ISBN: 9780132350884 (cit. on p. 42).

Mascheroni, Maximiliano Agustin and Emanuel Irrazabal (2018). “Continuous Testing and Solutions for Testing Problems in Continuous Delivery: A Systematic Literature Review.” In: *Computacion y Sistemas* 22.3 (cit. on p. 45).

Nolan, Godfrey (Dec. 2016). *Agile Swift: Swift Programming Using Agile Tools and Techniques*. Apress. ISBN: 9781484221013 (cit. on p. 45).

Pouclet, Romain (Aug. 2014). *Pro iOS Continuous Integration*. Apress. ISBN: 9781484201251 (cit. on p. 17).

Realm (2019). *SwiftLint Rules*. URL: `https://github.com/realm/SwiftLint/blob/master/Rules.md` (visited on 05/12/2019) (cit. on p. 42).

Rebouças, Marcel et al. (2016). “An empirical study on the usage of the swift programming language.” In: *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*. Vol. 1. IEEE, pp. 634–638 (cit. on p. 11).

Rossi, Chuck et al. (2016). “Continuous deployment of mobile software at facebook (showcase).” In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, pp. 12–23 (cit. on p. 19).

Shahin, Mojtaba, Muhammad Ali Babar, and Liming Zhu (2017). “Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices.” In: *IEEE Access* 5, pp. 3909–3943 (cit. on p. 19).

Slavec, Marc (Apr. 2016). *Integration of controlling Arduino boards via Bluetooth with Pocket Code for iOS using test-driven development* (cit. on p. 43).

Stack Overflow Insights (Apr. 2019). *Stack Ovvrflow Insights Developer Survey 2019*. URL: https://insights.stackoverflow.com/survey/2019/ (visited on 05/05/2019) (cit. on pp. 10, 11).

Statista, eMarketer (Mar. 2019). *Smartphone user share by operating system in the United States from 2014 to 2021*. URL: https://www.statista.com/statistics/201207/us-smartphone-user-share-since-2010-by-os/ (visited on 05/09/2019) (cit. on p. 5).

Statista, MediaTel (Oct. 2015). *Age profile of smartphone users in the United Kingdom (UK) in 2015, by operating system (OS)*. URL: https://www.statista.com/statistics/513988/smartphone-user-age-distribution-by-os/ (visited on 05/09/2019) (cit. on p. 6).

Vasilescu, Bogdan et al. (2014). “Continuous integration in a social-coding world: Empirical evidence from GitHub.” In: *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, pp. 401–405 (cit. on p. 63).

Zheng, Min et al. (2015). “Enpublic apps: Security threats using iOS enterprise and developer certificates.” In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM, pp. 463–474 (cit. on p. 13).

Appendix A.

Acronyms

ASC	App Store Connect
ASO	App Store Optimization
API	Application Programming Interface
CI	Continuous Integration
CD	Continuous Deployment
FOSS	Free/Libre Open Source Software
IDE	Integrated Development Environment
OOP	Object-oriented Programming
PR	Pull Request
SSH	Secure Shell
TDD	Test Driven Development
UI	User Interface
UUID	Universally Unique Identifier
VCS	Version Control System
XML	Extensible Markup Language
XP	Extreme Programming