Dissertation

Deployment of Open Source Multi-language Apps on Google Play

Kirshan Kumar Luhana

Graz, 2018

Institute for Software Technology
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgage Slany
Second reviewer: Prof. Dr. Asadullah Shah

There is never going to be a day where you feel like your thing is ready enough for public consumption so please just publish

(Justin Searls)

Abstract (English)

This thesis is a cumulative thesis, it contributes to the improvement of the Catrobat project in various ways. First, the project is promoted and explained by giving an overview of the projects goals and underpinning philosophy as well as a brief historical review. Second, in place of all Catrobat apps Pocket Paint app was used to test-drive the improvement of the user experience (UX) by fostering internationalization (I18n) and localization(L10n) of the app, by enhancing the interface to support right-to-left languages and automatically test the GUIs validity for all languages. Furthermore, the prototypical redesign of the app to comply with Googles Material Design guidelines and a follow-up usability test to analyze its acceptance were conducted. I18n and L10n not only contribute to an improved UX but also to broaden the apps user base which is also the focus of the third contribution, the enhancement of the transformation of Pocket Code apps to standalone Android application packages (APK). The enhanced version (still beta) is described where the user can transform Pocket Code projects into signed and aligned APKs which are ready for publishing at Google Play. Furthermore, with this beta version, it is now possible to easily include AdMob advertising into the transformed APK. The fourth and most important contribution was the adaptation of the deployment process. It is laid out in detail which steps could be automated to move the process towards continuous deployment. With this approach, it is now possible to deploy Pocket Code with descriptions in all languages including screenshots to Google Play. This deployment strategy was already successfully tested with Pocket Paint, another Catrobat app on Google Play, which shows it can be easily transferred to fit other apps supporting multiple languages.

Abstract (German)

Diese kumulative Arbeit trgt in mehrfacher Hinsicht positiv zum Catrobat Projekt bei. Zuerst wird das Projekt vorgestellt, die Ziele und Ideen des Projekts erklrt, sowie kurz auf die Entstehungsgeschichte eingegangen. Als zweites wurde, stellvertretend fr alle Catrobat Apps, Pocket Paint einer Usability-Verbesserung unterzogen und diese evaluiert. Einerseits wurde die Internationalisierung (I18n) und Lokalisierung (L10) der App verbessert, indem Sprachen untersttzt werden, die von rechts nach links geschrieben werden und die Validitt des Layouts, aller untersttzten Sprachen, automatisiert berprft. Andererseits wurde die Benutzeroberflche auf Google Material Design umgestellt und die Akzeptanz mit einem anschlieenden Usability-Test, mit Nutzern aus der Zielgruppe, besttigt. Internationalisierung und Lokalisierung tragen nicht nur zur Verbesserung der Nutzererfahrung bei, sondern auch um die Nutzerbasis beziehungsweise auch die Reichweite der App zu steigern, was auch der Fokus des dritten Beitrags zum Projekt ist - die Erweiterung der Umwandlung von Pocket Code Projekten in native Android Applikations-Pakete (APK). In der erweiterten Version, die sich noch in der Beta-Testphase befindet, kann ein Benutzer eigene Pocket Code Projekte in signierte APKs umwandeln, die sich fr eine direkte Verffentlichung in Google Play eignen. Optional kann man mit dieser Version, AdMob Werbung in diese APKs inkludieren. Der vierte und wichtigste Beitrag zum Catrobat Projekt ist allerdings die Anpassung des Release Prozesses. Es wird detailliert dargestellt, welche Schritte automatisiert bzw. in der Reihenfolge angepasst wurden, um den Prozess in Richtung voll automatisiertes Deployment (Continuous Deployment/CD) anzupassen. Damit ist es nun automatisch mglich, Pocket Code, komplett mit App-Beschreibung und Screenshots in allen Sprachen, zu erstellen und auf Google Play bereitzustellen. Dieser Release Prozess wurde bereits anhand von Pocket Code und anderen Catrobat Apps auf Google Play, erfolgreich getestet. Das zeigt, dass dieser Ansatz relativ einfach fr andere mehrsprachige Apps bernommen werden kann.

Dedication

To the memory of *Meme* and *Sorath*.

To ama (mother) and baba (father).

To *Preet* (Daughter), *Cheezal* (son), and *Poonam*.

... to all who are doing their bit for making this world a better place to live

Acknowledgement

Today, while presenting my thesis work, I would like to take this opportunity to express gratitude to everybody who direct, indirect support me in this journey towards Ph.D.

First and foremost I owe to University of Sindh, Jamshoro who provided me with the necessary funding for whole PhD. I must express my gratitude to my immediate boss Prof. Muhammad Hayat Keerio for believing in me and providing me with the opportunity to pursue this research work.

Special thanks to Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany for the opportunity and the support throughout my contribution time in the Catrobat organization. It would never have been possible for me to take this work to completion without his incredible support and encouragement. I also wish to extend my thanks to the external reviewer, Prof. Asadullah Shah for his valuable time in review.

I thank Dr. Christian Schindler, for his support and valuable inputs in my research work. Working with Christain was nice experience, his support and dedication was great source of encouragement for me. I also thank Sunny, Gulsher, Azhar, Yaqoob, Matthias Mueller, Thomas Schwengler, Bernadette Spieler, for their valuable inputs to my work.

Many thanks are owed to all my colleagues in TU Graz for their valuable input, help, and teamwork during this work, especially the Jenkins team. My sincere thank also goes to colleagues in Laar College for taking care of all official and unofficial work. Thanks to Zulifqar for sharing office. I extended my thank to the research community for sharing valuable work. Thanks to all co-authors of my research papers.

I gratefully thank all the respected teaching and non-teaching staff from TU Graz. I must acknowledge kind support from TU Welcome Center, HTU Graz on different issues.

I must express my gratitude to Feeroz and family, the kind neighbor at Steyrergasse, Prof. Janusz Mucha, Dr. Anna Małecka, and Kyoko Slany. I will always remember their love and kindness.

I also wish to extend my sincere thanks to friends in Europe for making me feel at home (too

many to list here!). I would also like to thank the Indo Pakistan community in Graz. Thanks to Piyar Ali, Sheeraz Memon, Pinkash, Gulsher, Altaf, Atia, Melita, and 'Krakow Gaddars' for their company in Europe.

I could not have reached so far without the support and the affection of my friends: Bhura, Ramesh, Narain, Rajesh, Sanaullah, Imran, Saeed, Tanveer, Naushad, Aijaz. Their text and calls always made me happy. They always tried best to support me and taking care of my family while I was away.

Last but not least, thanks to all my family members for love, support, and encouragement. Finally thank my parents and children (Preet, Cheezal) for allowing me to study aboard. The last lines of acknowledgment for my lovely wife Poonam. Besides increasing my work, stress, tension, she was always with me like a true friend. She never made me feel alone. Thanks for the support, love, and being mine.

Kirshan Kumar Luhana

Graz, 2009

Publications

Papers, directly or indirectly, included in this Thesis.

1. Awwad, Aiman M. Ayyal, Christian Schindler, Kirshan Kumar Luhana, Zulfiqar Ali, and Bernadette Spieler. “Improving pocket paint usability via material design compliance and internationalization & localization support on application level.” In Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services, p. 99. ACM, 2017.
2. Luhana, Kirshan Kumar, Christian Schindler, and Wolfgang Slany. “Streamlining mobile app deployment with Jenkins and Fastlane in the case of Catrobat’s pocket code.” In Innovative Research and Development (ICIRD), 2018 IEEE International Conference on, pp. 1-6. IEEE, 2018.
3. Luhana, K.K., 2018, May. Pocket code build variants. In Innovative Research and Development (ICIRD), 2018 IEEE International Conference on (pp. 1-6). IEEE.

Accepted Papers :

4. CHRISTIAN SCHINDLER , K. K. L. AND S LANY , W. 2018. Towards continuous deployment of a multi-lingual mobile app. International journal of Engineer & Technology.
5. LUHANA , K. K. 2018a. Building pocket code build-variants. International journal of Engineer & Technology.
6. KIRSHAN KUMAR L UHANA , MATTHIAS MUELLER , C. S. B. S. W. S. 2018. Rock bottom, the world, the sky: Catrobat, an extremely large-scaling and long-term visual coding project relying purely on smartphones. In Proceedings of Constructionism 2018

Statutory Declaration

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 marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Place, Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Ort, Datum Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Software distribution platforms such as Google Play, Apple iTunes, and Amazon provide great opportunities for software developers enabling them to reach out globally for market share. Success of this endeavor depends on many factors one of which is the applications' support for different languages. If applications are not internationalized and localized, they are not used in countries where the apps' languages are not understood. This limits the potential user base significantly [9]. Users expect software to "talk" to them in their language. The reason is not national pride, but a matter of productivity and end-user-friendliness [69]. Many software usually offer localization and internationalization features to attract a larger user base. Albeit, only localization, and internationalization in an application are not enough to attract a larger user base. The user, before selecting an application, is interested in knowing what exactly the application offers in terms of services. For this, it is important to publish the application description in different languages along with meaningful screenshots of the application to encourage users to use the application. Marketing teams also need the latest app screenshots for product promotion. This is the most challenging part: to put the app into the same configuration for all different languages and then create screenshots. With the increasing number of product flavors or software distribution platforms, requiring different formats of app descriptions and images (e.g., the description in XML or text, image file format, resolution) further challenges for the release engineers emerge. These are not only time-consuming tasks, but they are monotonous, hence boring and error-prone, especially if those who must do this work do not understand the language the screenshots have to be created [89]. Furthermore, different tasks are handled by different people or teams, e.g., developers, release engineers, testing teams, teams responsible for setting the metadata, and the authorized person to sign and deploy the application. Each task depends on the other teams' response, which potentially creates further delays. The process of deployment is changing from time to time with advancement in both sides i.e., development of product and production platform. Therefore, the manual deployment pipeline frequently dealing with following challenges [89, 75].

- Dependence on persons with intrinsic deployment knowledge. If the person who usually deploys the app is on vacation - who takes over?
- Usually the documentation of the manual deployment steps is not up to date with the actual process of deployment pipeline which raises the potential for errors in deployment if it is not done by the usual person.
- Since manual deployment usually needs a long-time minor bug fixes are not regarded by the deployment or render the current deployment obsolete depending on the criticality of the bug.

To address these challenges, development processes must adopt various release, deployment, and usage strategies (e.g., product lines and families, self-adaptive systems, configurable or customizable single systems, import & export services, plug-ins) [56]. Furthermore, to deliver software in a timely fashion, it is necessary to cut down on timespan in processes involved in the development lifecycle [19]. Automatic deployment removes all boring and repetitive tasks, significantly reduces release time, and enables developers to release reliably without manual interaction. Automatic deployment steps are written as code and do not require additional documentation. These executable steps can be triggered by any authorized team member [89]. While creating this, developers often spend a great deal of time and budget in building software and tailored scripts to automate setting environment, resource gathering, setting testing environment, and notifying project owners and other teams [75]. Recently, the trend to use third-party tools and libraries is becoming popular within the software industry. Companies can increase focus on real product development rather than developing other tools [139]. Besides, in most cases, it is a cost-efficient solution than developing from scratch. Usually, these tools and libraries are developed to address some particular issues and are developed by different companies or communities, therefore they fail to exchange information. Hence, for portability and interoperability, further development of translators/converters is required.

1.1. Background

Today in the social age [135], it is difficult for most of us, to think of software engineering without open source tools. Many gold standard innovative software systems are developed on free and open source software (FOSS) philosophy or contain at least some FOSS components. [42, 30]. In the commercial software development industry, workers get paid for their work, their efforts are managed, and rights of the created intellectual property are also controlled [86]. Open source software (OSS) projects are developed typically by interconnected communities through shared values and ambitions to create free, reliable, and high-quality software [123, 58]. Participation is usually voluntary, and the project source code is available for anybody to access and modify under certain conditions [114, 86]. The success stories of open source software such as Linux,

Apache, Android, and LibreOffice raised many interesting questions among industry leaders, business managers, researchers. "What drives Free and Open Source software (FOSS) developers to contribute their time and effort to the creation of free software products?" is a question often raised [87] and crucial for assessing the impact of open source software [68]. To explain involvement in open source development, research has identified two primary motivations making contributors to involve in such practices, i.e., Intrinsic Motivation and Extrinsic Motivation [68, 70, 114]. Intrinsic motivation refers to behavior that is driven by internal rewards, doing something for its own sake, enjoying or adoring oneself when taking part in an activity [70], feeling competent and self-determined [68, 70]. Extrinsic motivation is about doing something for sake of some reward [68], in order to achieve some external goals or meet some externally imposed obligations [70, 71]. Despite the fact, according to Raymond, an open source software is a result of "scratching a developer's personal itch" [72, 48, 2, 48]. Nowadays, potential OS projects are sponsored by commercial and public organizations, thus contributors are increasingly getting paid [72]. According to the Linux Kernel report published by the Linux Foundation in 2017 [50], development work is sponsored mainly by Intel, IBM, Samsung, Oracle and most of the developers are paid for Linux-related work [72]. Rewards may have a positive impact on developer's extrinsic motivation but may also leave a negative impact on intrinsic motivation in open source projects. However, Fang et al. [46] in their paper observed that extrinsic rewards do not negatively affect intrinsic motivation. The Linux Kernel 2017 report [50] also stated that "the volume of contributions from unpaid developers has been in slow decline for many years. It was 14.6 percent in the 2012 version of this report but is 8.2 percent this time around.", Bezroukov et. al. [17], also found that contributors to OSS projects are not motivated by money [50]. Working freely is a plausible investment activity in OSS project aiming at learning, reputation, enhanced personal status, and developing the quality of human capital [18]. The open source literature frequently highlights the significance of attracting users and developers [114]. In Open Source communities, hierarchical structure varies from project to project [138, 97]. The members of an open source community based on their interest and skill sets assume roles, responsibilities and contribute their parts. Whereas, in commercial firms, roles and responsibilities are assigned [114, 97]. The members are divided into two groups, i.e., users and developers. The user group identifies and reports the bugs, propose new ideas and translations of applications into their languages. The developer group implements new features, fixes bugs [114], and deal with release and deployment of the software. Project members may have one of the following roles [97]:

1. ***Passive users:*** They are interested only in system functionalities and use OSS in the same way as they are using commercial software.

2. ***Power user:*** Besides using the system, they are curious to understand inside the system, for this usually they review the source code of the system. Reading source code enables them to discover hidden features which might be missed in documents.

3. ***Bug reporter:*** While using the system, if they discover bugs or sometimes they test system in a way to explore bugs. If bugs are discovered, they report it for improvement but usually, they do not fix them.
4. ***Bug fixer:*** They fix bugs discovered by themselves or any reported bug.
5. ***Peripheral developer:*** They are occasional contributors; their involvement is irregular and for short time.
6. ***Active developer:*** They are regularly contributing new features and participating in bugs fixes.
7. ***Core members:*** They are senior members of the project who have been involved for a long time and made a significant contribution to development. Core members coordinates project, responsible for onboarding newcomers and integration of new code. They are also called maintainers in some OSS projects [97, 113].
8. ***Project Leader:*** Project leader is often the initiator of the project. The overall project direction and vision is controlled by the project leader [97].
9. ***Project owners:*** In large open source projects, the control is shared among different stakeholders.

Despite the success of the overall open source movement, many researchers demonstrated high project abandonment rates due to lack of newcomers, sustained participation, and overall survival for long-term success [22]. Because of the independent governance of each community and scattered members for such communities, it is very difficult to estimate numbers of paid and unpaid contributors. Nevertheless, it is presumed that number of free, unpaid volunteers are more than paid contributors and consequently, it is implausible that they work on boring, monotonous tasks, or take orders [73]. According to the GitHub open source survey 2017 [140], responsive maintainers, a welcoming community, active development, contributing guides, code of conduct, stability, user experience, and transparency are important values for open source projects. Agile project management and development methods have been widely practiced and gain much popularity in recent years [116, 41]. Agile respects the interest of all stakeholder in a project, be it the project manager, tester, developer or client. All stakeholders get a visible picture of the project. It encourages active user participation throughout the project lifecycle [6, 134]. Other benefits of an agile approach are quick and flexible responses to change, adaptive planning, evolutionary development, earlier delivery, continual improvement, shorter time-to-market, increased customer satisfaction, and high-quality software [116, 43, 2]. One common hypothesis is that an agile [134, 13] development process is similar to the open source software process; however, this hypothesis has not been validated so far [138]. One possible reason can be that due to the nature of the different OSS projects have different project specific process models are in use. In recent years, adoption of agile methods is visible with the popularity of agile processes. The Sauce

Labs reported as “The agile development movement continues to grow steadily year over year with its promise to cut development time and costs and yield better results for all stakeholders. In our survey conducted in 2015, 82% of software developers said that their organization had adopted an agile development methodology. This number grew to 91% in 2018.” in their “Testing Trends for 2018” report [82], which is also published on Statista portal [101] see figure 1.1. According to Fugetta, “[Agile methods] can be equally applied to proprietary and open source software” [54, 30]. Thus, one can relate adoption of agile methods in open source project [30].

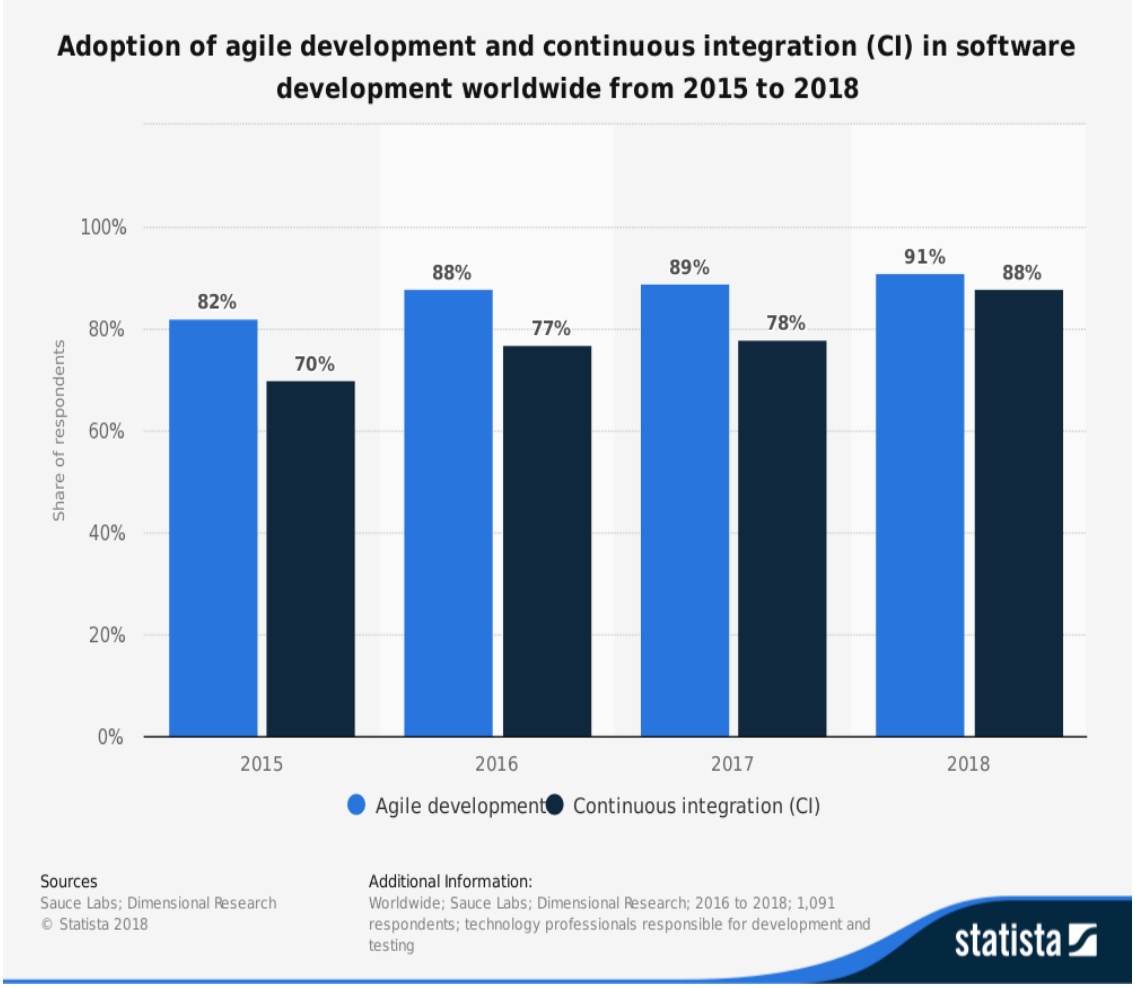


Figure 1.1.: Agile and CI adoption 2015 to 2018 [101]

In a project, different developers work on different components. Each component is developed independently. Combining these components into a system and determining how newly added components work together with other components of the system is called integration [20]. The

integration is time consuming and difficult task because the components often do not work together as intended. Therefore, many project schedule integrations after reaching a milestone or on a specific date of a month. “Waiting until the end of a project to integrate leads to all sorts of software quality problems, which are costly and often lead to project delays. Continuous integration (CI) addresses these risks faster and in smaller increments.” [39, 20]. Continuous Integration (CI) is a promising trend in automated software engineering [30] . The term continuous integration (CI) was introduced by Martin Fowler [57] as a representative practice of agile methods like Extreme Programming [53, 30]. It advocates the practice of frequent integration of newly written code to a centralized repository [26], automatic builds and tests of different granularity to maintain a codebase which is always ready to be deployed [117]. This helps teams to improve integration and find bugs quickly while reducing the risk [57] of a last-minute release hiatus or even a release cancellation. To achieve all this newly written code with the potential to be integrated into the code base must be built and tested automatically to ensure system stability. There are many tools available supporting automatic builds, testing, and packaging of new versions with or without human intervention [95]. If failures occur during the build process the tools can provide detailed information about the failure to code-contributors and integrators. Furthermore, they can be configured to generate different reports for different stakeholders. This is helpful for integrators not to waste time on reviewing unmergeable code and to deal with the social challenge of telling the code-contributor about the rejection. The project managers might want the newly integrated work in production as soon as possible for early feedback from clients. Modern CI servers are not only used for building and testing but are extended to automatically deliver and even deploy successfully built and tested software. However, it requires changes in test habits and the development workflow [26]. Continuous delivery (CDe) practice is one step further than continuous integration. In [29] the author found that from an organization’s point of view continuous delivery is rated the second most important agile software development practice. CDe enables teams to keep their software production ready and releasable on demand by employing continuous integration, automated testing and quality checks [28, 74, 117, 24] for their internal or external customers. Continuous delivery creates a deployable product leaving the decision on management to deploy or not deploy. This is useful in an application domain where the rate of frequent deployment is slow. Whereas, in the continuous deployment, the final step of pushing the product into production is also automatic [51]. Jez Humble visually explains CDe in an easy to grasp way [1] which is depicted in Figure 1.2

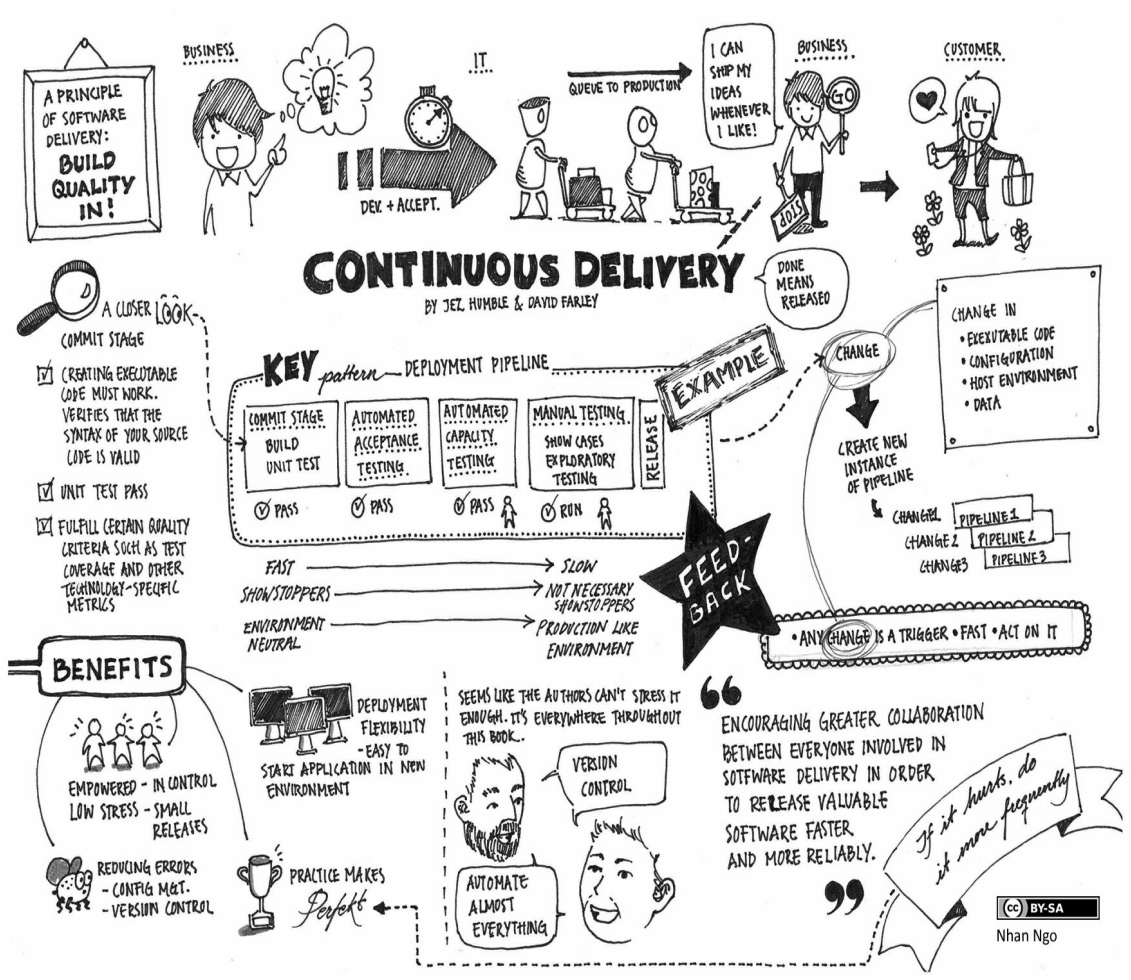


Figure 1.2.: Visual explanation of Continuous Delivery, visualizations by Nhan Ngo [1].

The state between “ready to release” and “released to production” reduces product value and delays user feedback. Furthermore, every time CDe builds “ready to release” binaries, they could be different from manually tested binaries. This creates frustration among testers and managers since that “ready to release” product passed all tests but failed to be put into production. This problem can be tackled by introducing continuous deployment (CD) [25]. Continuous deployment (CD) is the ultimate goal. It is important to release a new version of the project as soon as bugs are fixed, or suggested improvements have been made. Such continuous developments will persuade customers to use the software while lockout the competitors. The CD practice is to deploy every change in the codebase directly, so it is used in production. Frequent delivery impacts various stakeholders, including sales, marketing, operations, and customers [24]. Frequent deployment is a result of a shorter release cycle. Each short cycle introduces only a limited

volume of code changes. This reduces risk and helps diagnose problems, fix them timely and cost-efficiently [24, 47] and fosters quick feedback since the changes are clear and oversee able. Due to this shorter feedback loops development of unnecessary components is kept to a lower extent [24, 26]. Companies are migrating to agile development process in hopes that they will be able to deliver their product in high quality, quickly, efficiently, and reliably to the market [24, 19]. Nevertheless, most of them are experiencing deployment activities as unnecessarily cumbersome and frustrating [19]. Promoters of continuous deployment claim that CD can increase the focus on product development, automate repetitive tasks, and increase product quality by comprehensive automatic tests, integrated teams and processes with a unified pipeline to create a workflow across all development phases, i.e., development, testing and production. The purpose is to , harmonize the existing deployment toolchain and boost the overall productivity [14]. However, fully implementing CD is a very challenging task [26, 19, 25, 24, 67, 118]. Large-scale, complex projects are often deployed on several different platforms during their journey towards deployment in production. The manual setup of the build and test environment and the preparation for deployment can take days. The application domain of a product is an important factor for continuous deployment. For instance, deployment of a mobile or desktop application is different from a web or cloud application. Web or cloud applications can be easily rolled-back if any bugs are detected. Whereas, for desktop or mobile application, the decision to down- or upgrade is in the users hands [112] therefore, researchers and practitioners strive to gain deeper understanding and improve the CD process for mobile applications. “Continuous deployment is among the most difficult agile techniques to employ successfully and requires a very high level of agile maturity and discipline in the team” [19] and often needs focused continuous engineering research. Yet, very less attention has been devoted to the issues in the adoption of continuous deployment in open source [108]. Research on the deployment of product on third-party distribution platforms and publishing localized versions of deliverables (i.e., product description and UI screenshots in a different language) is still ongoing research.

1.2. Motivation of The Thesis

The motivation for the work presented in this thesis is based on contribution to the Catrobat[1] project. The Catrobat project was initiated in 2010 by Prof. Slany at the Institute of Software Technology at Graz University of Technology. It is an independent, free and open source software (FOSS) project which is hosted on GitHub. The Catrobat project is a modern, active, and large open source project [89, 103].

[1] https://catrobat.org

1.2.1. Catrobat KPI

According to Key Performance Indicator (KPI) for open source projects, an open source project is successful if it is adopted by users and possibly further developed by some of them. User code contribution is considered the main KPI metric for OSS projects [104, 132].

Adoption Catrobat's Pocket Code is available on Google Play for free to use. The project received more than 4500 (5 stars), 500,000+ downloads. The app is adopted by different educational institutes and is used in courses to introduce programming. Users of the Pocket Code are actively sharing their work on the "community" website[2] and post video tutorials on YouTube. In Figure 1.3 the rating trend of Pocket Code is depicted.

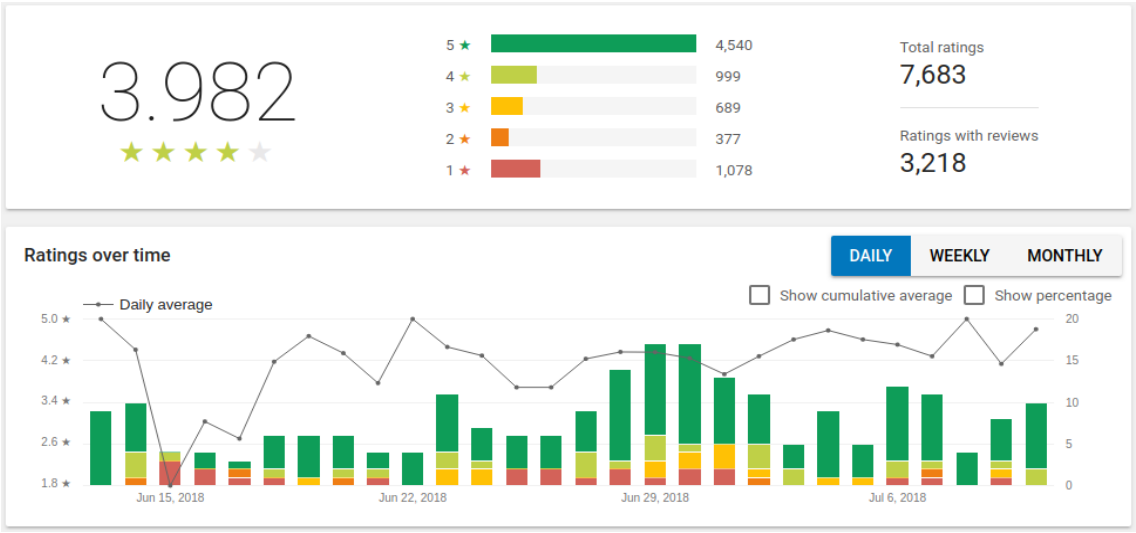


Figure 1.3.: Pocket Code's star rating trends in the projects Google Play console.

[2] https://share.catrob.at/pocketcode/

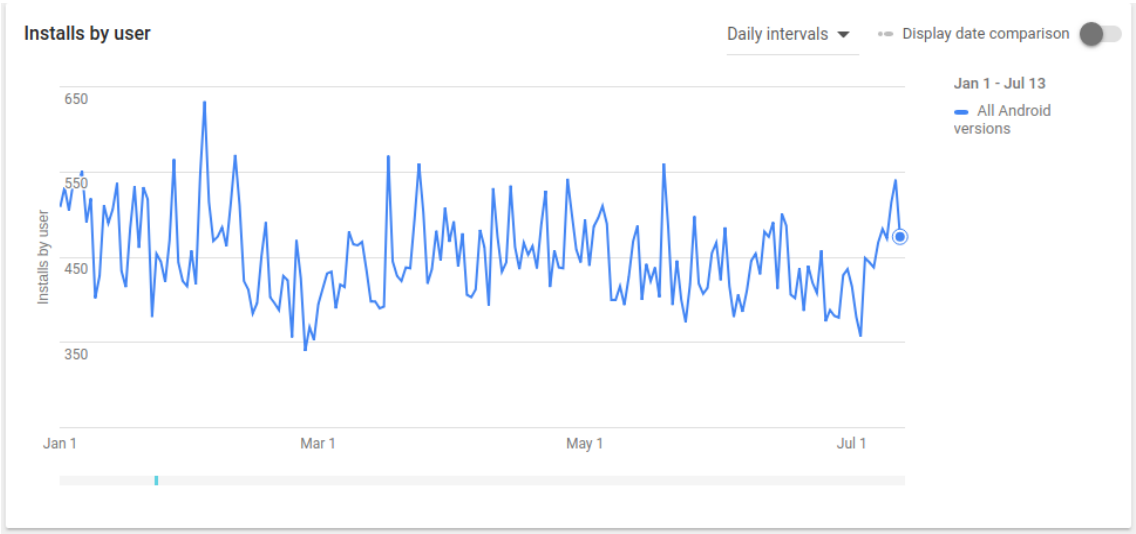


Figure 1.4.: Pocket Code installations from January 1st, 2018 to July 13th, 2018 in the projects Google Play console.

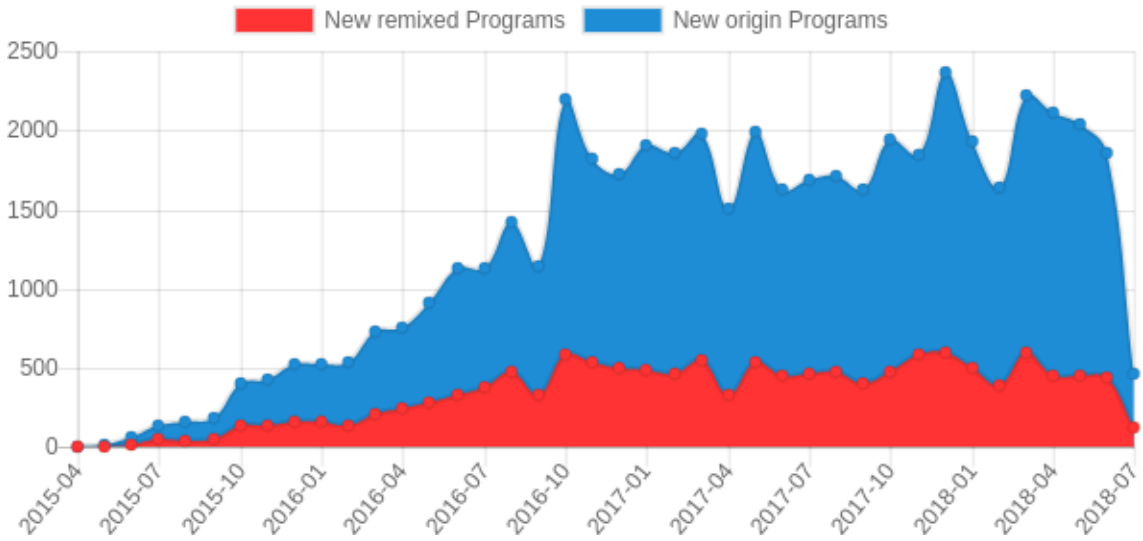


Figure 1.5.: Number of Pocket Code projects shared on the community website (new created and remixed).

Contribution From more than 20 countries more than 500 volunteers are contributing to the design, development, and translation of the Catrobat tools. Data generated by gitstats on 13th July 2018 [3], shows total commits 10765 (average 5.9 commits per active day, 3.7 per all days) by 219 (average 49.2 commits per author) in the age of 2899 days, 1816 active days (62.64%). Total number of pull requests (PRs) issued: 2445. Out of the 2445 PRs, 2431 were closed and 14 are still open. Out of the 2431 closed PRs, 1832 were merged. This data proves that the Catrobat project is actively developed and users are interested in contributing frequently.

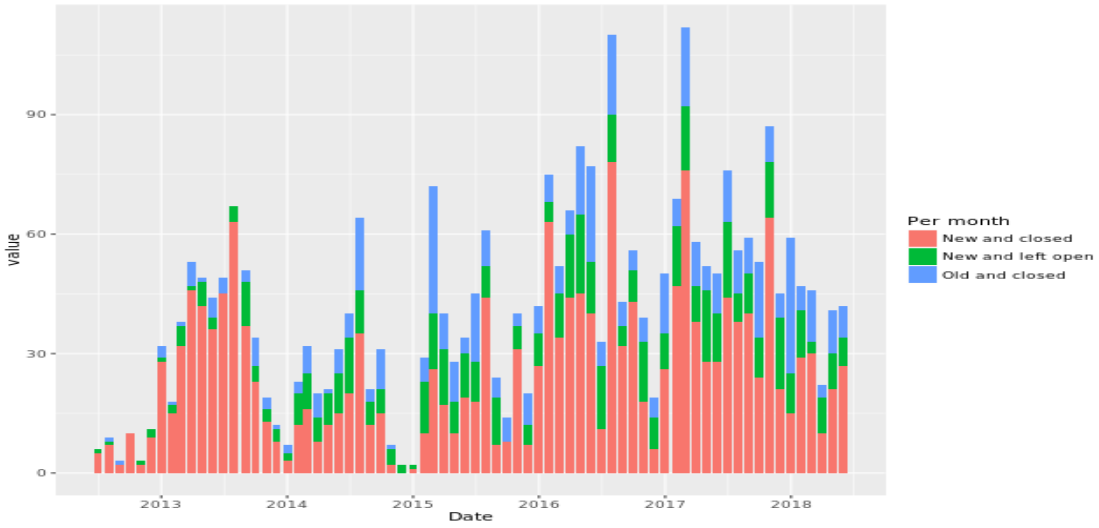


Figure 1.6.: Pocket Codes pull request backlog.

"The pull request history shows the number of pull requests processed per month. Even though a month is a relatively coarse-grained period for pull requests (where review and acceptance/rejection happen very fast), this view can be helpful to get an idea of the overall activity within the project." [59]

[3] http://gitstats.sourceforge.net

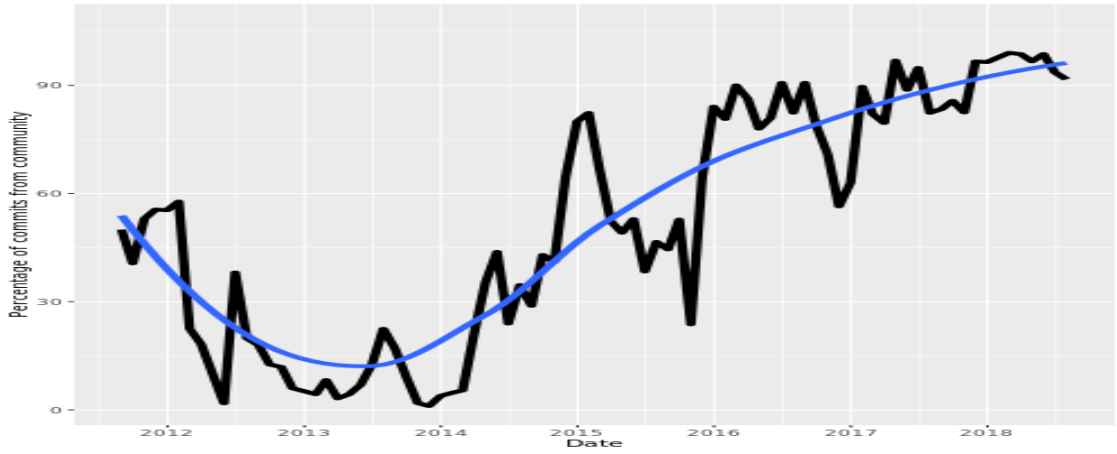


Figure 1.7.: Pocket Code GitHub Commits from the project community as percentage of total

"Percentage of total commits (and trendline) coming from the community. The more commits coming from the community, the more this project is a community effort." [59].

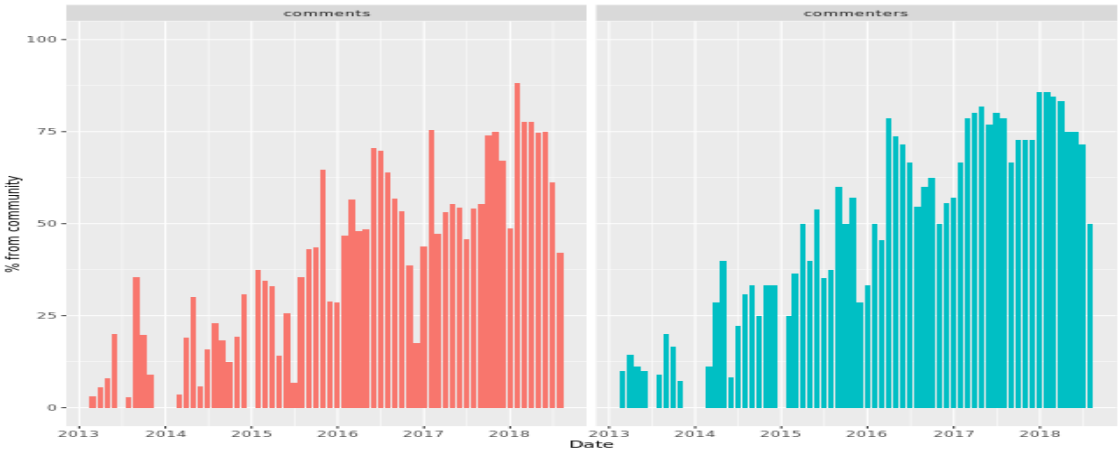


Figure 1.8.: Pocket Code GitHub Comments and commenters from the community.

"Percentage of comments (left) and people that commented (right) coming from outside the project's core development team. The more comments coming from the community, the more welcoming the project is to outsiders." [59]

Awareness The Catrobat community is not only active in development but it is also active in awareness camping. It maintains a website (https://www.catrobat.org) for general information about the project, in addition, separate websites for different sub-projects, for example, create @school (https://edu.catrob.at). Additionally, several volunteers are engaged in a social media campaign on Facebook, tweeter, writing blog posts and posting videos on YouTube sharing their experiences.

1.2.2. Catrobat

The Catrobat is a set of creative tools and a visual block-based programming language for different platforms, designed internationalization (i18n) and localization (l10n) principals. The focus of the project is the development of Pocket Code for Android and iOS [89]. The Android version is released in various flavors for different partners and projects (e.g., Create@School, or Phiro) [88]. Development is done Test-Driven [90] wherever possible and applicable along with other agile methods such as Ping-Pong-Pair-Programming [12], Collective Code Ownership, Continuous Integration [53], Robert C. Martin's Clean Code paradigm [92] and Simple Design. The intention behind this is to ensure a clear, concise, plausible , and 100% up-to-date documentation as well as high quality code [103]. Working in the Catrobat project is potentially beneficial for the contributors in many regards. The social and technical dynamics of working on a rather large-scale project, having access to a vast code base, using industry-standard tools in terms of configuration- and build management, true teamwork, and project management are profitable skills they are experiencing. The open source philosophy is also reflected in Catrobat's services for its end-users. Catrobat promotes the idea that coding is no magic and can be learned easy. It fosters computational thinking, facilitates learning to code and inspires the users to express themselves creatively through coding. Furthermore, it familiarizes the users with the free open source philosophy in a fun and engaging way on a worldwide scale [103].

1.3. Challenges and potential solution approaches

The study aims to streamline deployment in the Catrobat project. The research objective is subdivided into following research questions with elaborations, and a short summary of contribution.

1.3.1. Research Question 1

RQ1: What are the aims and objective of the Catrobat project and Pocket Code app?

Motivation

The first research question (RQ1) addresses explorative research on the Catrobat project to learn about targeted user-groups, available features, and a future development plan. The philosophy behind Catrobat is explained and new features are proposed and discussed.

Contribution

We conducted an exploratory study (see Chapter 2) on Catrobat. The study started by exploring the history of Pocket app up to its interesting upcoming features.

1.3.2. Research Question 2

RQ2: How internationalization (I18n) and localization (L10n) features are implemented and what is their impact on end-user?

Motivation

The second research question (RQ2) addresses the impact of the internationalization (I18n) and localization (L10n) of a mobile application on the user experience (UX). The intricacies of implementing I18n/L10n in general and especially on application level are investigated. This study is helpful for developers who wants to publish apps with I18n/L10n support. It is explained how the graphical user interface (GUI) can be tested for validity for all languages and why this is beneficial for broadening the user base.

Contribution

The purpose was to improve the quality and usability of a mobile app in the case of Pocket Paint, a paint editor that, among others, allows setting parts of pictures to transparent and zooming up to pixel level and is integrated into Pocket Code. The study laid out the requirements to thoroughly support internationalization, localization and bi-directional scripts. The study discussed issues in improving user experience and usability by implementing Googles Material Design and application-level I18n/L10n support. Furthermore, it showed how GUI validity could be automatically tested by parameterized Espresso test for all supported languages. Finally, a conducted UX test with a small set of typical users of the app hinted that the undertaken implementation efforts for UX improvement were positively perceived.

1.3.3. Research Question 3

RQ3: How to empower end-users by allowing them to release their Pocket Code projects on app stores?

Motivation

When users create games, animations, or any other applications in Pocket Code they can share their projects on the Catrobat sharing platform. Pocket Code projects must comply to the Affero General Public License (AGPL)[4] licensed. This license fosters learning by remixing and sharing of projects with the community [103]. These projects are platform-independent but need a runtime-environment such as the Android or iOS version of Pocket Code, or the Pocket Code HTML5 player which runs itself in any HTML5 compatible browser. Pocket Code projects on the sharing platform can be converted to Android application packages (APKs) [88]. A Pocket Code project as an APK file can be installed and executed on Android devices without the need of a Pocket Code installation. Each converted project contains its own unique package name, app title, and icon [88]. However, these projects can be shared as standalone app in APK format but cannot be released on Google Play store. The current project converter setup does not allow a user to change the app properties (version code, version name, app icon, title). This converter limitation restricts a user to release the project as a debug version which cannot be published via Google Play.

Contribution

We developed a new Pocket Code project converter feature (see Chapter 4), which is yet not public. The new feature allows registered users to sign and release the Pocket Code project as an app on Google Play. Users optionally can add in-app advertisement to earn money which is made possible by integrating AdMob on demand. Credentials for services such as AdMob account details must not be stored in the source code, especially in open source software [88]. However, the AdMob credentials need to be in source code while building the project binaries. The demand for potential change in the source code is a very expensive and risky solution [74]. For this, the mechanism is explained that allows users to inject such sensitive data by triggering parameterized builds.

1.3.4. Research Question 4

RQ4: How to streamline Catrobat's deployment pipeline.

[4] https://www.gnu.org/licenses/agpl-3.0.en.html

Motivation

The fourth research question (RQ4) addresses the streamlining and optimization of the tedious and error-prone deployment tasks for the Pocket Code app. Intellectual unchallenging, repetitive tasks are error prone, no matter how critical, humans get bored and start to make mistakes. Human factor researchers are increasingly concerned with developing tools for handling critical acts [109]. Automation helps in this regard and improves performance, reliability, availability, and productivity and hence it saves time and money. Catrobat's deployment process for Pocket Code poses no exception. With a vast number of supported languages, deployment for Google Play gets tedious, repetitive and error prone since for every supported language the description and screenshots must be created and uploaded to properly reflect the apps multi-linguality in the app store.

Contribution

The Pocket Code deployment process was adapted towards continuous deployment (CD). It is described which steps in the deployment pipeline were automated and which were reordered or postponed for that an automatic deployment to Google Play was made possible. It is laid out which tools were used, and which adaptations had to be done to fulfill the requirements to be able to deploy to Google Play automatically. The presented approach speeds up deployment of multi-language apps significantly since it automates the creation of the screenshots for all the supported languages. This deployment strategy was already successfully tested with Pocket Paint, another Catrobat app on Google Play, which shows it can be easily transferred to fit other apps supporting multiple languages.

1.4. Origin of Chapters

Each of the chapters in the thesis is peer-reviewed. Chapters 2 to 5 were peer-reviewed and published in software engineering conference proceedings.

1.5. Thesis Outlines

The thesis is structured as follows. Chapter 2 explores and evaluates the Catrobat project. Chapter 3 studies the usability of multi-language features in the Catrobat project. Chapter 4 discusses the Pocket Code software product line in particular the building of the standalone version of the app. Chapter 5 explains in detail the complete Catrobat delivery pipeline. Finally, Chapter 6 recapitulates the thesis contributions.

Chapter 2

Rock bottom, the world, the sky: Catrobat, an extremely large-scaling and long-term visual coding project relying purely on smartphones

This chapter is accepted for publication in Proceedings of Constructionism 2018. [80].

2.1. Abstract

Most of the 600 million teenagers everywhere in the world already have their own smartphones, but comparatively only very few of them have access to PCs, laptops, OLPCs, Chromebooks, or tablets. The free open source non-profit project Catrobat allows users to create and publish their own apps using only their smartphones. Initiated in 2010, with a first public version of our free app published in 2014, with 46 releases of the main coding app as of April 2018, our app currently has more than 700,000 users in 180 countries, is natively available in 40+ languages, and has been developed so far by almost 1,000 volunteers from around the world. The interface is strongly inspired by Scratch, very intuitive, has lots of accessibility settings, e.g., for kids with visual or cognitive impairments, has tons of constructionist tutorials in many languages, and has a plethora of extensions, e.g., for various robots, including drones, as well as for controlling arbitrary external devices through Arduino or Raspberry Pi boards. A TurtleStitch extension allowing to create one's own embroidery patterns for clothes is being developed in cooperation with the www.turtlestitch.org project. Our project among others focuses on including female teenagers. While a dedicated version for schools is being developed, our apps are meant to be primarily used outside of formal education settings, and are thus mainly discovered by our users through various app stores such as Google Play and via social media recommendations as well

as our presence on Code.org. Sharing and remixing is actively encouraged, but we also allow teenagers to compile and publish their apps commercially to earn money with their apps through sales and/or ads. All apps provided by Catrobat itself will however always remain ads-free and free of costs. The project has a very long term perspective in that it is independent of continuous funding and actively developed in a test-driven way by hundreds of mostly pro-bono volunteers from around the world.

2.2. Keywords

Pocket Code, Game Design, Gaming, Gender Inclusion, Coding, Mobile Learning, Social Inclusion, Constructionism, Girls, Teenagers, Apps, Smartphones, Tinkering

2.3. Introduction: Background, mission, and history

Knowledge in Computer Science (CS) is essential, and industries have increased their demand for professionals that have technical experience. The next generation of jobs will be characterized by new standards requiring employees with computational and problem solving skills in all areas, even if they are not actual technicians [5] . However, the number of young people, and women in particular, choosing to study and work in Information and Communication Technology (ICT) fields is decreasing dramatically [99, 100]. In the last decade, European technology employment has grown three times faster than all employment in total. The continuous improvements of technology and the numerous advancements in industrial processes made it possible to develop autonomous vehicles, robotics, 3D printing, genetic diagnostics, or Internet of Things (IoT) technologies. These technologies are already part of everyday life, and there is a corresponding growing worldwide need for qualified scientists, engineers, and technicians. For these reasons, society and governments have mandated that teenagers should acquire computing and coding skills [27], or even a new way of (critical) thinking and problem solving skills [136, 78, 128, 91]. Presenting coding as a range of diverse skills which can be learnt by adapting gaming concepts is a generally applied concept. Thus, a gamified and constructionist concept should hold teenagers' focuses to actively participate by activating intrinsic and extrinsic motivators (Ryan and Deci, 2000). Games can be played everywhere, even on smartphones, tablets, and other digital devices. Moreover, the mobile game market continues to grow faster than other game industries, e.g., the number of game apps on Google Play grew by 28% in 2017 [102, 124].

Catrobat's approach is inspired by Piaget's Constructivism theory 1948 [107], starting with first computer programming courses at the MIT in 1962 [66], and refined with Papert's Constructionism concept in 1980 [105]. Since then, different approaches were used to motivate kids

for coding. With our free open source non-profit project Catrobat our goal is to provide computational thinking skills for everyone, especially teens from less developed areas where other computational devices such as PCs are almost non-existent.

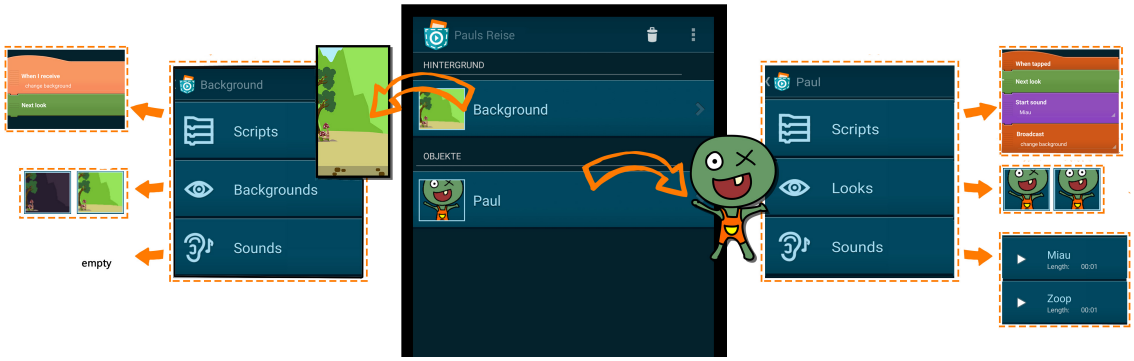


Figure 2.1.: Pocket Code's UI

Catrobat's apps and services have been immensely influenced by MIT's Scratch[1] project, and we consider Catrobat to be Scratch's little sister project for smartphones. Scratch itself has been strongly shaped by Papert's powerful ideas, and extends Papert's "Low Floor" but "High Ceiling" for the Logo programming language for kids (i.e., easy to start, but allowing to develop more complex projects as well) by adding "Wide Walls", emphasizing that Scratch supports a wide variety of projects as well as ways to learn and play, according to the needs and interests of its users [110]. Following the metaphor of the room, we have extended it to Catrobat's mission statement, "Rock Bottom, the World, the Sky". "Rock Bottom" because on the one hand, we aim at building upon Scratch's focus on making the first experiences in coding as easy and satisfying as absolutely possible for our teenage user group, though we must note that this is really still a guiding principle, as there is yet ample room for improvement. On the other hand, because of our reliance on smartphones, it also is meant to evocate that our users can and to a large percentage do leave the "room" to code outside or create outdoor projects, in some cases literally "on the rock". Many of our users are indeed developing their projects while on the go, far from classrooms and their homes, and have developed apps that take advantage of the various sensors such as the cameras, GPS, compass, or acceleration sensors that are built into smartphones and that allow to create, e.g., augmented reality, geocaching, or dynamic outdoor sports games. Additionally, today's teenagers all over the globe have, to an already very large degree and also increasingly, their own smartphones readily available and permanently connected to the Internet, even in rural areas in Africa and other regions in the world where there may be no central power supply at

[1] Scratch MIT: https://scratch.mit.edu/

home and kids charge their phones via solar panels in a central community facility of their village. Catrobat also has begun to become available in languages that are not supported by the phones makers themselves, such as Swahili, Gujarati, or Sindhi. Catrobat thus strives at reaching out to all corners of "the World" in an effective and efficient way that is indefinitely sustainable. The reliance on already existing smartphones, as well as the free open source character of Catrobat, which lets it thrive with little to no funding, also are significant aspects that allow Catrobat to scale up by avoiding the high costs and logistics that have hampered the success of similarly motivated projects in the past. Regarding the third part of our mission statement, Catrobat allows you already since 2017 to program drones flying autonomously in "the Sky" (in particular, the popular Parrot Augmented Reality Drone 2.0), with a real-time video being transmitted to Pocket Code's screen, under full programming control by the user. While coding with Pocket Code has been done in airplanes, including during take-off and landing, for the future we plan on going even higher. Because phones are small, lightweight, and portable, off-the-shelf Android phones have already been used to control balloons up to the stratosphere, and our extensions via Bluetooth and local WiFi connections to battery powered Arduino and Raspberry Pi allow to control any hardware of these flying computational systems, as long as the isolation keeps the harsh environment at bay and there's enough energy. On a further note, PTScientists' private enterprise Moon rover mission project[2], scheduled to lift off in 2019 and sponsored by, among others, Vodafone, Nokia Bell, Audi, and Red Bull, has several experiments on-board that rely on regular Android phones to lower the costs of otherwise extremely expensive "rocket science" hardware, with Vodafone and Nokia Bell sponsoring the installation of a 4G data network based on standard phone transmission technology from the rovers to the base station on the Moon and from there back to Earth. One of our dreams is that we will empower kids to use Pocket Code to design experiments and to program autonomous robots on the Moon, and possibly even farther away. The sky has no limit, both in the concrete as well as in the metaphorical sense. Regarding the latter, our goal is to allow every kid to create complex, high resolution, and high performance apps using our tools, which they will be able to offer to other users, even commercially.

On a historical note, one of the initial motivations for starting the Catrobat stems from Neal Stephenson's science fiction book "The Diamond Age: Or, A Young Lady's Illustrated Primer: a Propdeutic Enchiridion in which is told the tale of Princess Nell and her various friends, kin, associates, &c.". In the book, human tutors are hired anonymously on demand by an infinitely affluent industrialist and aristocrat, to remotely educate a very young girl living under gruesome conditions, by mistaking her, because of circumstances described in the book, for a princess for whom the primer was actually created. The illustrated primer, which is highly portable and has a voice- and touch sensitive interface that allows her to communicate with her tutors, accompanies Nell throughout her adolescence up to young adulthood, when she becomes a leading force and changes the fate of millions of the most underprivileged kids on Earth. One of the main story

[2] http://ptscientists.com/

lines spanning a large part of the book is the acquisition of computational thinking skills by Nell through the Illustrated Primer. The book has won the Hugo and Locus science fiction awards, and was also cited by the developers of the One Note per Child Project as well as of Amazon's Kindle as a motivational inspiration. In 2017, Catrobat won the "Closing the Gender Gap" prize for a new subproject called "Remote Mentor", in which we have started to implement the necessary technical infrastructure and study the required social aspects to realize Stephenson's Illustrated Primer. We have partnered with sociology scientists focusing on gender aspects for the research part. In November 2017 we have begun to conduct initial remote mentoring experiments under real conditions, first with sponsoring from Google as a Google Code-in mentoring organization at the end of 2017, with several hundreds of teenagers from all over the world being remotely mentored by a pool of 38 Catrobat mentors over a period of 8 weeks, followed by the "Remote Mentor" project itself, with initial funding from the Internet Foundation Austria until the end of 2018. Because of Stephenson's book, Catrobat has focused from its very beginnings to aim at empowering female teenagers in less affluent regions such as rural areas in India, Tanzania, or Brazil. Teens have reacted very positively to these first remote mentoring experiments, and we plan to eventually integrate the remote mentoring features into our tools and services so that mentors and mentees will be anonymously and automatically matched on demand, internationally in all languages, in a large scale, long term, and continuously further developed way.

This paper is organized the following: First, we emphasize two trends that have emerged in the last decade (both were important in developing our app Pocket Code): block-based and visual coding, and an increased use of mobile devices among our target group (teenagers from 13 to 19 years). Second, the focus lies on the Catrobat project and the educational app Pocket Code. Third, we describe our planned next steps and subprojects that are being developed, followed by a summary and conclusion.

2.4. Computational Thinking Skills for All

Computational Thinking promotes the importance of coding and computer science activities, thus delivering concepts that are more applicable and highly essential to prepare teenagers for the future [125, 128]. After 2006, there was a rapid increase in the number of published articles about learning to code [137]. The ongoing movement of promoting coding through visual programming languages has its origin at that time.

Trend 1: Block-based and Visual Coding

In the last decade, a number of block-based visual programming tools, e.g., Scratch, have been introduced which should help teenagers to have an easier time when first practicing programming. These tools have all had very similar goals: they focus on younger learners, support novices in

their first programming steps, they can be used in informal learning situations, and provide a visual/block-based programming language which allows teenagers to recognize blocks instead of recalling syntax [129]. Unlike traditional programming languages, which require code statements and complex syntax rules, here graphical programming blocks are used that automatically snap together like Lego blocks when they make syntactical sense [84].

Another important differentiator is the fact that all elements of the programming environment and also the programming language itself, including the formula elements, are translated to the language of the young developer. Especially for languages that are not written with the Latin alphabet, this is a huge advantage for users, as they are not used to think in English and have very commonly difficulties to read Latin scripts. In case of underdeveloped and developing countries, only a very small percentage of the population understands English, in which most user interfaces are available. Localization of a software can revolutionize E-learning, resulting in more educated workforce and improved economy [60]. Pocket Code supports localization and internationalization on the application level. The app's language and locale can be changed without changing the system locale. Languages such as Sindhi and Pashto, which are yet to be supported by operating systems, can thus be seamlessly used by our users [7]. Catrobat shares this features, which improves accessibility and inclusiveness to users from all regions of the world, with Logo, Scratch, and Snap!, and certainly contributes in a major way to the positive worldwide reception of these programming languages.

Thus, visual programming languages provide an easier start and a more engaging experience for teenagers. The ease of use, and simplicity of such programming environments enables young people to become game makers.

Trend 2: Smartphone Usage

With mobile games, more people can engage who were previously limited to use other platforms such as PCs or consoles. Further, children nowadays grow up with mobile devices and feel comfortable using them. Considering current prices, and the forecast of the user penetration of smartphones in Austria, France, Germany, and the United Kingdom from 2014 to 2021 (Statista Market Analytics, 2016), we can conclude that smartphones will be used more by teenagers in the future than the more expensive tablets or laptops. Smartphones and the use of apps are already a part of our culture and are changing the way in which many people, particularly teenagers, act in social situations. For most adolescents the smartphone performs several functions of their daily lives and it contributes, e.g., to identity formation through self-presentation on the Internet. In addition, the smartphone is used a lot in spare time (most games are played in the evening (Verto Analytics, 2015)) or for just killing some time while waiting. In addition, online games and mobile games play an important role in the daily lives of teenagers [16]. Here, a recent study

which examined American female players' experiences analyzed that 65% of the Android users who play mobile games consists of women [65].

Teenagers increasingly have mobile devices on their own, which enables them at any time to creatively express themselves and to use apps that bring their ideas and creations to life. With a more meaningful use of mobile devices, teenagers worldwide can acquire powerful knowledge that will make them into better problem solvers, thinkers, and learners.

2.5. Catrobat and the Pocket Code App

The Free Open Source Software (FOSS) non-profit project Catrobat[3] was initiated 2010 in Austria at Graz University of Technology. The multidisciplinary team develops free educational apps for teenagers and programming novices. The aim is to introduce young people to the world of coding [120]. With a playful approach, teenagers of all genders can be engaged, and game development can be promoted with a focus on design and creativity. A first public version of our free app was published in 2014, with 46 releases of the main coding app as of April 2018. Our app currently has more than 700,000 users in 180 countries, is natively available in 40+ languages (including several languages not directly supported by the underlying operating system), and has been developed so far by almost 1,000 volunteers from around the world.

These volunteers are implementing software, designing educational resources, translating the app, or provide other services that help to advance the project. The contributors work together in a cooperative way, having the chance to engage in a field they like and create something within a community that follows the same shared vision. Besides attracting students, educators, and other interested contributors from all over the world, Catrobat also benefited from being part of Google's Summer of Code and Code-In initiatives. These initiatives promote open source worldwide and motivate teenagers and students to get involved in projects such as ours. Our openness towards international contributors helps us to represent different cultures, bring in various viewpoints, and generate new ideas how the project can further develop. Catrobat's project management as well as development is done in an agile way, allowing our contributors to adopt new technologies, respond to user feedback, and embrace upcoming ideas quickly. An example where a feature request issued by users was implemented, is our web based automatic APK (Android Package) generation. It is currently being extended for users to be able to sign their apps and add AdMobs based ads, so that they can publish their Catrobat projects on Google Play and other app stores and earn money with them. Another example is of technical nature and affects our deployment workflow, which was redesigned and fully automated to enable us to deploy to Google Play including the localized screenshots and app descriptions in 47 languages with only

[3] https://www.catrobat.org

two mouse clicks. All this helps to provide a motivating user experience for our young target group, support them in their learning process, and foster collaboration.

2.6. Pocket Code: Creating your own Games

The app Pocket Code[4] is an Android-based visual programming language environment that allows the creation of games, stories, animations, and many types of other apps directly on phones or tablets, thereby teaching fundamental programming skills. This app consists of a visual Integrated Development Environment (IDE) and a programming language interpreter for the visual Catrobat Programming language. The IDE automatically translates the underlying code parsed by the XML file into visual brick elements and vice versa. With the use of simple graphic blocks, users can create their own game, colorful animations, or extensive stories directly on the mobile phone without prior knowledge.

The drag and drop interface provides a variety of bricks that can be joined together to develop fully fledged programs. The app is freely available for Android on Google's Play Store and soon will be available on Apple's iTunes Store for iPhones. Figure 2.1 and 2.2 shows Pocket Codes' UI and an example project with "Alice in Wonderland" characters.

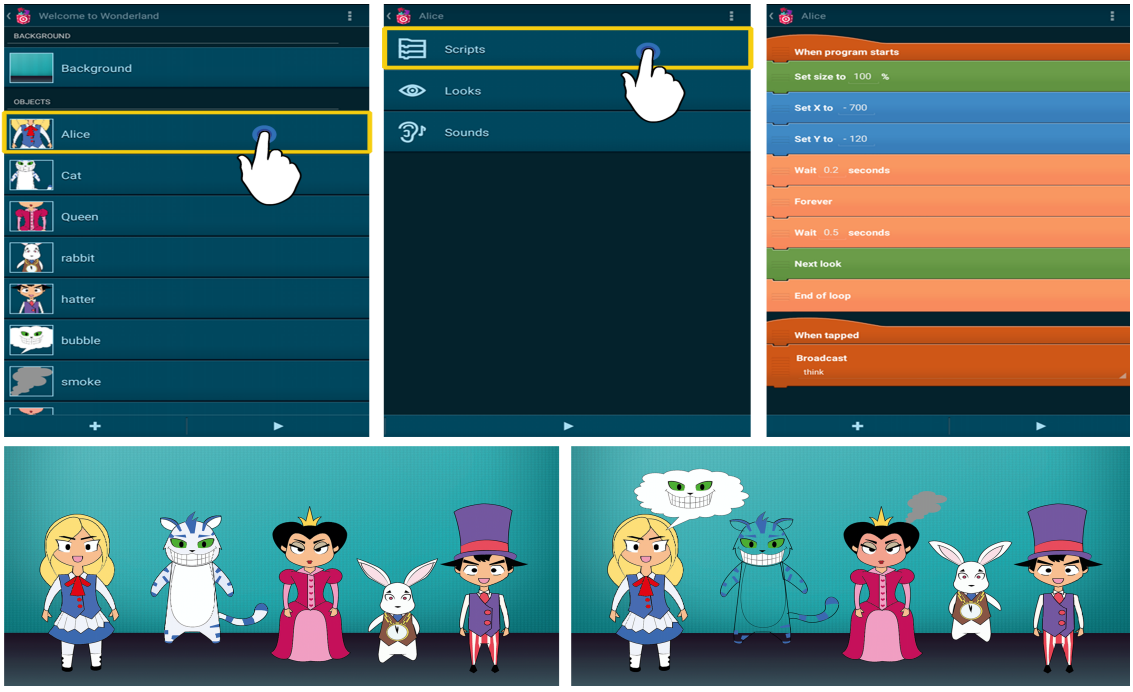


Figure 2.2.: Pocket Code Alice themed program

[4] https://catrob.at/pc

2.7. Pocket Code: the Mobile Integrated Coding Environment

Projects in Pocket Code follow a similar syntax to the one used in Scratch and are created by snapping together command bricks. They are arranged in scripts that can run in parallel, thereby allowing concurrent execution. To communicate between objects, to trigger execution of scripts, or scripts beyond objects, broadcast messages are used. By means of this mechanism, sequential or parallel execution of scripts is possible, either within the same object or over object boundaries. In addition to the basic control structures, Pocket Code offers event triggering building blocks for event-driven programming. Familiar concepts, such as variables, lists, or Boolean logic, are included as well.

In addition to Scratch, Pocket Code has a 2D physics engine integrated which enables the user to define certain physical features of objects and the stage (e.g., collision detection, velocity, gravity, mass, bounce factor, or friction) to create from simple up to complex simulations of real world experiments. With Pocket Code's intuitive merge functionality, the new parts of two projects can be seamlessly merged together into one larger project parts of the two initial projects that exist in both are not duplicated. This makes programming cooperatively much easier. Modern smartphones are equipped with a large number of sensors, although most mobile games only use few or none of them [77]. Within Pocket Code, users can create games using the device's sensors, such as inclination, acceleration, loudness, face detection, GPS location, or the compass direction, which makes user input easy and engaging. With Pocket Code it is also possible to connect via Bluetooth to Lego® Mindstorms robots or Arduino™ boards. The following extensions are available: Lego Mindstorms NXT/EV3, Parrot AR.Drone 2.0 and Parrot Jumping Sumo Drone, Arduino, Raspberry Pi (via WiFi), NFC tags, Phiro robots[5], and Chromecast. These kinds of computational construction kits make creating programmable hardware accessible to even novice designers and combines coding and crafting with a rich context for engaging teenagers [77]. In the context of robots, being able to program a smartphone makes much more sense, as the smartphone can be mounted on the robots, thus allowing to give it a face, a voice and other sounds, and additional sensors such as acceleration, inclination, magnetic field, GPS, voice recognition, computer vision. Also, since only a smartphone is needed with Catrobat, the programming can be done on the spot, outside, e.g., when using one's land-based robot or flying drone outdoors. With Catrobat no laptop or PC is necessary, thus, coding can take place anytime and anywhere, and in particular can be widely made available even in less affluent communities around the world. In addition, Catrobat released a Scratch Converter to allow the conversion of existing Scratch projects to the Catrobat language directly within the app, so there are, in fact, now around 30 million projects available for remixing and inspiration to our users.

[5] http://www.robotixedu.com/phiro.aspx

The Pocket Code interface consists of several very distinct areas: First, the app itself with a main menu and the collection of downloaded or developed-by-oneself projects, second a community sharing[6] platform, which is integrated into the app as a web-view, and which serves as a learning, sharing, remixing, cooperation, and publishing place, third the "stage" where projects get executed on the phone, and additionally a sophisticated graphical editing program that allows to draw and edit the looks of all actors, objects, and backgrounds of one's projects. This community website provides an online platform for users to download and upload programs, share them with other users, search for programs, and to provide feedback, e.g., write a comment to a project or rate a project. In addition, tutorials and starter projects are provided. In the community website's project overview, users can execute the project directly in desktop browsers (HTML5 web player), download the project to the Pocket Code app, or download the project as a standalone Android app. In addition, the tool automatically creates statistics from Pocket Code projects and provides an online code overview. The project details page is illustrated in Figure 2.3. Figure 2.4 illustrates the options within the main menu.

If the user starts first with a new and empty program, it initially only contains one empty background object. With the "+" sign users are able to add objects, looks, or sounds (depending on which activity he or she is in). The background object itself can be assigned to several backgrounds, which can be exchanged during runtime. The background can also have its own scripts. Every project can consist of multiple objects and at least one background (which is a special kind of object). Every object can hold a.) scripts that define the behavior of the object, b.) looks which can be changed and used, e.g., for object animation, and c.) sounds to make the object play music, other sounds, or recorded speech. Scripts can control the looks and sounds. Looks can be drawn and edited with Pocket Paint. Pocket Paint[7] is a second app of Catrobat available on Google Play, which allows users to create their own objects with a pencil or different shapes (note that we currently work on integrating this second app completely in Pocket Code in order to simplify the installation for our users). Distinctive features of Pocket Paint include the ability to use transparency, to zoom in up to pixel level, to change the dimensions of the looks, and to use layers, the latter being particularly interesting to create consecutive looks from an animation series. In addition, users can add looks with their camera, from their phone's memory, or use Catrobat's Media Library with a collection of predefined graphics. To add a new sound the user can either record a sound directly in Pocket Code, add a sound from the Catrobat Media Library, or add a sound from the phone's memory. This workflow is illustrated in Figure 2.5.

[6] http://share.catrob.at
[7] http://catrob.at/PPoGP

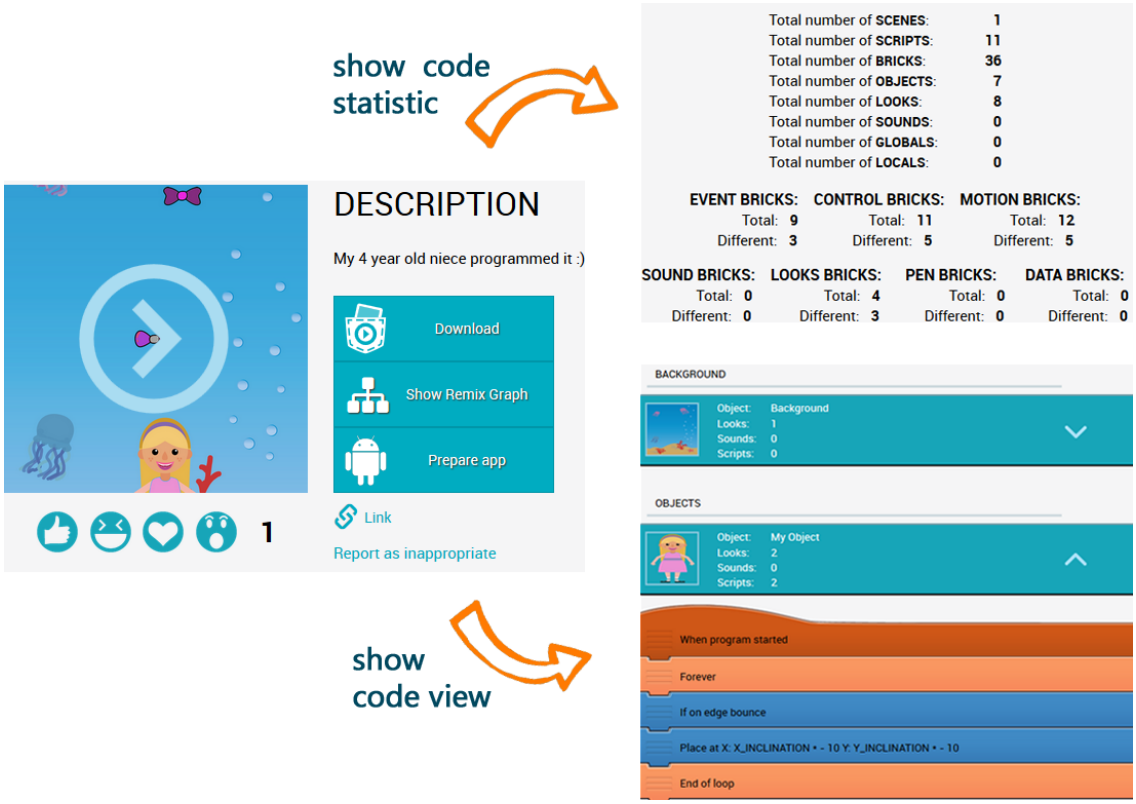


Figure 2.3.: Pocket Code web share: project details page with code statistic and code view

A script is a collection of code blocks that contain the logic of programming and define the operations of the object. Thus, it is possible to move the object and access its properties and change them. For adding scripts there are seven different brick categories (see Figure 2.6): a.) The Event category in dark orange that contains hat-bricks or broadcast bricks. Hat bricks are special kinds of bricks that, depending on certain circumstances such as a tap on an object, start the attached script; b.) The Control category in orange contains if-then-else bricks, loop-bricks to control the ow of the script, bricks to switch between scenes, clone bricks, etc; c.) The Motion category in blue color contains bricks to manipulate the object's position, orientation, or movements; d.) The Sound category in purple contains bricks to start and stop sounds, manipulate the volume, or accept spoken input; e.) the Looks category in green contains bricks to change the graphical appearance of the object, e.g., set/change size, brightness, transparency or hide/show the object as well as set a certain look to animate the object, to show speech and think bubbles, or to ask for written user input; f.) The category Pen in dark green holds bricks for drawing lines (a pen that follows the object) and the option to leave stamped marks of the object on the background, g.) The Data category in red contains bricks to manipulate variables and lists, e.g., to set/change

variables, maintain lists, add/insert/replace items, and show variable content on the stage. This color scheme makes it possible to understand scripts more easily through of the bricks' color which supports readability. By activating extensions in the settings menu, additional categories appear for Lego (yellow), Drone (brown), Arduino, the Phiro robot (both in cyan), etc.

Figure 2.4.: Pocket Code main menu: within the settings menu you can find, e.g., the accessibility preferences or the Scratch Converter; 2) create a new project by starting with an example game or with an empty game; 3) project overview: tap on one to execute or modify it; 4) find help: videos, tutorials, step-by-step tutorials, education page for teachers and students or google groups forum; 5) download and play games from other users; 6) upload your game to the sharing platform.

preferences or the Scratch Converter; 2) create a new project by starting with an example game or with an empty game; 3) project overview: tap on one to execute or modify it; 4) find help: videos, tutorials, step-by-step tutorials, education page for teachers and students or google groups forum; 5) download and play games from other users; 6) upload your game to the sharing platform.

The **formula editor** looks like a calculator and allows the creation and execution of mathematical and logical formulas that can be used in bricks. The formula editor is shown in Figure 2.7. It

consists of an input field to show and compose the formula, a keyboard, and a compute button to display the current result. On the keyboard, five categories for various values, functions, and operators are available. a) Object: a collection of values of the current object, e.g., values for the X- and Y-coordinate, or the current speed, b) Functions, such as sin or cos, a random number generator, or list and string functions, c) Logic is used to compare values or to combine logical expressions, d) in Device there is information that the smartphone or tablet records, e.g., inclination, loudness, or GPS data, and e) Data stores created variables and lists and shows their last value. With a tap on the play button the program starts. The objects are shown on t**he stage**and the scripts are executed. To stop or to pause the program, the user has to tap on the back button of the phone. A stage menu appears which can be seen in Figure 2.8. The stage is organized in a logical coordinate system with an X- and Y-axis, which allows an exact positioning of the objects. This axis can be displayed in the stage menu (see Figure 2.8c).

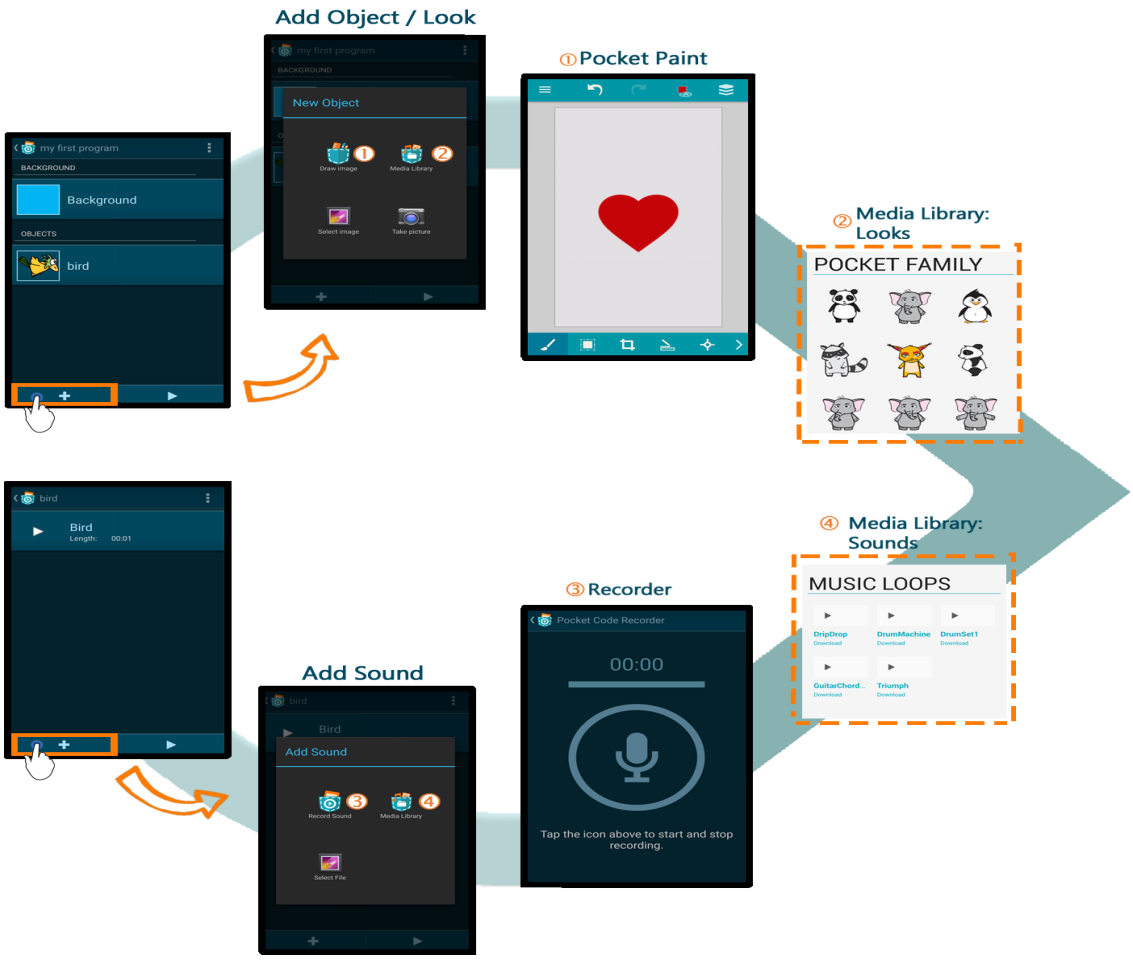


Figure 2.5.: Pocket Code's UI: add a look/object or add a sound with the "+"sign

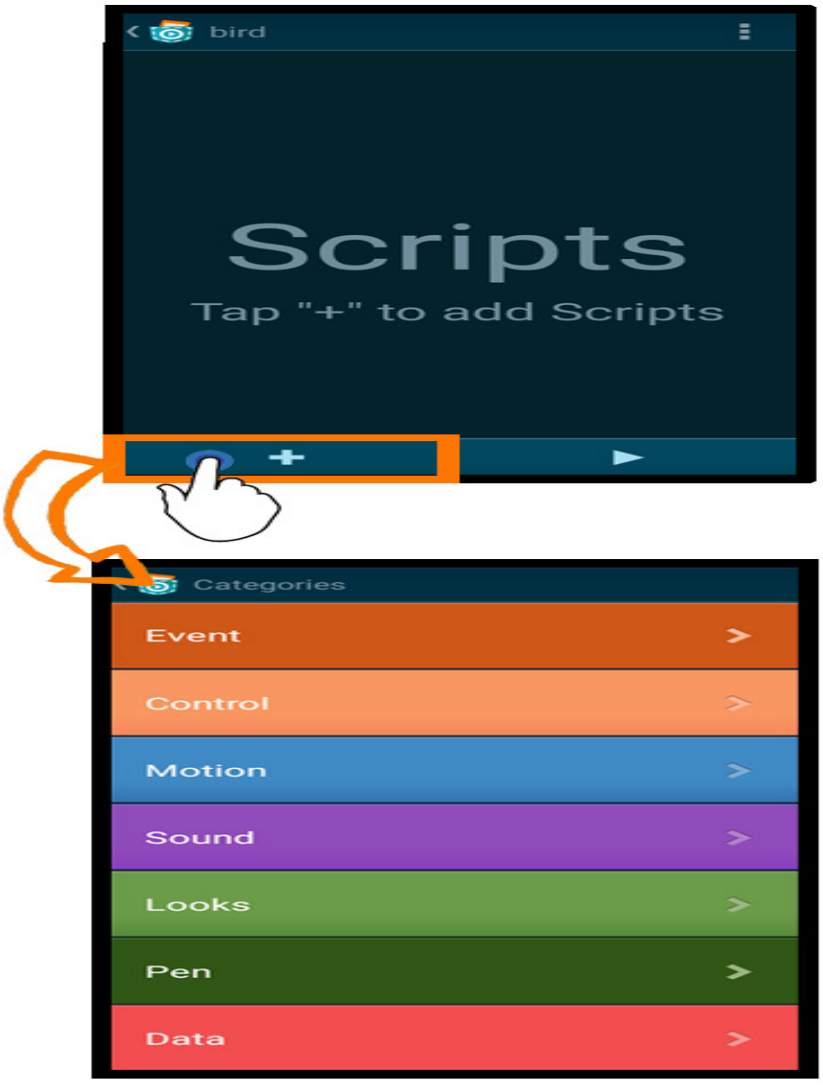


Figure 2.6.: Script categories: choose bricks from the seven basic available categories.

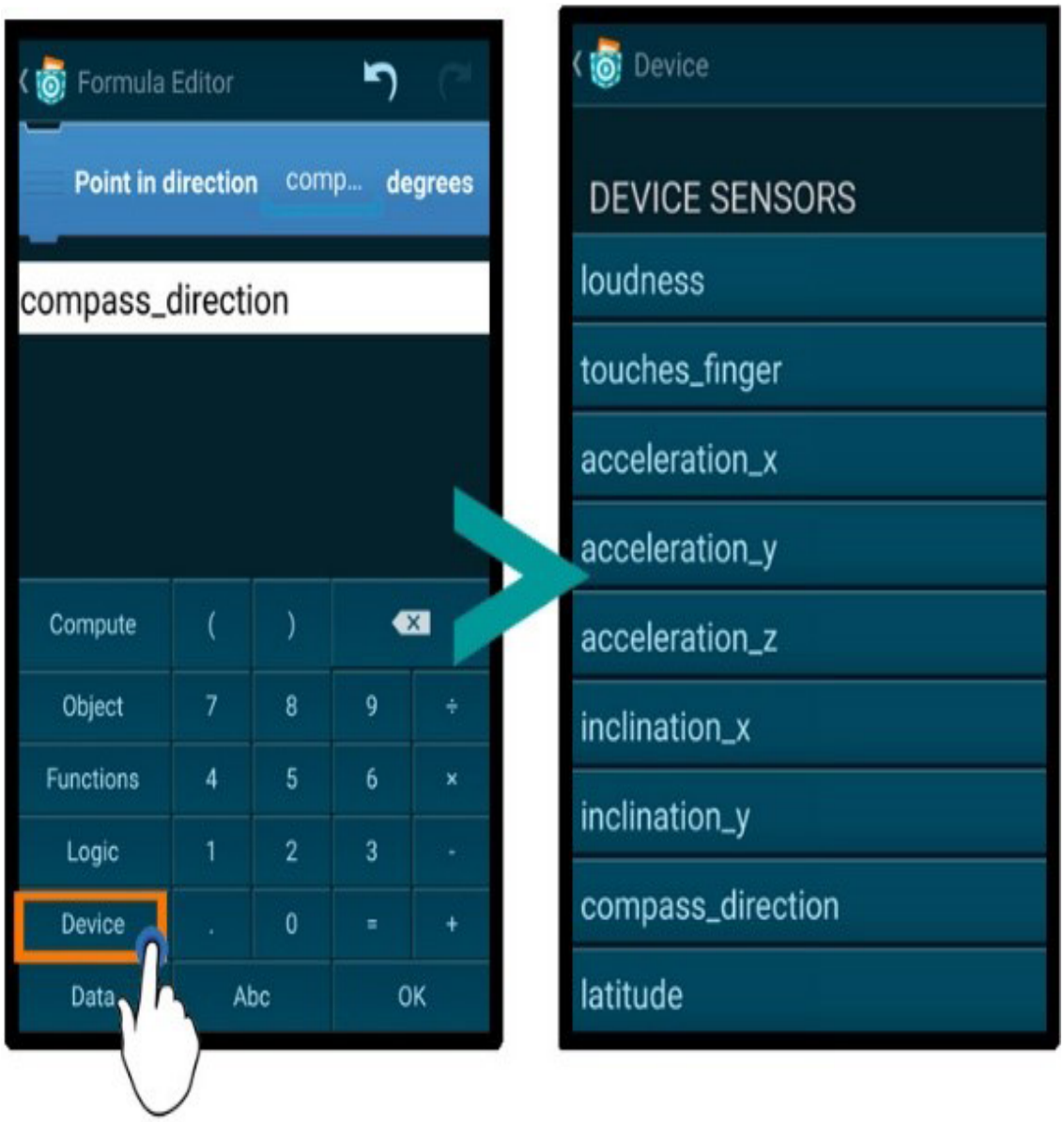


Figure 2.7.: Formula editor: the value for the direction can be dened as a constant or, e.g., a sensor can be chosen by tapping on “Device” .

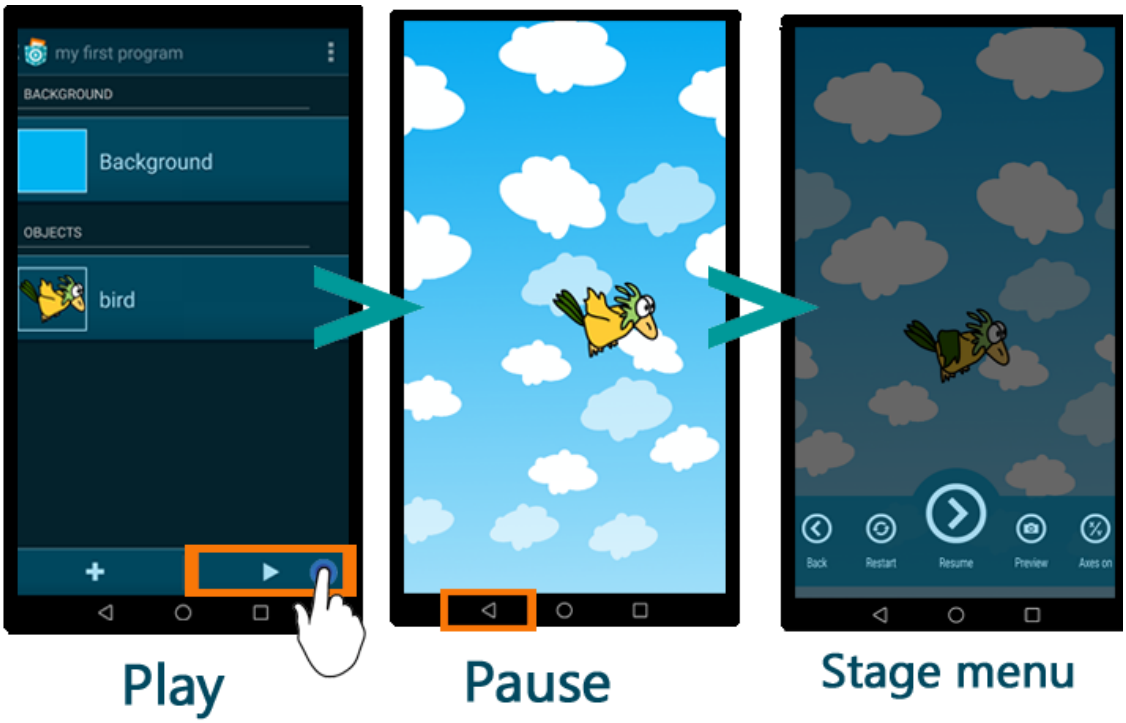


Figure 2.8.: Stage; a.) tap the play button to start the program, b.) tap the back button of the phone to pause the game, c.) in the stage menu the user has five options: 1. tap back again to stop the game and switch back to editing of project, 2. restart the game, 3. resume the game, 4. add a new preview picture to the project (this will be shown, e.g., on the sharing platform), and 5. display the x/y axes on the device screen.

2.8. Projects and Further Work

This new and forward-thinking approach to code on mobile devices received national and international recognition. The Catrobat project has won a number of awards, including 2016 two Lovie Awards ex aequo with Red Bull and Doctors without Borders, evaluating the best European digital projects in London, and the Reimagine Education Award for innovative educational projects at the Wharton Business School of Pennsylvania[8]. Additionally, in March 2017, Catrobat won the "Platinum Award" in Best Mobile App Awards Best Educational App category and 2016 the "Internet for Refugees"[9] award for a Right-to-Left language implementation of Pocket Code, which supports several RTL languages, e.g., Arabic or Farsi, and particularly focuses on refugees and teenagers in crisis or development areas. A new project was started in January 2018 which

[8] http://www.reimagine-education.com/awards/reimagine-education-2016-honours-list/
[9] http://www.tugraz.at/en/tu-graz/services/news-stories/tu-graz-news/singleview/article/preis-internet-for-refugees-fuer-programmier-app-der-tu-graz

promotes remote mentoring by connecting female role models with female programming beginners. This idea was awarded with the "Closing the Gender Gap" prize of the Austrian NetIdee in November 2017. During the European H2020 project "No One Left Behind" (NOLB), the team developed a special flavoured school version of the app with the name "Create@School". This version compromises, e.g., the gathering of analytics data for visualization, the integration of accessibility preferences, and the development of pre-coded templates. Currently a new flavored version customized for female teenagers is in development. This version with the name "Luna&Cat" promotes special content for girls, like featured and user-contributed programs, media assets, and tutorial videos.

One new feature which is currently in development is an extension to program embroidery machines. Once available, self-created patterns and designs can be stitched on t-shirts, pants, or even bags or shoes. With Pocket Code, the embroidery machines will be programmable, similar to the existing TurtleStitch[10] project, which realizes this concept on a PC (while with Pocket Code only a smartphone is needed). As a result, teenagers have something they can be proud of, something they can wear, and they can show to others. This feature has proven to be especially engaging for female teenagers and show them new ways of expressing themselves creatively through coding. Figure 2.9 shows an example of an embroidery pattern made with Pocket Code.

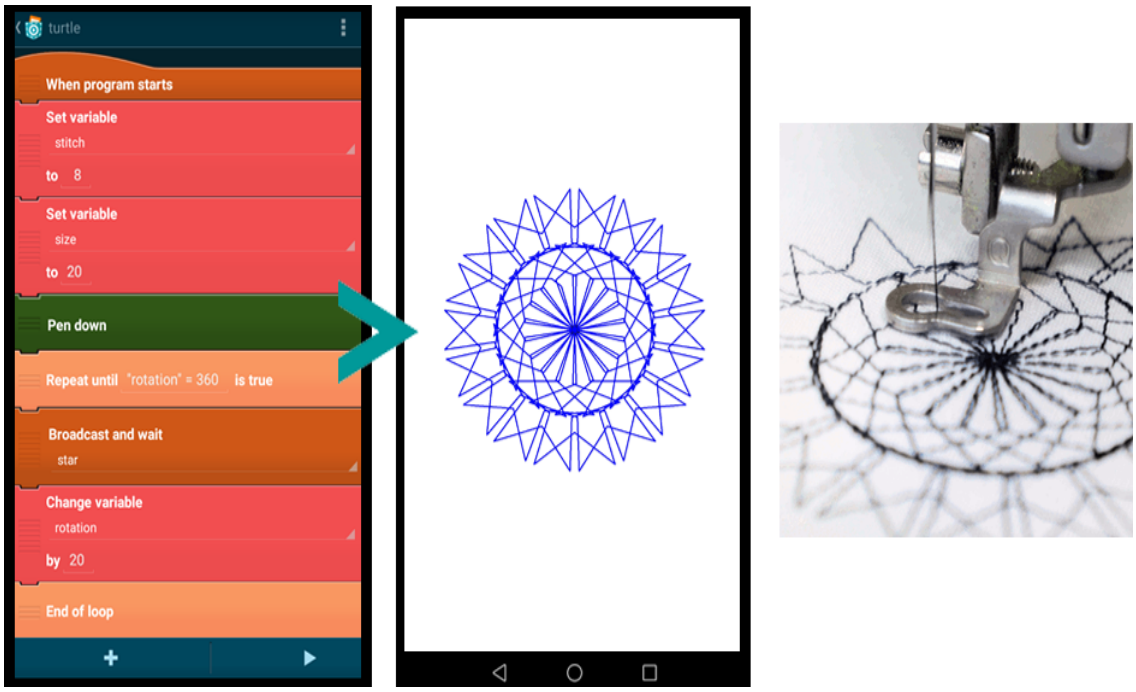


Figure 2.9.: "Stitched" patterns in Pocket Code. Picture on the right with kind permission from Andrea Mayr-Stalder, www.TurtleStitch.org project.

[10] http://www.turtlestitch.org/

Another new beta feature allows registered users to sign and release the Catrobat project as apps on Google Play. Users optionally also add mobile ads to earn money. In developing countries, mobile technologies are playing an important role in developing economies [4]. This is because mobile phones and the mobile internet require considerably fewer financial resources in comparison to a traditional desktops and laptops [122]. Due to lack of alternate employment opportunities in developing countries, the low cost of investment is a critical enabling factor for new entrepreneurs [4]. The economic advantage of releasing an app on Google Play or integrating AdMob within apps can motivate many people to learn programming and solve issues digitally. This is especially beneficial for under-privileged user groups who have access to limited resources like computers and continuously available electricity. They can benefit by sharing their creativity with others and serve a global market with minimal resources such as a low-cost smartphone and mobile internet, which are increasingly available everywhere. Figure 2.10 shows an example of apps with AdMob integration.

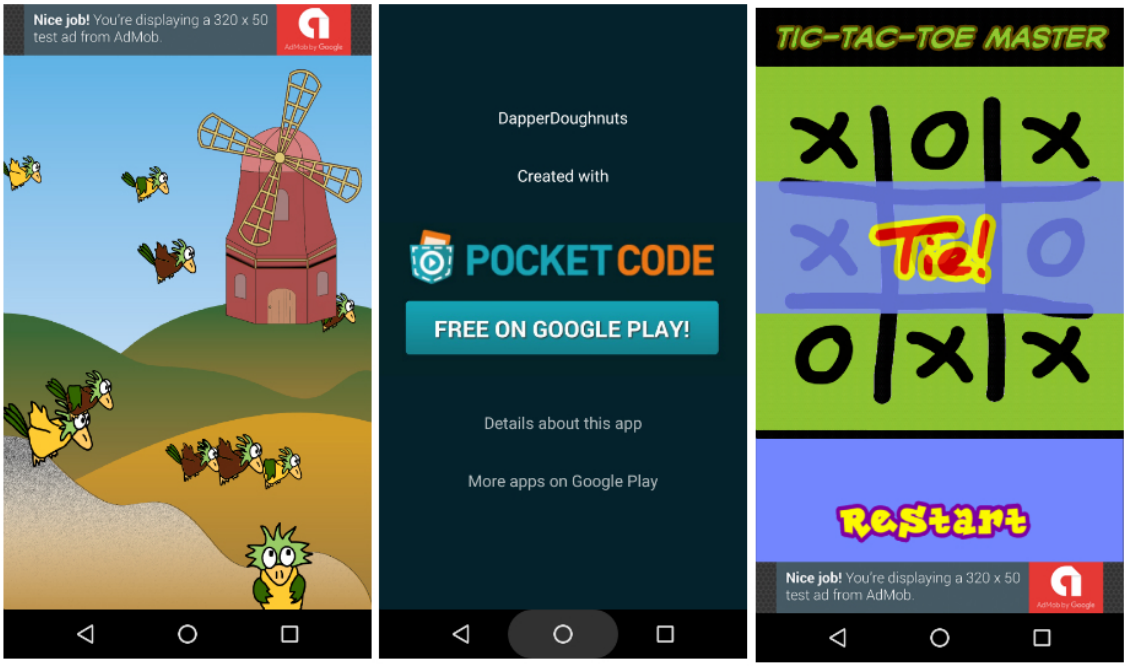


Figure 2.10.: Android apps with AdMob banner advertisement created with Pocket Code

2.9. Conclusion and Discussion

The aim of this paper was to provide an overview about the Catrobat project and the Pocket Code app. Pocket Code is an easy way to start coding or to be used in school settings for project work. Pocket Code is not intended to develop standard applications, but to promote understanding the logic behind coding and foster conceptual thinking, thus following a constructionist approach in learning by doing and the creation of sharable artifact. Creative and artistic talents can be recognized and in schools learning can occur in a student-centered, project-based setting with the use of new media. Users of Pocket Code are mostly teenagers who can learn from each other and share their ideas to create new games and other apps together. The community sharing platform allows users to give and receive feedback, support, and assistance from others around the world, thus allowing our users to stand on the shoulders of their peers and learn from each other. They can try out new ideas and realize the projects they define for themselves, aided and inspired by likeminded others in a user-friendly and social environment. Catrobat fosters diversity and learning in a worldwide community. Our goal is to empower teenager all over the world to realize their potential and express themselves creatively with today's and any future technology.

Chapter 3

Improving Pocket Paint's Usability via Material Design Compliance and Internationalization & Localization Support on Application Level

This chapter was originally published in the the Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services. ACM'17 *[9].*

3.1. Abstract

This paper discusses the implementation of Google's material design guidelines, internationalization (I18n), and localization (L10n) for mobile applications in the case of Pocket Paint, an Android painting application. The intended goal of this redesign is to improve the app's usability and hence broaden its users base. The main challenges in implementation of the redesign, beside the refactoring and extension of the code base, to comply to the design guidelines, are the intricacies to thoroughly support bi-directional scripts, e.g., the positioning, translation, mirroring of graphical elements, the when and when not to mirror. Through a UX test with six [11] users of our target group (age 13-17), feedback about the perceived quality of the guideline implementation and the improved usability were collected. The outcome of this test showed that, all participants rated the redesigned app as simpler, more appealing and concise in comparison to the previous version.

3.2. Introduction

User experience (UX) is the result of enjoyable interactions and/or anticipated interactions with a product [83]. It is concerned with look, feel and usability of a product. The two most influential disciplines involved in UX design are backend development and visual/graphics design [83]. Google's Material Design are visual, interactive and motion guidelines for mobile app and web development. By following such guidelines users benefit from known usability patterns shared amongst apps and web sites complying to these guidelines although they inhibit innovation in usability [121]. App vendors strive for more users, users want better, more stable and more usable apps which pushes app development towards an increased usable feature set. To broaden the user base an app should be internationalized and localized. Taking into account that for instance Arabic is spoken in more than 23 countries[1] or more than 30 languages[2] use scripts written from right to left (RTL) not only multi language support but also bi-directional script support is needed. To further increase usability of an app its language support should be independent from the underlying Operation System.

Localization is a long term commitment and is much more than just translation. Usability, especially mobile usability is a broad topic and there are various localization considerations necessary when developing a world-ready app, e.g., language selection and translation, layout, text, directions-sensitive graphics (especially for bi-directional support), and gestures. Pocket Paint is a mobile paint editor developed by the free and open source non-profit Catrobat[3] project (creator of the free educational Pocket Code app, to visually create programs directly on one's mobile device). Pocket Paint supports transparency and zooming up to pixel level, which are not wide spread functionalities for mobile paint apps. It is integrated into Pocket Code - a mobile app to visually program games and various apps directly on one's mobile device - but can also be used on its own. Although there are no commercial interests behind Pocket Paint (it is open source software, free of charge and advertisement), a face lift was necessary since the app was designed complying to the Android Design guidelines of the year 2012 and first released in 2013.

Good mobile user interfaces (UIs) should comply to a set of consistent usability guidelines. Such guidelines do exist for every (mobile) platform to make all available apps compliant with the overall system concept. The guidelines for Android are the Material Design Guidelines[4]. UX Guidelines are beneficial for users since usability patterns are recognized over application borders and makes operation of different apps similar and hence easier. With the expanding software market in general and the mobile market in particular it is necessary for applications to support different languages, scripts and cultural idiosyncrasies. To enable such support and mobilize additional user groups, already the design of the software has to be done with focus on interna-

[1] http://www.nationsonline.org/oneworld/countries_by_languages.htm#Arabic
[2] http://www.i18nguy.com/temp/rtl.html
[3] https://www.catrobat.org
[4] https://material.io/

tionalization (I18n) and localization (L10n). The Pocket Paint redesign was started not only for material design guideline compliance and hence improved usability but also to introduce a limited set of feature requests and to broaden the user base by multi-language and bi-directionality support on application level which enables languages using right-to-left (RTL) scripts.

3.3. Background

3.3.1. Material Design

Material Design guidelines are compiled and updated frequently by Google since 2014 and can be accessed via https://material.io describing visual, interactive, and motion guidelines for mobile app and web development. Material is a metaphor, a system for uniting style, branding, interaction, and motion under a consistent set of principles [61]. The aim of Google Material Design is to create a unique visual design that incorporates all the company's services [81]. With its rich set of guidelines, principles and resources [119], it became leading industry standard in development.

3.3.2. Internationalization, Localization, Bi-directional Script Support

According to data collected by Google and Admob in March 2014, the number of users who have stopped using an app because it was not localized properly varies between 34% and 48% depending on the country these data were collected (United States; China; Japan; United Kingdom and South Korea) [62]. Paintroid previous versions supported internationalization (I18n) and localization (L10n) on operating system (OS) level. This means that the system language is detected and the app's language is changed accordingly. From a usability point of view it makes sense for users to be able to switch the language of an app either with the language setting of the operating system or independently so users can choose whether to use an app language different from the OS. This is particularly true for poorly translated applications. Furthermore there are languages, e.g, Sindhi or Pashto, which are not (yet) supported by the OS.

Companies developing and selling mobile applications always strive to enhance their app downloads and revenue. Internationalization (I18n) and localization (L10) are now important factors in increasing the market share of an app. When trying to market a game or other mobile app, it is necessary to understand all the localization aspects needed to increase the potential user base of the app. It is essential to internationalize and localize mobile apps if a company intends to deploy applications in various countries. Developers must ensure that different languages will not break the application's UI elements or have an impact on its usability.

Although internationalization and localization are colloquially often used synonymous there is a subtle difference. Internationalization refers to the process of (re)engineering an application

so that it can be easily changed to various languages and regions without further modification (make the app world-ready). While localization refers to the process of adapting internationalized software for a specific region or language by adding locale-specific features and translated text (making the app language and culture specific) [10].

Internationalization and localization of an application requires more than just translating UI strings [3]. It should be considered from the very beginning of the application design and development since late changes and adaptations are time consuming and hence expensive. The following list contains generic items about internationalization that should be considered by app developers [10].

- Locale and culture awareness (formats dates, times, addresses, units of measurement, and phone numbers)
- Numbering style (Arabic or Hindi digits)
- Layout direction (e.g. left to right in Western languages, right to left in Arabic, Hebrew and Persian)
- Character encoding scheme for textual display
- Case conversion
- Complex scripts awareness
- Sorting and string comparison

Localization deals with adaptation of software and content.

- Language translation text
- Layout direction (i.e., mirroring awareness)
- Direction-sensitive graphics (e.g. undo, redo, back, and forward)
- Spelling variants for different countries where the same language is spoken, e.g. localization (en-US, en-CA) vs. localization (en-GB, en-AU)
- Images and colors: issues of cultural appropriateness
- Names and titles

Adjusting layouts, graphics, colors, and menus for other regions and languages should be performed by people who are familiar with the cultural and linguistic requirements of the particular country and specific region. It would be beneficial if the adaptations is done by usability engineers or people aware about software ergonomics and cultural variations.

3.3.3. How does bi-directionality affect UI design?

User interface design is the most critical issue in bi-directional (BIDI) language software. Text can flow in both directions. Western languages are written from left to right (LTR). Arabic and Hebrew, for instance, are written from right to left (RTL). The direction of reading and writing affects how information should be presented on the screen (i.e. mirroring awareness) [10]. Mirroring is needed to give a consistent RTL look and feel to the UI. This requires, not only the text alignment and text reading direction rendered from right to left, but also the UI elements. Mirroring for a layout is done by coordinate transformation from LTR (where the origin (0, 0) is on the upper left edge of a screen) to RTL (where the origin (0, 0) is on the upper right edge) as shown in Figure 3.1.

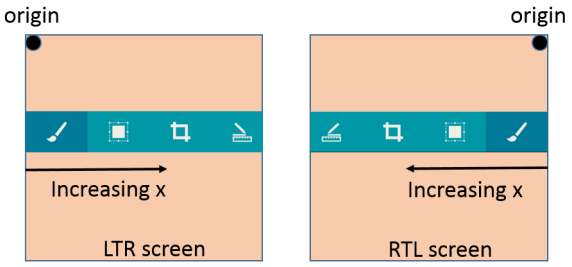


Figure 3.1.: Layout mirroring for LTR and RTL screens.

RTL text also affects the direction of some icons and images, especially those conveying a sequence of events[Google2014material]. For instance the flow of time is depicted from left to right in LTR languages but vice versa for RTL languages. In general an RTL layout is the mirrored image of an LTR layout, and it affects layout, text, and graphics. In RTL languages specific terms like hyperlinks, company or brand names are not mirrored or translated but written in the orientation of their origin. When a UI layout is mirrored, these variations occur [61]:

- Icons are displayed on the opposite side of an input field
- Navigation buttons are displayed in reverse order
- Icons having a particular directional orientation, e.g., undo and redo, are mirrored for bi-directional languages.
- Translated text is aligned to the right

The types of items are not mirrored in bi-directional languages [Google2014material]:

- Icons that do not have a particular directional orientation, such as a camera
- Video controls and timeline icons because they symbolize the direction of the tape
- Numbers and phone numbers
- Charts and graphs (x-and y-axes in RTL are the same as in LTR)

- Musical scores
- Images (except if they depict direction or order)

3.3.4. Multilingual User Interface Support on App level

For 2020 the shipments of smartphones with Android or iOS are forecast to reach around 1.8 billion units worldwide [76]. Due to this global situation, companies vending mobile applications can not afford to publish apps containing critical bugs or usability issues. If users have a bad experience, they will uninstall the app and switch to a competing product. If applications do not support different languages, they will not be used in countries where the apps' languages are not supported. This can limit the potential user base significantly.

Making an app available in several languages and locales improves its global reach. When an app is available in an app store for the target locale and language, the app will be used by more people who speak the local language. To reach a global audience, Pocket Paint supports localization on application level which makes it independent of the language of the underlying operating system. It enables the end user to select the preferred language from a list of supported languages.

Pocket Paint includes culture-specific strings that are translated into the languages of the target locales. It's a good practice to keep such resources isolated from the rest of the app which facilitates the support for more locales and languages. For every language and locale unique directories are created inside the app's resources folder.Alternative translation string files are stored in these locale-specific resource directories, containing graphics, sounds, layouts, and other locale-specific resources, e.g.,layout-ldrtl ... specific layout for any "right-to-left" languagedrawable-ar ... specific graphic files for Arabic

The operating system will load the appropriate resources according to the user language settings of the app at runtime. When the language of an app is changed, the whole app is recreated. So if the app has any data object, the object's instance state must be handled (saved and restored). Figure 3.2 illustrates the sequence of steps that must be implemented to change the locale (language and country) according to the language preferences. The multilingual feature provides an interface that handles user actions to change language independent of the smartphone operating system's language settings. Since language support is implemented on application level languages can be supported by the app which are not yet supported by the operating system.

3.4. Redesigning Pocket Paint

3.4.1. New Features and Enhancements

The new version of Pocket Paint and its new features such as navigation drawer, landscape device orientation, layer support and feature discovery (help) now comply to the material design

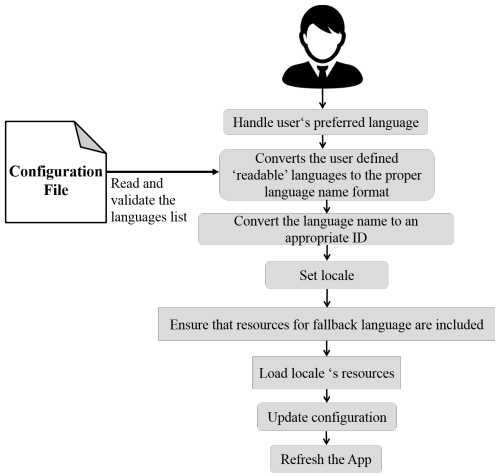


Figure 3.2.: The sequence of steps to change locale (language and country).

guidelines. The nav drawer is used as a new main menu. The app is now aware of the device's orientation and changes the GUI between portrait and landscape accordingly. Layers have been introduced to enable the user to metaphorically create a composite image from a stack of transparent slides. Feature discovery is an interactive help system which is shown at the first start of the app. This quick tutorial interactively explains the user the most important features of the Pocket Paint. Feature discovery can be skipped at any point and will not show up until triggered by the help item from the nav drawer menu.

Drawing tools have been enhanced to support more new shapes and the color theme has been updated to a more brighter theme which better matches material design. The most important enhancements in the new version of Pocket Code are multilingual and bi-directionality support on application level.

3.4.2. Bi-directional design guidelines for Pocket Paint

Localization is not only about different languages but include besides text, layout, and graphics.

Text and Layout

For bi-directional languages the default direction is from right to left. The layout of all UI elements should follow this direction. Therefore supporting bi-directional scripts needs layout customizations for text and also for all user interface (UI) elements, including buttons, textviews, seek bars, sliders, checkboxes, menus, and dialog boxes.

Graphics

Human computer interaction happens mostly via graphical interfaces. Therefore all graphical controls must not include any specific cultural attributes or idioms. Graphical UI components of a product need to be revised (made culture neutral) for the international market (i.e. internationalized). Since such attributes might be easily misunderstood users of different regions. During the localization phase, potential ambiguities can be easily adjusted. Also the use of puns in user interfaces should be avoided since they are prone to be misunderstood by non native speakers. This will minimize the expensive localization needs for graphical elements.

Icons that have a specific directional orientation such as back, next, undo, and redo should be given a special attention for bi-directional languages in order to be locale-aware. The localization team can present feedback about the graphics elements used in the product to ensure that they are convenient for all locales.

When text, layout, and icons are mirrored to follow the RTL direction, anything that describes time should be depicted as moving from right to left. The most prominent example for mirroring are back and forward buttons. In RTL layouts, backward points to the right $\rightarrow$, and forward points to the left $\leftarrow$.

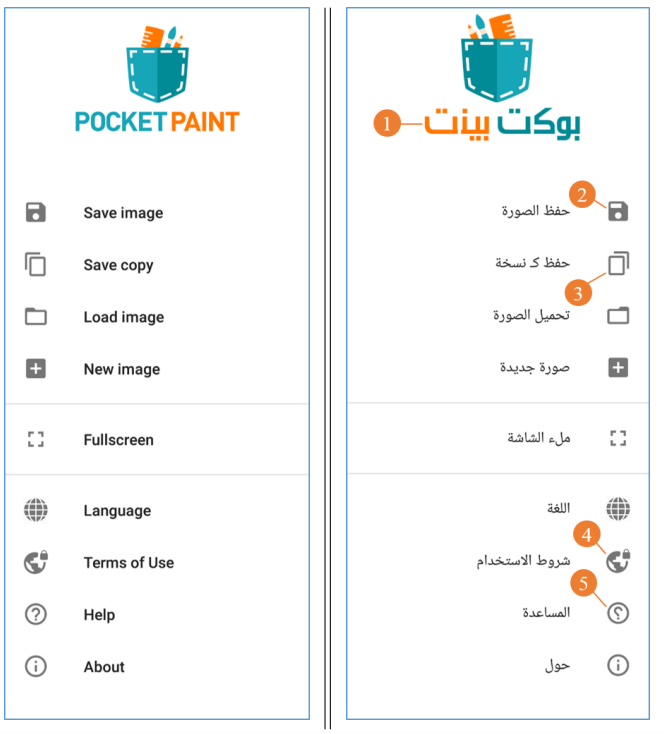
Since the RTL languages are written from right to left, printed documents or screen of applications are read from the upper right corner. RTL readers start scanning the screen in the upper right corner (i.e., a right-to-left, top-to-bottom order) therefore important information should, be placed there. Hence, icons containing representations for documents or packages need careful mirroring.

In Table 3.1 a direct comparison between left to right and right to left rendering of Pocket Paint's nav drawer is depicted. On RTL screens, text, icons, and UI elements are displayed flowing from right to left, with following five intricacies:

1. Localized version for logo in Arabic
2. Save icon appears right of the text - it should not be mirrored since it represents a real object (floppy disc)
3. Icons that communicate direction are mirrored
4. Icons that do not communicate direction are not changed
5. The question mark is just reversed and faces rightwards

For localization into RTL languages, the UI icons layout should naturally follow the RTL direction. Sometimes, the circular direction of time is depicted in icons, e.g., the undo and redo buttons. In Table 3.2 a direct comparison between LTR and RTL rendering of the drawing canvas is depicted. The intricacies are enumerated from 1 to 5.

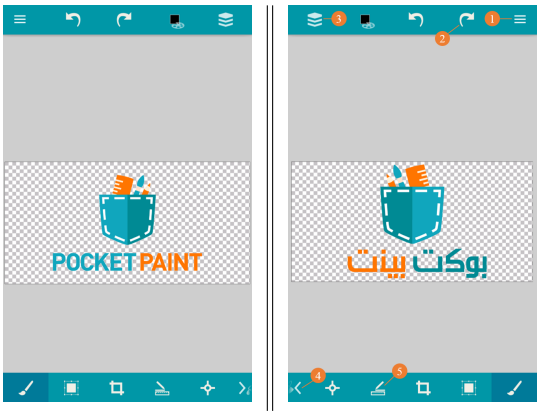
1. Navigation drawer button is right-aligned



a.) LTR & b.) RTL

Table 3.1.: Pocket Paint navigation drawer

2. Undo icon is mirrored to follow the RTL direction
3. Layer button is left-aligned
4. Next button points to the left
5. Icon that communicate direction is mirrored



a.) LTR & b.) RTL

Table 3.2.: Pocket Paint drawing canvas

The differences of the color chooser LTR and RTL rendering are shown in Table 3.3. The focus is set to the following intricacies:

1. Dialog's title is aligned to the right
2. Color palettes is right-aligned
3. Text appears to the right of slider
4. Slider is mirrored and should progress RTL
5. Hindi digits are used for numbering style

a.) LTR & b.) RTL

Table 3.3.: The color chooser

3.5. UX-Test

To get first feedback from real users to the redesign (changes and newly introduced features) of Pocket Paint, a UX-test was conducted with seventh grade students (age 12-13) using this app in their art classes. Due to the time constraints only a small group consisting of six students could be tested. All of them knew the previous version of Pocket Paint and therefore noticed the changes hence they could give feedback in relation to the new version. The UX test was designed to complete six small drawing tasks by creating certain picture elements. Every tasks was designed to use different drawing tools or aspects of the application. Completing these tasks resulted in similar images. The students were observed during their work on the tasks and then asked for feedback (see Table 1). Qualitative judgement was done in school-grade like manner: A=1 ... best grade, B=2, C=3, E=4, F=5 ... lowest grade. In average the simplicity of the app was perceived as having improved from 3.2 to 1.8, and the optical appeal from 3.5 to 1.3. This is a hint that the redesign is a step in the right direction. The students stated that the new version is clearly better and offers all features the old version had. Nevertheless during the test a couple of improvement points were isolated: a.) tools or functionality which could not be found within a short time so students needed guidance, for e.g., shapes, opacity, color palette, zoom as well as gallery and save, and b.) not or misunderstood functionality, e.g., image load vs. import,

Students:	Alfi	Bravo	Charlie	Delta	Echo	Foxy
New: simplicity	2	2	2	2	2	1
Old: simplicity	3	3	2	5	4	2
New: appeal	1	2	2	1	1	1
Old: appeal	4	4	3	4	3	3
New: advantages	- more is possible - concise interface	- creative drawing is now possible	-	- wide range of functionality - step-by-step intro is helpful	- draw easily as one likes - import images	- it is difficult but fun
New: need improvement	- hard to find transparency	- sometimes need to start over again	-	- some things don't work out of the box	-	- drawing on tablet is hard
Old: advantages	-	- no	-	- No, all features are included in new version	-	-

Table 3.4.: Results of the UX test with six 7th-graders

undo/redo functionality, deselection of tools, crop functionality. After analysis of the students' feedback the following action items were identified: a.) give hints how to use touch gestures, e.g., zoom & drag, b.) improve visual feedback for placement of shapes and import-graphics c.) rework of undo/redo d.) unify the appearance of the color dialog and finally e.) include the layer-feature as well as the navigation drawer into the app's introduction and help. These items along with some minor bugs which were discovered during the test will be implemented and fixed until the official release in the Google Playstore around June 2017.

3.6. Conclusion & Future Work

In this paper we laid out what is necessary to thoroughly support Internationalization, Localization and bi-directional script support and their intricacies. We showed how the mobile app Pocket Paint was redesigned by complying to Google's material design guidelines to increase usability and broaden the multi-language and bi-directional script support on application level. The UX test showed that the design changes were positively accepted. It is planned to start a sequence of UX-Tests with different potentially slightly larger sets of people (using sample set estimation according to [130]) of our target user group (age 13-17) to incrementally improve the usability and stability of our Paint app especially with focus on multi-language and bi-directional script support. Furthermore we will monitor the development of number of downloads in Google's Play Store to see whether the language support on app level lead to more people installing and using the application.

Chapter 4

Pocket Code Build Variants

This chapter was originally published in the the Proceedings of IEEE International Conference on Innovative Research and Development (ICIRD) , pp. 1-6. IEEE'18 *[88].*

4.1. Abstract

This case study is about Pocket Code's build variants. Pocket Code is a free and open source integrated development environment (IDE) for the brick based visual programming language Catrobat. It is released in various flavors for different partners and projects (e.g., Create@School, Phiro and Standalone). All flavors use the same code base but slightly differ in design and functionality. If different flavors are maintained as separate projects, all projects require proper maintenance. Any feature introduced or updated in one project must be ported to all others, for that they dont diverge. With an increase in the number of flavors, efforts to maintain will also increase which renders the project unmaintainable. If all flavors are maintained in one project, it is challenging to release more than one version of an application with a different set of functionalities and different user interface (UI) enhancements. In this paper, Pocket Code's different build variants are discussed particularly the standalone build variant. To build a standalone version of any app hosted on the Pocket Code sharing platform, the user has to trigger the build via the web interface on the remote Pocket Code server. Resource files and app configuration are generated based on user input. This paper can be of interest to organizations dealing with dynamic build variants triggered by external actors.

4.2. Background & Introduction

Software development became increasingly complex and rapidly evolving. Due to diversity in hardware, operating system, cultures, user-groups, accessibility, and functionality requirements.

One common problem the software industry faces is releasing different flavors of the same application with a different set of functionalities [94] [21], e.g., free and paid versions of the same application, and or releasing an app with different brand name, logo, or color theme. A simple solution to achieve this is to maintain separate projects [133]. That is to say, each project with a separate design and functionality set. With this approach refactoring or updating core functionalities across all variants manually, i.e., keeping them synced, will turn this into a tedious, error-prone, and non-scalable project [133] [94], however another approach is to make core functionalities as a library project and use the library across all flavors. This would prove to be a good workaround to maintain core functionalities for all projects. Albeit, the approach actually turns the development into a library project, which is different to app development [36]. Moreover, developers need to update the latest version of the library in a flavor-specific project [74]. To avoid "dependency hell" [74], a better approach would be when one project implements all different variants along with a configuration system, responsible to enable or disable features or aesthetic details. Configuration can change the behavior of software at build time, deploy time, and runtime [74] [115]. The configurable software is not always a solution as cheap as it appears [74]. It includes all resources, code, settings for all variants in a single application. Due to the increased size of polymorphous software, it is necessary to build software variant out of the main repository having only features and resources used by that flavor. To achieve this, software industries are using dynamic software product lines (DSPLs) [106][21]. Software product lines are used for building products, reusing features, and exploiting variability and configurable options [21]. Software product line engineering is considered one of the major accelerator in software development process [38]. They drastically reduce time spent on build configuration, lesser time to market, increased productivity gain, user-satisfaction, product quality and development moral [38] [93].

Mobile Application Product-line Engineering is an emerging and promising research area [106]. Software product line (SPL) engineering aims at variable software by generating a set of tailor-made programs from a common code-base (e.g., for different customers or application contexts) [111]. In configurable systems and product lines, variability is an important characteristic [55]. "Software variability is the ability of a software system or artifact to be changed, customized or configured for use in a particular context." [15]. Today, many systems are variability-intensive to meet functional and non-functional aspects of software and various business goals, such as reduce time to market, budget efficiency, and quality enhancement [55]. There is little research on variability. Specifically, the usage of variability mechanisms and techniques in Android are still unclear [55].

The Pocket Code Android version is released in various flavors for different partners and projects (e.g., Create@School, Phiro and Standalone). All flavors use the same code-base while only slightly differing in design and functionality. To release various flavors from one common code-

base, Pocket Code DSPL uses Gradle as its build-tool and Jenkins-CI server as a build and delivery server.

The focus of this paper is the standalone build variant. This service enables the user to transform any Pocket Code project to an APK which can then be released on app stores. The configuration, e.g., which Pocket Code project is to be transformed, which icon, which app ID and key for signing shall be used, is set by the user through the web-based interface.

4.2.1. Pocket Code

Pocket Code (initial name Catroid) is a project of the Catrobat organization[1]. It is a visual coding framework for Android and iOS and allows users to develop games, animations, and other apps easily by dragging code blocks together instead of writing scripts [120]. These code blocks are called "bricks". Pocket Code supports its users to learn how to coding and create apps in a very short time with little or no programming experience and install these apps directly on their mobile devices [120]. Furthermore, it is designed on localization and internationalization software design principles. These help users better understand and comfortably use the app in their chosen language [7]. It also interfaces with the Catrobat sharing community where a user can directly upload their own or download others projects without opening the community website in an extra browser. Pocket Code is also released with custom features for their partners and projects such as Create@School[2] and Phiro[3].

4.2.2. Catrobat share community

Catrobat runs a web-platform where users can share their Pocket Code projects. The Catrobat share community is based on the philosophy of "sharing the learning" [120]. A user can view the source code of any project, download and remix them for improvement and learning. Users can transform any of these projects into an Android application package (APKs) file. The APK file format is used by the Android operating system for distribution and installation of mobile apps [63]. User projects as APKs can be installed on Android devices and executed without the need of the Pocket Code IDE.

4.2.3. Gradle

Gradle is an open source build management system based on Groovy [126]; it was introduced in 2013 as the preferred build system for Android apps. When Android introduced the Gradle plugin for Android and Android Studio [33], they focused on making it easier for code reusability,

[1] https://www.catrobat.org/
[2] https://edu.catrob.at/
[3] https://www.phiro.science/

build-variant creation, and build process configuration and customization [33]. Besides, they introduced easy IDE integration while keeping the build system independent from the IDE. In case the developer runs Gradle from the command line or on a continuous integration system, e.g., Jenkins, the results will be the same as running a build from Android Studio.

4.2.4. Jenkins

Jenkins is an open source continuous integration and build automation tool, which can be used to automate all parts of software development where no human interaction is needed such as building, unit testing, and delivering or deploying software [127]. Jenkins jobs can be triggered manually, by REST API calls or on a pre-defined schedule [127]. A Jenkins job can also receive parameters, which can be used to customize the job [79].

4.2.5. Catroid Repository

Pocket Code (aka Catroid) repository [4] uses git and Github as its version control. "A successful Git branching model" [5] is used as branching model where "master" presents the currently released software and the default "develop" branch reflects the current state of development. When a feature is under development, a branch is created e.g., feature/afeature and on successful completion, it is merged in the main developed branch. All features and build variants reside in one branch. Keeping all build variants in one branch supports maintenance of core functionalities which are used by all build variants. This scalable and efficient approach nevertheless creates a challenging task for the release engineer to release different variants out of same code base with different business requirements.

4.3. Pocket Code build variants

As mentioned above, there are different Pocket Code flavors for, e.g. partners or projects and for standalone build flavors. All different build flavors have different acceptance criteria, additionally there are two build-types (debug and release). When all build-type properties are added to the specific product flavor properties, it is called a build variant.

4.3.1. Pocket Code

This is the main flavor of the IDE with standard features, available in two build variants, i) debug for internal testing, and ii) release for the public (i.e. publishing it on the app store). Features

[4] https://github.com/Catrobat/Catroid
[5] https://goo.gl/4xE91p

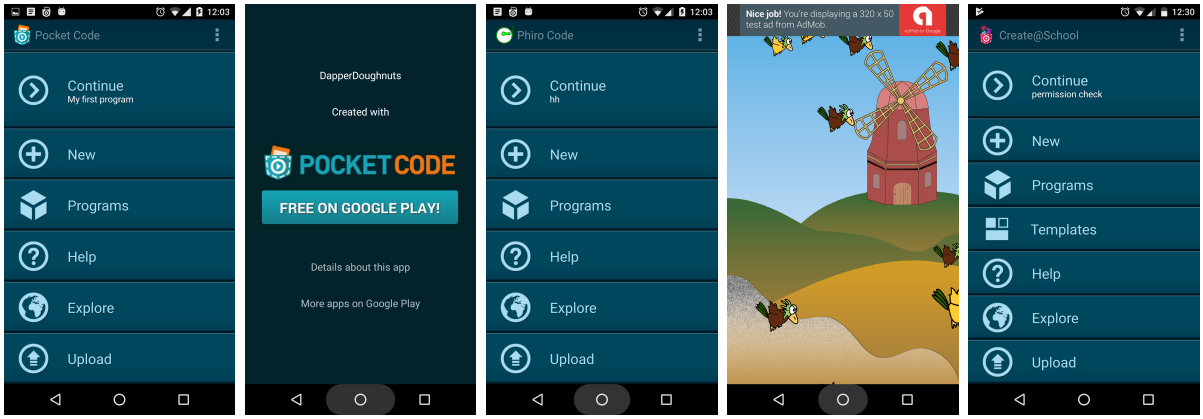


Figure 4.1.: Pocket Code flavors. Pocket Code, Standalone, Phiro, Publishable with Admob, and Create@School.

such as crashlytics and snack bar are not required in a debug variant and must be disabled while building a debug variant.

4.3.2. Create@School

The Create@School flavor is designed with a focus on schools. It is available on Google Play as an independent app with its own application name and launch icon. It has all common functionalities of Pocket Code with some additional school-related features. To correctly build this variant all required features must be included, the package name and the app name adjusted as well as a different launch icon configured. Only then this flavor can be released,installed and used in parallel with Pocket Code.

4.3.3. Phiro

The Phiro flavor is solely designed to control Phiro[6] robots wirelessly and is available on Google Play as an independent app. Only Phiro related functionalities are required for this app. The build must not include all other features.

4.3.4. Build-Standalone

This is an atypical Pocket Code build variant which allows users to run their user projects as an Android application (known colloquially as “app”) without the need of Pocket Code IDE. Each app contains its own unique package name, app title and icon. The flavor is available in two variants.

[6] http://robotixedu.com/phiro/

Unpublishable build-standalone

Any user can generate any project as a standalone APK. These apps display the "Made with Pocket Code" banner on the application exit event. Besides, users cannot customize the app properties such as version code, versions name, package name and others. This is to discourage intellectual property theft, so users do not publish someone elses work for reputation or monetary gain.

Publishable build-standalone

The publishable build-standalone is in beta-testing phase and is not available to public. This variant allows project owners to sign and release the Catrobat project as an app on app stores like Google Play without the "Made with Pocket Code" banner. While generating an APK, a user can customize the app properties, and optionally also include mobile advertisements to earn money.

4.4. Building Pocket Code variants

```
buildTypes {
 debug {
  ....
    resValue "string", "SNACKBAR_HINTS_ENABLED", "false"
    ext.enableCrashlytics = false
 }
 release {
    buildConfigField "boolean", "CRASHLYTICS_CRASH_REPORT_ENABLED", "true"
    resValue "string", "SNACKBAR_HINTS_ENABLED", "true"
    ...
 } }
```

Code-Snippet 4.1: Pocket Code build type configuration in build.gradle file

A build variant is a cross product of a build-type and product-flavor [34] [35]. When an Android app is created in Android Studio, by default Gradle creates two build-types (Debug and Release). The following code snippet 4.1 shows Pocket Code build-types.

With the debug build type the snack bar and crashlytics are disabled, whereas, in the release version, they are enabled. The 'buildConfigField' Gradle method creates the 'BuildConfig' Java class with the defined constants of the code snippet while building the APK. The 'resValue' Gradle method creates a resource variable. The 'resValue' generates a resource of the type specified e.g., String into 'res' (Resource) directory and could refer it via XML with '@string/SNACKBAR_HINTS_ENABLED'. These variability mechanisms allow Pocket Code to control specific behavior and pass certain values which are not available in the code base. In section 4.5 we will discuss the 'buildConfigField' method in more detail. The Gradle 'productFlavor' feature allows building different flavors out of the same code base. The following code snippet 4.2 shows

configurations for different flavors. For each flavor a unique appID is assigned, this identified applications and if different allows apps to be installed in parallel with other flavors [37]. Flavor specific code and resources (icon, logo, etc) are placed under its respective folder e.g ./src/[flavor-name], as shown in figure 4.2.

```
productFlavors {
catroid {
appId = 'org.catrobat.catroid'
buildConfigField "String", "START_PROJECT", "\"No Starting Project\""
buildConfigField "boolean", "CRASHLYTICS_CRASH_REPORT_ENABLED", "true"
...
}
createatschool {
appId = 'org.catrobat.catroid.createatschool'
buildConfigField "boolean", "CREATE_AT_SCHOOL", "true"
...
}
phiro {
...
buildConfigField "boolean", "PHIRO_CODE", "true"
...
}
```

Code-Snippet 4.2: Pocket Code build Flavor configuration in build.gradle file

For a specific build variant, Gradle determines from the folder structure which files are to be included in this particular variant. The "main" directory contains all resources and Java code. The resources and code that are located under flavor named folders are only used by that corresponding flavor. Source code and resource files in the flavored folder are handled differently. Resource files, such as strings or layout files, can be overridden easily. They just have to be placed in the same location with the 'same name' to replace the 'main' resources during the build. For example, for a different app logo, the new image just needs to be placed under "./../src/flavor-name/res/drawable-****/ic_launcher.png". This means that 'manifest' settings for a build type overrides the 'manifest' settings for a product flavor. Gradle will throw a "duplicate class" error if a Java class in flavor contains same name as in 'main' folder [35]. This can be handled by making an abstract base-class and extend the class in flavor-specific directory. If a file shares the same name, as depicted in Figure 4.2 the "PrivacyPolicyDialogFragment.java" file extends the file "BasePrivacyPolicyDialogFragment.java" available in the the "main" folder. This makes it easy to include resources which are only required for a particular build, and which changes the apps behavior by extending flavor specific code. For Pocket Code, Create@School, and Phiro flavors, the configuration, app properties and launching icon are predefined in the code base. The build is usually triggered manually, and monitored by release engineer.

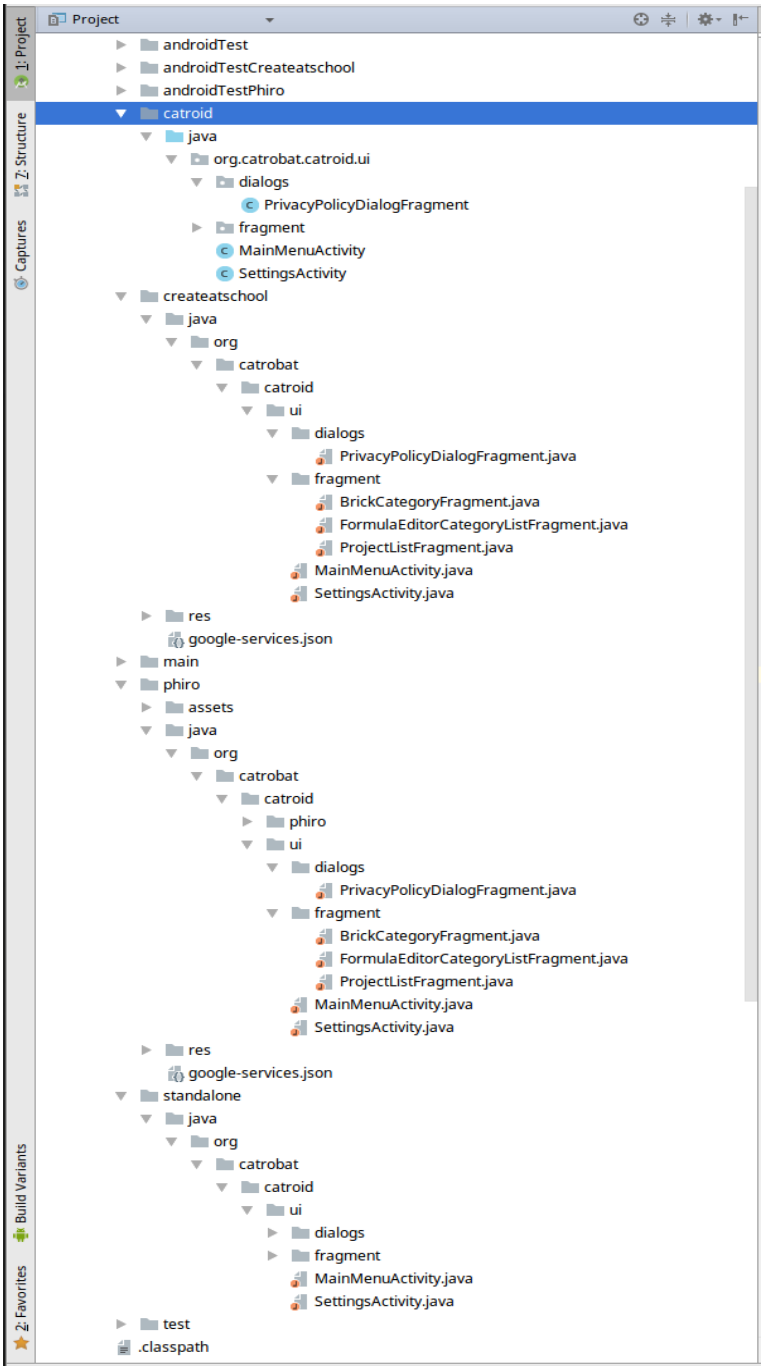


Figure 4.2.: Pocket Code project file hierarchy

4.5. Standalone-Build variant

A Catrobat project can be either executed with the browser-based Pocket Code HTML5 player, or within the Pocket Code IDE. It is also possible to run these projects as standalone apps independent of a Pocket Code installation. To build a standalone version of a project hosted on the Catrobat sharing platform, a user must trigger the build via the web interface on the remote Pocket Code server. When a user clicks "Prepare App" button on Pocket Code community platform for any project, behind the scene, it triggers a job on Catrobats CI server with the configured parameters. For the standalone variant, resources, app properties, app permissions, and launching icon are undefined before the build is triggered by the user. To build a users project as a standalone app, it is necessary to configure system properties and collect or generate the needed resources. With modern CI servers, it is possible to create staged builds, run multiple tasks simultaneously, and aggregate the results [74]. To set app resources and configuration, Jenkins runs a Gradle task to download the project which is to be built and places it in appropriate folders. A user project file contains information about permissions of the app which are required to run it on an Android device. Based on this information it generates an Android Manifest file containing requisite permissions. The Catrobat file also contains an icon of the app, which is extracted and moved to relevant folders. When all resources are generated and in place, Jenkins runs another Gradle task to assemble an APK. The first task dynamically generates resources and the second task builds the APK. Once the build is done, Jenkins deploys the APK on the sharing platform for download.

```
./gradlew -Pdownload="${DOWNLOAD}" -Papk_generator_enabled=true -Psuffix="${SUFFIX}"
    buildStandalone

./gradlew -Pdownload="${DOWNLOAD}" -Papk_generator_enabled=true -Psuffix="${SUFFIX}"
    assembleStandaloneDebug
...
```

Code-Snippet 4.3: Gradle command with parameters from build server

The code snippet 4.3 shows the unpublishable debug variant of the Gradle tasks. The assembleStandaloneDebug task tells Gradle to assemble the standalone flavor using the debug build-type. If an APK file is not digitally signed it cannot be installed on an Android device. When an APK is built as debug, the Android SDK automatically generates a signing keystore file and signs the APK with the debug certificate [64]. To be able to release an app on Google Play, it is necessary to sign it with a personal key. This key must be kept safe for releasing an updated version of the same app [64]. The building mechanism of the publishable variant is the same as for the unpublishable variant. Publishable variants need more information which must be provided by the user. The release build-type is shown in snippet 4.4.

```
gradlew -Pdownload=https://pocketcode.org/download/817.catrobat
    -Papk_generator_enabled=true -Psuffix=org.catrobat.standalone817 -Pversioncode=1
    -PADMOB_APP_ID=ca-app-pub-123 -PADMOB_TEST_DEVICE=xxx -PADMOB_AD_UNIT_ID=ca-app-pub-321
    -PADMOB_DIRECTION=bottom assembleStandaloneRelease
```

Code-Snippet 4.4: Gradle command with parameters for AdMob integration

Configuration information is often modeled as a set of name-value strings. If a value is different from one variant to another variant, it is good to assign the value via a variable. Following code snippet in 4.5 is taken from the build.gradle file which illustrates how in the standalone Gradle build definition variables are used to set configuration for publishable APK. One can also see that the snack bar is enabled in all other release variants except standalone.

```
standalone {
applicationId getPackageNameSuffix()
versionCode getVersionCodeForStandAlone().toInteger()
versionName getVersionNameForStandAlone()
appName = $appName
appIcon = '@drawable/icon'
buildConfigField "String", "PROJECT_NAME", "\"${(String) getProjectName()}\"";
buildConfigField "String", "START_PROJECT", "\"$projectId\""
buildConfigField "String", "PROJECT_NAME", "\"$appName\""
buildConfigField "boolean", "FEATURE_APK_GENERATOR_ENABLED", "true"
buildConfigField "boolean", "FEATURE_STANDALONE_RELEASED", "true"
...
buildConfigField "String", "ROOT_FOLDER", "\"${(String) appName}\"";
resValue "string", "SNACKBAR_HINTS_ENABLED", "false"
//admob data
buildConfigField "String", "ADMOB_TEST_DEVICE", "\"${(String) getAdMobTestDevice()}\"";
buildConfigField "String", "ADMOB_ADMOB_APP_ID", "\"${(String) getAdMobAppID()}\"";
buildConfigField "String", "ADMOB_UNIT_ID", "\"${(String) getAdMobUnitID()}\"";
buildConfigField "String", "ADMOB_DIRECTION", "\"${(String) getAdMobAdDirection()}\"";
}
```

Code-Snippet 4.5: Pocket Code Standalone build Gradle configuration

In case of a publishable variant, a custom package name for the app is allowed, and optionally AdMob integration. The Gradle task receives the needed signing and AdMob configuration information as parameters. The build.gradle code shows all properties are set as key-value pairs.

Changing variable values is less risky than changing source code [74]. Sensitive or personal data such as AdMob information or credentials must not be placed into the source code, especially in open source software. It is good to use a variable which are set prior to compilation and used at compile-time.

The 'buildConfigField' Gradle method creates the 'BuildConfig' class with constants while building the APK. The constants created with the 'buildConfigField' method are used in the application source code as shown in code snippet 4.6. To control app behavior, properties, and resources the Pocket Code product line uses variability mechanisms. *Conditional Execution* is used to control software behavior and *Module Replacement* to include or exclude resources.

```
if (BuildConfig.FEATURE_APK_GENERATOR_ENABLED){
...
MobileAds.initialize(this, BuildConfig.ADMOB_ADMOB_APP_ID);
.addTestDevice(AdRequest.DEVICE_ID_EMULATOR)  .addTestDevice(BuildConfig.ADMOB_TEST_DEVICE)
...
}
```

Code-Snippet 4.6: Admob inegration and conditional behavior handling

4.6. Conclusion

This paper showed our approach to building Pocket Code, an Android applications in different variants. As of April 2018, according to our records, 52,055 user projects are available on Catrobat sharing platform. Since standalone generation is available on the Catrobat sharing platform, users triggered 42,806 project conversions into APKs. In total these generated APKs were downloaded 205,576 times. These figures show that our approach is well accepted and there is a great demand for conversion of Pocket Code projects to APKs. For the experimental purpose of our publishable standalone apps, three Catrobat projects were already converted to APKs and published on Google Play. These projects are "Space Portal"[7], "Phiro Play"[8], and "Tic-Tac-Toe Master"[9]. Once the releasable APK generation feature will be public, it will potentially bring many interesting apps to the market.

[7] http://catrob.at/gpsp
[8] http://catrob.at/PhiroPlay
[9] http://catrob.at/gptttm

Chapter 5

Streamlining mobile app deployment with Jenkins and Fastlane in the case of Catrobat's Pocket Code

This chapter was originally published in the the Proceedings of IEEE International Conference on Innovative Research and Development (ICIRD) , pp. 1-6. IEEE'18 *[88].*

5.1. Abstract

This paper describes how we improved speed and reliability for deployment in the case of Catrobat's Pocket Code, a mobile open source project with over 500 contributors and 28k active installs, by moving to continuous deployment. Pocket Code is a mobile app supporting multiple languages including right to left languages such as Arabic, Farsi, and Urdu. This leads to additional repetitive tasks during deployment. The main challenge of a transition to continuous deployment is acceptance tests done by product owners, which in our case, take place as a step during deployment and lead to overall deployment prolongation. Another challenge is the translated application descriptions for the app store for all supported languages which lead to a huge amount of repetitive tasks. Creating screenshots for these languages is tedious and error-prone and further, prolong the deployment. This paper describes how we used Fastlane, a mobile app release framework, in conjunction with Jenkins, a continuous integration server, to improve app deployment in terms of speed and reliability. Deployment steps which are not automatable are moved out of the actual process which is supported by the staged deployment approach of Google Play. The presented approach was also successfully tested with Pocket Paint, another Catrobat app on Google Play, which shows it can be easily transferred to fit other apps supporting multiple languages.

5.2. Introduction & Background

5.2.1. Catrobat Project

Catrobat[1] is a visual programming language and a set of creativity tools for different platforms and devices. It is an independent free and open source software (FOSS) project which is hosted on GitHub. All contributors are volunteers from more than 20 countries, working on design, development and translation of the Catrobat apps. Development is done using agile methods, like extreme programming [120] and its underlying principles. The focus of development is Pocket Code for Android and iOS platforms. According to our statistics Pocket Code (for Android) currently has over 437k downloads in total and over 28k active installs.

5.2.2. Pocket Code

Pocket Code is an integrated development environment (IDE) for the brick based visual language Catrobat. It is designed for Android and iOS platforms. Pocket Code is also released with custom features for partners and projects like Phiro[2] or Create@School[3] with internationalization (i18n) and localization (l10n) support.

5.2.3. Continuous practices

Continuous practices [117] are emerging software development industry practices based on agile methods mitigating the gap [49] between development and deployment (Continuous deployment), business and developer (bizDev) and developer and operation (DevOps). Shorter feedback loops between developers and customers improve the product's quality. Frequent releases lead to increased developer confidence, improved customer satisfaction and bonding [85, 117].

5.2.4. Deployment pipeline (DP)

A deployment pipeline is a way to progress through the release process in stages [74, 52]. Usually, the first stage of the pipeline is to download the latest codebase from the repository to build the binaries for further use [52]. Later stages could be automatic or manual depending on business needs or tools and technology limitations where human authorization or input is required. The last stage is usually deploying software to the production environment. However, with a manual deployment pipeline there are still the following challenges to cope with:

- Dependence on persons with tacit deployment knowledge. If the person who usually deploys the app is on vacation - who takes over?

[1] https://catrobat.org
[2] http://www.robotixedu.com
[3] https://edu.catrob.at/no1leftbehind-for-teachers

- Often the documentation of manual deployment steps is not up to date with the actual process. This raises the chance of errors during deployment, especially in the case when the responsible person changes.
- Since manual deployment usually needs a long time, minor bug fixes are not regarded by the deployment or render the current deployment obsolete depending on the criticality of the bug.

Automatic deployment eradicates all boring, repetitive tasks, significantly reduces release time, and enables one to release reliably without manual interaction. Automatic deployment steps are written as code and do not require additional documentation and can be triggered by any authorized team member.

5.2.5. Release strategy

A release could be a new app or an upgrade with new or modified features [98]. A release also could be a bug fix or a refactored version with better performance. Nayebi et al [98] suggest that "an app's release strategy is a factor that affects the ongoing success of mobile apps". For free open source software (FOSS) projects release strategies can be classified as time or feature based [96]. Whether time based or feature based, [23] stated that there is increasing interest in adoption of frequent releases in FOSS projects. Frequent releases imply a limited amount of new code which reduces the risk of errors [47]. There are two main motivations for adoption of a frequent release approach, a) the increase of project attractiveness, and, b) maintenance and the increase of market share [23]. Google and Admob published in March 2014, that the number of users who have stopped using an app, because it was not localized properly, varies between 34% and 48% depending on the origin of the data (United States; China; Japan; United Kingdom and South Korea) [62]. Therefore multi language apps in combination with a frequent release approach need deployment automation. The consumer IT market is rapidly growing [67] due to new hardware, services and platform development. Consumers have plenty of choices to pick an app for their business or personal activity. Considering the velocity of the IT market change, software development companies need to pay special attention to what consumers want [67]. Releasing software faster than competitors is also an important success factor [40].

5.2.6. Localization and internationalization

Localization (l18n) and internationalization (l10n) in the current software market is considered as an important factor to attract users around the globe [8]. Users feel more comfortable and productive if the application is translated to the users' language and reflect their cultural values. Pocket Code is designed by following localization and internalizations design principles. It has the capability to easily adapt to different languages including right to left languages (Arabic,

Urdu, Farsi etc). To localize a product, it needs translation by professional translators. There are many user friendly desktop and online applications to allow translators to contribute. They are capable to export translations in different formats.

Pocket Code uses the Crowdin localization management platform. It is free[4] for open source and academic projects. It facilitates translators and managers to complete the translation job in a reasonable time. Crowdin provides a RESTful API over HTTP using GET or POST to up- and download files and web-hooks to integrate with GitHub[5] and other source code management platforms. Catrobat has more than 500 contributors on Crowdin who translate Pocket Code into various languages. Currently Pocket Code supports more than 47 languages (partially) and displays application details in 26 languages on Google Play Store[6].

5.2.7. Continuous integration and deployment tool support

Within the Catrobat project, Jenkins-CI is used for continuous integration of the Android specific platform applications, and Fastlane for continuous deployment.

Jenkins

Jenkins[7] is a free and open source tool for build automation. It facilitates frequent building and testing of software projects either triggered manually, by external events like GitHub pull requests or on a preconfigured regular basis. Depending on the configuration and installed extensions, Jenkins can be used as a mere continuous integration, as a continuous delivery or as a full blown continuous deployment tool.

Fastlane

Fastlane is a free open source tool to automate the deployment pipeline of Android and iOS apps. It handles all monotonous task such as taking screenshots. When screenshots are created manually usually different people are required who are capable of understanding and operating the device in the various supported languages. With Fastlane, this can be done automatically. Furthermore, signing and uploading the app to the app stores is taken care of by Fastlane as well.

5.2.8. Challenges in Catrobat's Pocket Code deployment

Human factor researchers are increasingly concerned with developing tools for handling critical acts [109]. Automation improves performance with Fastlane, reliability, availability, and productivity hence it saves time and money. The main challenges in Catrobat's Pocket Code deployment

[4] https://crowdin.com/pricing
[5] https://support.crowdin.com/github-integration
[6] https://goo.gl/SSJkQj
[7] https://jenkins.io

are a.) the many languages Pocket Code supports and which have to be reflected by the app store descriptions including screenshots, b.) the acceptance tests by the product owners which currently are done during deployment preparation which delays the actual deployment and c.) the manual steps the release responsible person has to fulfill to set up the environment to build, sign, align the APK (Android Package Kit, i.e. Android application package) as well as test and copy the release candidate for actual upload to the app store. This is usually done as teamwork and all team members depend on each other. One of the responsible development team members creates the release branch, after that a senior member signs, aligns and uploads the APK to our internal cloud for acceptance testing by the product owners. On final approval, an authorized member uploads the APK to Google Play. App description translations and screenshots are usually not updated frequently since this is tedious and monotonous manual work. The downside of this behavior is that the APK and the description with screenshots diverge with the time. In the following sections, we describe the status quo and the transition to continuous deployment.

5.2.9. Challenges using Fastlane and Crowdin

For using Fastlane and Crowdin in conjunction some adaptations to file-structures and directory naming have to be implemented. The Fastlane Screengrab tool is capable of capturing screenshots by changing system locals and retrieving them from the emulator or device for further processing. Screengrab uses its folder naming convention as "languageCode-CountryCode", e.g., en-US, en-UK, de-DE, ur-PK (see source code of Screengrab[8]). Crowdlin export feature offers different custom naming conversation for its directory structure and export all languages as zip file containing separate folders for each language. Each folder contain an XML file google_play.xml with four properties "title" "description", "promotion_text" (i.e. the short description), "app_updates" ("What's new" section which is not yet maintained with Pocket Code).

At the moment Pocket Code offers app localization and internationalization for 57 languages including those languages which are not supported by the Android system for example Sindhi, and Pashto. Furthermore, Pocket Code supports different dialects such as French African and French French which are not supported by Google Play. Since Crowdin is not aware which languages and dialects are supported by Google Play the language export contains all available translations including those which cannot be uploaded to Google Play but are used in Pocket Code.

Currently, Google Console offers app listings in 78 languages[9] but not in a uniform way. It uses only language code for some languages and languagecodeCountryCode for others, e.g., "ar" for all Arabic languages, "hr" for Croatian, "ca" for Catalan but "csCZ" for Czech, "enUS" for English United States, "enUK" for English United Kingdom.

[8] https://goo.gl/kbhSTV
[9] https://support.google.com/googleplay/androiddeveloper/answer/113469

5.3. Catrobat Deployment status quo

Releasing an app requires careful planning. Failures in the released software or any mistakes during the release process are problematic not only for the organization's reputation and budget [44] but also for users. In the Catrobat project, the actions for deployment are potentially automizable although up until now it was not considered as the most important venture and hence postponed. The Catrobat project uses git and GitHub as its versioning system. Currently, a branching model is used which follows the example of "A successful Git branching model" with a master branch which reflects the status of the currently released project and a default development branch. Whether this is the ideal solution for the project is subject to discussion but as of now it is established and accepted by the development. Catrobat's deployment phase starts after feature development has been finished, the code was reviewed, integrated and considered as potentially shippable due to product owner feature acceptance. The deployment steps are most of the time very similar between releases but have not been automated yet since there are still manual interactions in the workflow (see Figure 5.1).

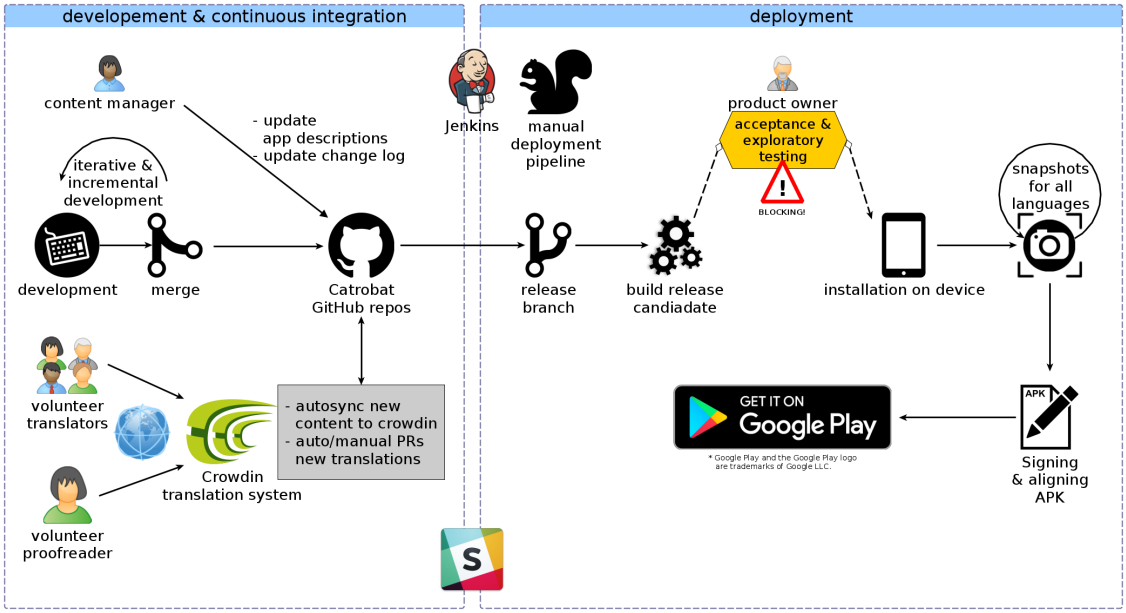


Figure 5.1.: Catrobat manual deployment workflow

When all features are finished and accepted which happens in our continuous integration workflow phase, the person who is responsible for the release will create a release branch from the development branch. The development branch is in a state with potentially shippable code only,

until the release branch was branched off. This release-branch is once again tested on Jenkins and all automated test (lint, PMD, unit, integration, and acceptance) are executed just to ensure no breaking code made it to the release branch. When there are no errors, the outcoming APK is an artifact which is potentially shippable. This APK is uploaded to our internal Wiki along with the release notes for final acceptance by our product owners. After thoroughly manual testing predefined scenarios and also in an exploratory manner, this release candidate is accepted as a whole by the product owners. If there have been fundamental changes in the UI (user interface) then the latest screenshots for the app-store description have to be captured manually for all languages. The next step is the signing and aligning of the APK. After this is done, the last step is to upload the APK with the language dependent descriptions and screenshots to the app store. Finally, the release is announced via our internal communication channels. These steps are well documented in our internal Catrobat Wiki but they are subject to frequent optimization changes since the team works on improving and streamlining the workflow to reduce the chances of errors and mitigate tedious deployment steps. Nevertheless, humans are not good at repetitive tasks [131] and the deploy process is error-prone especially if the number of repetitive steps increase due to, e.g., more different supported languages, and different flavors. Especially senior level member quickly become bored and start to make errors due to repetitiveness [28, 45].

5.3.1. Rapid increase of manual steps

Since Pocket Code exists in different flavors (special featured versions for partners) these flavors have to be released separately in the above described manner. Furthermore, the plan to support different app stores in future and the rising number of supported languages lead to an explosion of the number of manual steps which poses a real problem. This can only be alleviated by automating as much as possible and removing manual intervention out of the deployment phase.

5.4. Streamlining - moving towards CD

The goal of continuous deployment is to eliminate all manual steps in the deployment phase and automate as much as possible to minimize errors introduced by human intervention. According to our workflow depicted in Figure 5.1 the only blocking activity is the final approval of the product owner. This must be moved out of the deployment phase. With automatic app deployment (see Figure 5.2) there are no drawbacks in moving this approval after the deployment has happened, of course only under the premise that this APK does not reach the public without thoroughly testing and final approval. The Google Play Developer API helps in this regard allowing one to upload new APKs of an app to different release tracks,. The following tracks are available by default:

- **Alpha** the track where only alpha testers are subscribed (e.g., product owner).

- **Beta** the track for a limited number of beta testers.
- **Rollout** this track reaches a defined percentage (range from 5% to 50%) of the app's users which are randomly selected.
- **Production** this track is to publish the app for all users.

For the Catrobat project, only alpha and production tracks are currently used. On the alpha track, the product owners are registered who are informed about the deployment. They now have the opportunity to install the app via Google Play. This is a definite advantage over the installation by hand, where one must copy the APK to the device and manually install it. When the product owners approve this version a job can be triggered on Jenkins to finalize the deployment and move this version to the production track and upload the app description and screenshots for all languages. If there is any problem, either during final approval or during automatic deployment the central point of communication is the Catrobat's slack infrastructure with its various channels. In the best case, the deployed APK is moved from alpha to production channel. Otherwise, the errors are communicated to be fixed by the developers, translators or designers. The following steps have been automated with Jenkins/Fastlane. The best case deployment boils down to triggering two events - the trigger for automatic deployment to the alpha track and the final approval, i.e. triggering promotion from alpha to production including the upload of the updated app metadata including screenshots for all languages.

5.4.1. Releasing to alpha

The Jenkins job "deploy to alpha track" sets up the environment and clones the development branch, which contains releasable code, to its workspace. Then it builds the release and debug APKs, and runs all necessary checks and tests (LINT, PMD, UI) using the Android emulator.

5.4.2. Signing and aligning APK

Once the APK is built, Jenkins executes commands to sign and align the APK. A signing certificate is required which is securely stored by the credential plugin of our Jenkins server. Aligning an app ensures that all uncompressed data such as images, raw files start with a particular alignment relative to the start of the file. This approach reduces RAM consumption and allows direct access of all portions even if they contain binary data with alignment restrictions. It is recommended to always use zipalign before distributing APK to end-users [31]. All android apps need to be digitally signed with a certificate in order to distribute via Google Play [32]. It is important to always sign all versions with the same certificate.

5.4.3. Screenshots for all languages

This part of the deployment is one of the most interesting improvements in terms of speed and reliability. The challenge is to put the app for all different languages into the same configuration and then take screenshots. This is not only a time consuming task but it is very error-prone especially if the one who has to do the work does not understand the language the screenshots have to be taken. Furthermore, the screenshots have to be downloaded from the device to be used for the app description. After all previous tests passed, Jenkins runs the Fastlane screengrab tool to capture screenshots in all languages for the app description on Google Play. Fastlane Screengrab tool automatically changes the system language and captures screenshots in the provided Espresso test package. These "screeengrab"-Espresso tests have the only purpose to navigate the app to the desired configuration and then capturing the screen. These tests are held very simple since there is no functionality to test. Usually, they are created during development. Via the package structure, one is able to combine certain tests to meaningful collections.

5.4.4. Combining Screengrab and Crowdin

Extra efforts have to be made to rename the language folders according to the Google Play Console listing format which leads to removal of unsupported languages such as Urdu and Sindhi. Fastlane deployment pipeline accepts title, full description, short description, and app update information as separate text files ("title.txt", "full_description.txt", "short_description.txt" and "whatsnew.txt") in each language folder. Pocket Codes deployment pipeline downloads all translations from Crowdin via Crowdin console client as a zip file. This archive contains language folders with following naming convention: "languagecode-CountryCode" (e.g. "en-US", "ar-SA", "sd-PK"). Crowdin's naming convention is not compatible with the Google Play naming convention for languages. The first step of this stage separates the "google_play.xml" file and splits up the content into three separate text files which are needed by Fastlane, "full_description.txt", "short_description.txt" and "title.txt". The next step merges the Crowdin folders containing the translation files (three files) with the screenshots folders created by Fastlanes Screengrab. The third step renames these folders according to the Google Play naming convention, e.g., "ar-SA" to "ar", "bg-BG" to "bg" and deletes folders of languages which are not supported by Google Play, such as "sd-PK" and "ur-PK". Pocket Code currently supports 57 languages whereas Google Play does not support 10 languages Pocket Code does, hence they have to be removed prior to the upload to Google Play. Therefore Pocket Code on Google Play displays the app descriptions and screenshots for only 47 languages.

5.4.5. Creating the release branch

After signing and aligning of the APK, the workspace contains everything needed for deployment. Code, APK, screenshots, changelog and app description in different languages. A release branch is created, all artifacts are committed and pushed to the release branch.

5.4.6. Deploy to alpha channel

Once the release candidate pushed to the release branch, it is also uploaded to Google Play. At this stage Jenkins uses Fastlane's supply tool to upload the APK to the alpha track without any metadata since Google Play would automatically publish the metadata. This must not happen until the app is promoted to the production track by the product owners.

5.4.7. Post build actions

Catrobat uses Slack for project communication. For every deployment build Jenkins posts notifications about failure or success of the job.

5.4.8. Product owner approval

The product owner (PO) app approval is the most unpredictable event in terms of time in the deployment pipeline and it cannot be automatized. Due to a staged deployment approach supported by the Google Developer API the app acceptance and exploratory tests are postponed until the technical deployment to the alpha track is finished. In the Catrobat project the alpha track is subscribed by POs only, so the app is only available for then in this stage.

5.4.9. Release to production

Once the product owners have approved the release candidate from alpha track, a Jenkins job is triggered which consists of two main steps 1) uploading description and screenshots from the release branch to Google Play, and 2) Promotion of the APK from alpha to production track.

5.5. Deployment time

Manual deployment as it is implemented according to Figure 5.1 suffers from four main drawbacks, a.) the manual overhead human interaction poses, e.g., task switching, b.) the creation of

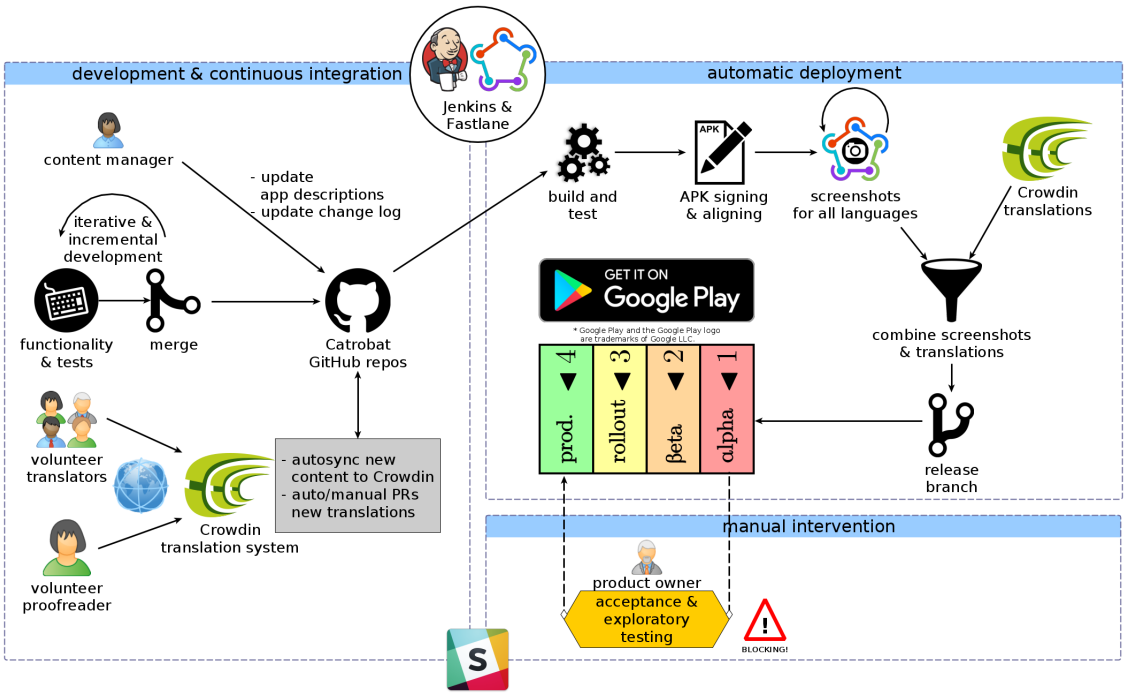


Figure 5.2.: Catrobat automatic deployment workflow

screenshots, c.) setting of translations, and d.) product owner approval. Whereas d, the product owner approval due to a staged approach can be postponed after deployment to the alpha track. Drawback b, c, and d have the most impact on deployment time and can be alleviated via automation and pipeline restructuring. According to the release responsible person, a manual deployment to the app store needs about 25-35 minutes without running tests (lint, PMD, unit, UI) and updating descriptions or screenshots. Running the tests adds 15 minutes to the deployment time. Due to the tedious process of creating screenshots manually, in reality, the app descriptions are updated only when the gap between the released UI and the published screenshots become evident. This leads to a situation where description and screenshots are obsolete which can lead to confusion on the user side. The time needed for screenshots depends on i.) which languages the screenshots are made, ii.) how complicated the setup of the app is until the actual screenshot can be taken. For instance, if one wants to create screenshots of Pocket Code scripts with variable names, these have to be translated too otherwise one ends up with screenshots for, e.g., Thai and with German variable names. The magnitude for the time needed for screenshots in different languages was empirically determined with eight different languages including Japanese, Chinese, and Arabic. It turned out that six screenshots per language could be manually created within 3 minutes in average, independent of the language. During the creation of the screenshots in eight languages only one error was introduced, thanks to the simple app setup which was used. The experiment further showed, as soon as a manual error was introduced the fixing of the app's con-

figuration tripled the amount of time and it took almost nine minutes to finish the six screenshots for this language. It is safe to assume that errors increase as soon as the configuration grows complexity in conjunction with the used languages. For instance, a non Arabic speaking person doing the screenshots, in the case of an error is completely lost in the UI and has to switch back to the native language to fix the configuration and then switch back to Arabic to complete the screenshots for this language. Assuming, no errors are made during taking the screenshots and the setup of the app is simple, the overall time for the 26 languages are at least 78 minutes. All screenshots need to be placed in respective folders of their language, e.g., Arabic screenshots to the Arabic language folder and Chinese in Chinese language folder, etc. This is not an easy job as one must understand which language screenshot must be put and this is not obvious for a person who does not know the languages and the differences between, e.g., Arabic or Farsi. In contrast the automatic screenshot creation for all 26 languages with six screenshots each, takes less than 10 minutes (545 seconds) including setting the desired UI, capturing, downloading and storing screenshots in respective folders. Product owner approval can be neglected in the comparison between manual and automatic deployment. There is nothing to be automated or shortened. In the case of the Catrobat project, overall app approval by the product owners takes up to two days. When deployed manually this leads to a delay until the app reaches the app store. In the staged approach, this approval is done after upload to the alpha track, where the app can be promoted within seconds to production. For time comparison between manual and automatic deployment, running the automatic tests, create screenshots and uploading to the app-store is considered. Manual deployment adds up to 2hr 8 min (15 min automatic tests, 35 min manual intervention, 78 min screenshot creation). Automatic deployment adds up to less than 25 min (14 min automated tests, 10 min creating screenshots, less than 1 min uploading to app store) this means a saving of 1 hour and 43 minutes. From the first release in 2013 up until 2018 Pocket Code was manually released 42 times to Google Play. Accumulating the delta between manual and automatic deployment 72 hours have been wasted. This might not seem too impressive but the gained accuracy, flexibility and readiness to deploy any time with only two mouse clicks, as well as, no humans are bored with repetitive tasks are the true benefits of this approach.

5.6. Conclusion

Using Jenkins in conjunction with Fastlane, Crowdin and a staged deployment approach makes it possible to shrink deployment time significantly. It can be triggered on demand and repeated as often as needed hence fostering frequent releases. Furthermore, it relieves the release responsible person from manual tasks, increases the accuracy of the process especially the creation of screenshots in different languages for the app's metadata and reduces errors introduced by manual intervention. In case of the implementation of automated deployment in the Catrobat project manual interactions are reduced to two mouse clicks for releasing the app to production. First to

trigger the pipeline to create all metadata and deploy the app to alpha track and second to trigger promotion of the app to production and publish the app's metadata. With Jenkins and Fastlane it is feasible with reasonable effort to automate the deployment of mobile applications. The most interesting part of this approach is the automatic creation of app screenshots for different languages, setting translations and dealing with compatibility issues between different tools. This is especially interesting for all app providers who strive to deliver multi-language apps to increase market share.

Chapter 6

Conclusions

The main contributions of this thesis can be summarized as follows with recommended future work.

6.1. Summary

We conducted the study to analyze the impact of the internationalized and localized feature on an application. The study carried out a user experience test. The UX test showed that design changes led to improved simplicity, conciseness, and increased appeal.

We developed the new feature that allows users to release their project on Google Play, with an option to earn by integrating Google AdMob program in their project. Introduced Pocket Code software product line assemble resources, app properties at compile time.

We presented the Pocket Code deployment pipeline. The staged deployment approach using Fastlane, Crowdin and Jenkins build server significantly reduces the deployment time. The deployment of a new feature or a bug fix can be triggered as soon as it is ready for release. Automation in deployment decreases release time, increases the accuracy of and confidence in the process. Furthermore, by limiting repetitive and error-prone tasks the deployment activity becomes more engaging and therefore boosts the self-esteem of release-responsible-persons. The two click deployment pipelines obsolete all human efforts of setting test environment or deliverables. The first click facilitates product owners to get the latest binaries automatically via Google Play alpha track. The second click uploads the latest app's metadata to Google Play and promotes the app to production for that it becomes publicly available. An interesting part of our deployment pipeline and where most of the time is saved is the automatic capturing of app screenshots and the handling of compatibility issues which usually arise when different tools are used in conjunction. The Pocket Code deployment pipeline approach can be of interest to all app providers who publish multi-language apps and strive to maintain an internationalized and localized Google Play

presence. This approach supports the organization to keep its deployment efforts low and focus on development. By automating repetitive deployment tasks this approach helps to properly present multi-language apps, reach a broader audience, and potentially increase market share.

6.2. Future work

To continue research in thesis direction some areas remain open

The study was carried out while Pocket Paint was under beta version with a very small number of participants. Another study is recommended to observe if multi-language support in the Catrobat project helps increase its user-base. Monitoring Google developer console data can be helpful to learn whether the language support on app level leads to more people installing and using the application in the specific regions of the world.

The Standalone flavor excludes many permissions and features which are not in use by the generated application. However, still, it requires more optimization to reduce application size. The source code refactoring and properly productFlavors function utilization can separate Pocket Code UI and runtime. Once runtimes are identified and separated, the standalone can exclude UI related resources and code. On the organizational side, respect for intellectual property in open source projects and licensing issues should be handled carefully.

To validate introduced deployment pipeline the author intends to conduct a survey to gauge stakeholder- satisfaction on new workflow and to explore any new suggestions such as mandatory tests and how to automate them, developing a general solution to support more app distribution platforms.

Bibliography

[1] Visualizations of continuous delivery - continuous delivery. https://continuousdelivery.com/2014/02/visualizations-of-continuous-delivery/. (Accessed on 07/31/2018). (Cited on pages xix, 6, and 7.)

[2] ABRAHAMSSON, P., SALO, O., RONKAINEN, J., AND WARSTA, J. 2017. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*. (Cited on pages 3 and 4.)

[3] ABUFARDEH, S. AND MAGEL, K. 2008. Software localization: the challenging aspects of arabic to the localization process (arabization). *IASTED Proceeding of the Software Engineering SE*, 275–279. (Cited on page 40.)

[4] ALDERETE, M. V. 2017. Mobile broadband: A key enabling technology for entrepreneurship? *Journal of Small Business Management 55,* 2, 254–269. (Cited on page 34.)

[5] ANJA BALANSKAT, K. E. 2015. Computing our future: Computer programming and coding - priorities, school curricula and initiatives across europe. Tech. rep., European Schoolnet. (Cited on page 18.)

[6] ANUSHREE VERMA. 2016. 5 good reasons why agile should be applied - dzone agile. https://dzone.com/articles/you-will-thank-us-5-good-reasons-why-agile-should. (Accessed on 07/17/2018). (Cited on page 4.)

[7] AWWAD, A. M. A., SCHINDLER, C., LUHANA, K. K., ALI, Z., AND SPIELER, B. 2017a. Improving pocket paint usability via material design compliance and internationalization & localization support on application level. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM, 99. (Cited on pages 22 and 51.)

[8] AWWAD, A. M. A., SCHINDLER, C., LUHANA, K. K., ALI, Z., AND SPIELER, B. 2017b. Improving pocket paint usability via material design compliance and internationalization & localization support on application level. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM, 99. (Cited on page 63.)

[9] AWWAD, A. M. A., SCHINDLER, C., LUHANA, K. K., ALI, Z., AND SPIELER, B. 2017c. Improving pocket paint usability via material design compliance and internationalization & localization support on application level. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*. MobileHCI '17. ACM, New York, NY, USA, 99:1–99:8. (Cited on pages 1 and 37.)

[10] AYYAL AWWAD, A. M. AND SLANY, W. 2016. Automated bidirectional languages localization testing for android apps with rich gui. *Mobile Information Systems 2016*. (Cited on pages 40 and 41.)

[11] BARENDREGT, W. AND BEKKER, M. M. 2003. Guidelines for user testing with children. *Department of Industrial Design, Eindhoven University of Technology*. (Cited on page 37.)

[12] BECK, K. 1999. Embracing change with extreme programming. *Computer 32,* 10, 70–77. (Cited on page 13.)

[13] BECK, K., BEEDLE, M., VAN BENNEKUM, A., COCKBURN, A., CUNNINGHAM, W., FOWLER, M., GRENNING, J., HIGHSMITH, J., HUNT, A., JEFFRIES, R., ET AL. 2001. Manifesto for agile software development. (Cited on page 4.)

[14] BELAGATTI, P. 2016. Benefits of continuous deployment - dzone devops. https://dzone.com/articles/benefits-of-continuous-deployment. (Accessed on 07/19/2018). (Cited on page 8.)

[15] BEUCHE, D., PAPAJEWSKI, H., AND SCHRÖDER-PREIKSCHAT, W. 2004. Variability management with feature models. *Science of Computer Programming 53,* 3, 333–352. (Cited on page 50.)

[16] BEVANS, A. 2017. Who plays mobile games? — gamesindustry.biz. https://www.gamesindustry.biz/articles/2017-06-14-who-plays-mobile-games. (Accessed on 6/4/2018). (Cited on page 22.)

[17] BEZROUKOV, N. 1999. Open source software development as a special type of academic research: Critique of vulgar raymondism. *First Monday 4,* 10. (Cited on page 3.)

[18] BONACCORSI, A. AND ROSSI, C. 2003. Why open source software can succeed. *Research policy 32,* 7, 1243–1258. (Cited on page 3.)

[19] BOWLER, M. 2013. Continuous Delivery : The Ultimate Challenge for Software Development Managers. *PM World Journal II,* I, 1–6. (Cited on pages 2 and 8.)

[20] BURTSCHER, D. 2013. Introduction of a continuous integration process in an open source project. M.S. thesis, Graz University of Technology. (Cited on pages 5 and 6.)

[21] CAPILLA, R., BOSCH, J., TRINIDAD, P., RUIZ-CORTÉS, A., AND HINCHEY, M. 2014. An overview of dynamic software product line architectures and techniques: Observations from research and industry. *Journal of Systems and Software 91*, 3–23. (Cited on page 50.)

[22] CARILLO, K., HUFF, S., AND CHAWNER, B. 2017. What makes a good contributor? understanding contributor behavior within large free/open source software projects–a socialization perspective. *The Journal of Strategic Information Systems 26,* 4, 322–359. (Cited on page 4.)

[23] CESAR BRANDÃO GOMES DA SILVA, A., DE FIGUEIREDO CARNEIRO, G., BRITO E ABREU, F., AND PESSOA MONTEIRO, M. 2017. Frequent releases in open source software: A systematic review. *Information* 3. (Cited on page 63.)

[24] CHEN, L. 2015. Continuous delivery: Huge benefits, but challenges too. *IEEE Software 32,* 2, 50–54. (Cited on pages 6, 7, and 8.)

[25] CHEN, L. 2017. Continuous delivery: overcoming adoption challenges. *Journal of Systems and Software 128*, 72–86. (Cited on pages 7 and 8.)

[26] CLAPS, G. G., SVENSSON, R. B., AND AURUM, A. 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology 57*, 21–31. (Cited on pages 6 and 8.)

[27] COMMISSION, E. 2016. New skills agenda for europe - employment, social affairs & inclusion - european commission. http://ec.europa.eu/social/main.jsp?catId=1223. (Accessed on 01/03/2018). (Cited on page 18.)

[28] DAVIS, J. AND DANIELS, K. 2016. *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. " O'Reilly Media, Inc.". (Cited on pages 6 and 67.)

[29] DE CESARE, S., LYCETT, M., MACREDIE, R. D., PATEL, C., AND PAUL, R. 2010. Examining perceptions of agility in software development practice. *Communications of the ACM 53,* 6, 126–130. (Cited on page 6.)

[30] DESHPANDE, A. AND RIEHLE, D. 2008. Continuous integration in open source software development. In *IFIP International Conference on Open Source Systems*. Springer, 273–280. (Cited on pages 2, 5, and 6.)

[31] DEVELOPER, G. 2016. zipalign. accessed: 2017-12-16. (Cited on page 68.)

[32] DEVELOPER, G. 2017. Sign your app. accessed: 17th December, 2017. (Cited on page 68.)

[33] DEVELOPER, G. 2018a. Android plugin for gradle release notes — android studio. https://developer.android.com/studio/releases/gradle-plugin.html. (Accessed on 04/25/2018). (Cited on pages 51 and 52.)

[34] DEVELOPER, G. 2018b. Configure build variants — android studio. https://developer.android.com/studio/build/build-variants.html. (Accessed on 04/25/2018). (Cited on page 54.)

[35] DEVELOPER, G. 2018c. Configure your build — android studio. https://developer.android.com/studio/build/index.html. (Accessed on 04/25/2018). (Cited on pages 54 and 55.)

[36] DEVELOPER, G. 2018d. Create an android library — android studio. https://developer.android.com/studio/projects/android-library.html. (Accessed on 04/25/2018). (Cited on page 50.)

[37] DEVELOPER, G. 2018e. Multiple apk support — android developers. https://developer.android.com/google/play/publishing/multiple-apks.html. (Accessed on 04/25/2018). (Cited on page 55.)

[38] DÜRSCHMID, T., TRAPP, M., AND DÖLLNER, J. 2017. Towards architectural styles for android app software product lines. In *Mobile Software Engineering and Systems (MOBILESoft), 2017 IEEE/ACM 4th International Conference on*. IEEE, 58–62. (Cited on page 50.)

[39] DUVALL, P. M., MATYAS, S., AND GLOVER, A. 2007. *Continuous integration: improving software quality and reducing risk*. Pearson Education. (Cited on page 6.)

[40] DYCK, A., PENNERS, R., AND LICHTER, H. 2015. Towards definitions for release engineering and devops. In *Proceedings of the Third International Workshop on Release Engineering*. IEEE Press, 3–3. (Cited on page 63.)

[41] DZAMASHVILI FOGELSTRÖM, N., GORSCHEK, T., SVAHNBERG, M., AND OLSSON, P. 2010. The impact of agile principles on market-driven software product development. *Journal of software maintenance and evolution: Research and practice 22,* 1, 53–80. (Cited on page 4.)

[42] EBERT, C. 2008. Open source software in industry. *IEEE Software 25,* 3 (May), 52–53. (Cited on page 2.)

[43] ECK, A., UEBERNICKEL, F., AND BRENNER, W. 2014. Fit for continuous integration: How organizations assimilate an agile practice. (Cited on page 4.)

[44] ERICH, F., AMRIT, C., AND DANEVA, M. 2014. Report: Devops literature review. *University of Twente, Tech. Rep*. (Cited on page 66.)

[45] FALLIS, A. 2013. *Effective DevOps*. Vol. 53. (Cited on page 67.)

[46] FANG, M., GERHART, B., AND LEDFORD JR, G. E. 2013. Negative effects of extrinsic rewards on intrinsic motivation: more smoke than fire. *World at Work Quarterly 16,* 2, 17–29. (Cited on page 3.)

[47] FEITELSON, D. G., FRACHTENBERG, E., AND BECK, K. L. 2013. Development and deployment at facebook. *IEEE Internet Computing 17,* 4 (July), 8–17. (Cited on pages 8 and 63.)

[48] FELLER, J. AND FITZGERALD, B. 2000. A framework analysis of the open source software development paradigm. In *Proceedings of the Twenty First International Conference on Information Systems*. ICIS '00. Association for Information Systems, Atlanta, GA, USA, 58–69. (Cited on page 3.)

[49] FITZGERALD, B. AND STOL, K.-J. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software 123*, 176–189. (Cited on page 62.)

[50] FOUNDATION, L. 2017. 2017 state of linux kernel development - the linux foundation. https://www.linuxfoundation.org/2017-linux-kernel-report-landing-page/. (Accessed on 07/04/2018). (Cited on page 3.)

[51] FOWLER, M. 2013a. Continuousdelivery. https://martinfowler.com/bliki/ContinuousDelivery.html. (Accessed on 07/19/2018). (Cited on page 6.)

[52] FOWLER, M. 2013b. Deploymentpipeline. https://martinfowler.com/bliki/DeploymentPipeline.html. (Accessed on 07/15/2018). (Cited on page 62.)

[53] FOWLER, M. AND FOEMMEL, M. 2006. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf 122*, 14. (Cited on pages 6 and 13.)

[54] FUGGETTA, A. 2003. Open source software—-an evaluation. *Journal of Systems and Software 66,* 1, 77–90. (Cited on page 5.)

[55] FUSSBERGER, N., ZHANG, B., AND BECKER, M. 2017. A deep dive into android's variability realizations. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. ACM, 69–78. (Cited on page 50.)

[56] GALSTER, M., ZDUN, U., WEYNS, D., RABISER, R., ZHANG, B., GOEDICKE, M., AND PERROUIN, G. 2017. Variability and complexity in software design: Towards a research agenda. *ACM SIGSOFT Software Engineering Notes 41,* 6, 27–30. (Cited on page 2.)

[57] GEISS, M. 2012. Continuous integration and testing for android. *Berlin Institute of Technology (TU-Berlin)*. (Cited on page 6.)

[58] GHAREHYAZIE, M., POSNETT, D., VASILESCU, B., AND FILKOV, V. 2015. Developer initiation and social interactions in oss: A case study of the apache software foundation. *Empirical Software Engineering 20,* 5 (Oct), 1318–1353. (Cited on page 2.)

[59] GHTORRENT.ORG. 2018. Performance report for catrobat/catroid. http://ghtorrent.org/pullreq-perf/Catrobat-Catroid/. (Accessed on 07/14/2018). (Cited on pages 11 and 12.)

[60] GHUMAN, A., MAHAJAN, J., BHATIA, S., SINGH, J., AND KULKARNI, M. D. 2017. Empowering e-learning with localization. In *E-Learning & E-Learning Technologies (ELELTECH), 2017 5th National Conference on*. IEEE, 1–6. (Cited on page 22.)

[61] GOOGLE. 2014a. Material design is a unified system that combines theory, resources, and tools for crafting digital experiences. Material Design Website. (Cited on pages 39 and 41.)

[62] GOOGLE. 2014b. Share of app users who have stopped using an app because it was not localized properly as of march 2014. Statista - The Statistics Portal, Statista. (Cited on pages 39 and 63.)

[63] GOOGLE. 2018a. Apk expansion files — android developers. https://developer.android.com/google/play/expansion-files.html. (Accessed on 04/25/2018). (Cited on page 51.)

[64] GOOGLE. 2018b. Sign your app - android studio. accessed: 4th April, 2018. (Cited on page 57.)

[65] GOOGLE AND NEWZOO. 2017. Change the game. the world of women and mobile gaming. Tech. rep., Google. (Cited on page 23.)

[66] GREENBERGER, M. 1962. Computers and the world of the future. transcribed recordings of lectures held at the sloan school of business administration, april, 1961. (Cited on page 18.)

[67] HAMUNEN, J. ET AL. 2016. Challenges in adopting a devops approach to software development and operations. (Cited on pages 8 and 63.)

[68] HARS, A. AND OU, S. 2001. Working for free?–motivations of participating in open source projects, 00 (c), 1–9. In *34th Annual Hawaii International Conference on System Sciences (HICSS-34), Havaí*. 25–39. (Cited on page 3.)

[69] HAU, E. AND APARÍCIO, M. 2008. Software internationalization and localization in web based erp. In *Proceedings of the 26th Annual ACM International Conference on Design of Communication*. SIGDOC '08. ACM, New York, NY, USA, 175–180. (Cited on page 1.)

[70] HENNESSEY, B., MORAN, S., ALTRINGER, B., AND AMABILE, T. M. 2015a. Extrinsic and intrinsic motivation. *Wiley encyclopedia of management*, 1–4. (Cited on page 3.)

[71] HENNESSEY, B., MORAN, S., ALTRINGER, B., AND AMABILE, T. M. 2015b. Extrinsic and intrinsic motivation. *Wiley encyclopedia of management*, 1–4. (Cited on page 3.)

[72] HERTEL, G., NIEDNER, S., AND HERRMANN, S. 2003. Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research policy 32,* 7, 1159–1177. (Cited on page 3.)

[73] HOLCK, J. AND JØRGENSEN, N. 2003. Continuous integration and quality assurance: A case study of two open source projects. *Australasian Journal of Information Systems 11,* 1. (Cited on page 4.)

[74] HUMBLE, J. AND FARLEY, D. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education. (Cited on pages 6, 15, 50, 57, 58, and 62.)

[75] HUMBLE, J., READ, C., AND NORTH, D. 2006. The deployment production line. In *Agile Conference, 2006*. IEEE, 6–pp. (Cited on pages 1 and 2.)

[76] IDC. 2016. Forecasted unit shipments of smartphones worldwide from 2016 to 2021 (in million units), by operating system. Statista - The Statistics Portal, Statista. (Cited on page 42.)

[77] KAFAI, Y. AND VASUDEVAN, V. 2015. Hi-lo tech games: crafting, coding and collaboration of augmented board games by high school youth. In *Proceedings of the 14th International Conference on Interaction Design and Children*. ACM, 130–139. (Cited on page 25.)

[78] KAHN, K. 2017. A half-century perspective on computational thinking. *Tecnologias, Sociedade e Conhecimento 4,* 1, 23–42. (Cited on page 18.)

[79] KAWAGUCHI, K. 2017. Parameterized build - jenkins - jenkins wiki. https://wiki.jenkins.io/display/JENKINS/Parameterized+Build. (Accessed on 04/25/2018). (Cited on page 52.)

[80] KIRSHAN KUMAR LUHANA, MATTHIAS MUELLER, C. S. B. S. W. S. 2018. Rock bottom, the world, the sky: Catrobat, an extremely large-scaling and long-term visual coding project relying purely on smartphones. In *Proc. Constructionism Conf.* constructionism2018. (Cited on page 17.)

[81] KUZHELEVA-SAGAN, I. P. AND SUCHKOVA, N. A. 2016. Designing trust in the internet services. *AI & society 31,* 3, 381–392. (Cited on page 39.)

[82] LABS, S. 2018. Testing trends for 2018 — sauce labs. https://saucelabs.com/resources/white-papers/testing-trends-for-2018. (Accessed on 07/12/2018). (Cited on page 5.)

[83] LACHNER, F., NAEGELEIN, P., KOWALSKI, R., SPANN, M., AND BUTZ, A. 2016. Quantified ux: Towards a common organizational understanding of user experience. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*. ACM, 56. (Cited on page 38.)

[84] LEE JR, JERRY, F. ET AL. 2009. *Scratch programming for teens*. Cengage Learning. (Cited on page 22.)

[85] LEPPÄNEN, M., MÄKINEN, S., PAGELS, M., ELORANTA, V.-P., ITKONEN, J., MÄNTYLÄ, M. V., AND MÄNNISTÖ, T. 2015. The highways and country roads to continuous deployment. *IEEE Software 32,* 2, 64–72. (Cited on page 62.)

[86] LERNER, J., PATHAK, P. A., AND TIROLE, J. 2006. The dynamics of open-source contributors. *American Economic Review 96,* 2, 114–118. (Cited on page 2.)

[87] LESSIG, L., CUSUMANO, M., AND SHIRKY, C. 2005. *Perspectives on free and open source software*. MIT press. (Cited on page 3.)

[88] LUHANA, K. K. 2018. Pocket code build variants. In *Innovative Research and Development (ICIRD), 2018 IEEE International Conference on*. IEEE, 1–6. (Cited on pages 13, 15, 49, and 61.)

[89] LUHANA, K. K., SCHINDLER, C., AND SLANY, W. 2018. Streamlining mobile app deployment with jenkins and fastlane in the case of catrobat's pocket code. In *Innovative Research and Development (ICIRD), 2018 IEEE International Conference on*. IEEE, 1–6. (Cited on pages 1, 2, 8, and 13.)

[90] MADEYSKI, L. 2009. *Test-driven development: An empirical evaluation of agile practice*. Springer Science & Business Media. (Cited on page 13.)

[91] MANNILA, L., DAGIENE, V., DEMO, B., GRGURINA, N., MIROLO, C., ROLANDSSON, L., AND SETTLE, A. 2014. Computational thinking in k-9 education. In *Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference*. ACM, 1–29. (Cited on page 18.)

[92] MARTIN, R. C. 2009. *Clean code*. (Cited on page 13.)

[93] MARTINEZ, J., ZIADI, T., BISSYAND, T. F., KLEIN, J., AND TRAON, Y. L. 2017. Bottom-up technologies for reuse: Automated extractive adoption of software product lines. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 67–70. (Cited on page 50.)

[94] METZGER, A. AND POHL, K. 2014. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on Future of Software Engineering*. ACM, 70–84. (Cited on page 50.)

[95] MEYER, M. 2014. Continuous integration and its tools. *IEEE software 31,* 3, 14–16. (Cited on page 6.)

[96] MICHLMAYR, M., FITZGERALD, B., AND STOL, K.-J. 2015. Why and how should open source projects adopt time-based releases? *IEEE Software 32,* 2, 55–63. (Cited on page 63.)

[97] NAKAKOJI, K., YAMAMOTO, Y., NISHINAKA, Y., KISHIDA, K., AND YE, Y. 2002. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*. ACM, 76–85. (Cited on pages 3 and 4.)

[98] NAYEBI, M., ADAMS, B., AND RUHE, G. 2016. Release practices for mobile apps–what do users and developers think? In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*. Vol. 1. IEEE, 552–562. (Cited on page 63.)

[99] NCWIT. 2015. Ncwit fact sheet — national center for women & information technology. https://www.ncwit.org/ncwit-fact-sheet. (Accessed on 01/4/2018). (Cited on page 18.)

[100] NCWIT. 2017. By the numbers — national center for women & information technology. https://www.ncwit.org/resources/numbers. (Accessed on 01/04/2018). (Cited on page 18.)

[101] (N.D.)., S. L. 2018. Adoption of agile development and continuous integration (ci) in software development worldwide from 2015 to 2018. https://www.statista.com/statistics/673786/worldwide-software-development-survey-agile-development-continuous-integration-adoption/ (Accessed on 08/12/2018). (Cited on pages xix and 5.)

[102] ONG JINGLI. 2017. New tab. chrome://newtab/. (Accessed on 07/4/2018). (Cited on page 18.)

[103] ONLINE, C. 2017. Catrobat. http://developer.catrobat.org/. (Accessed on 07/06/2018). (Cited on pages 8, 13, and 15.)

[104] OPENSOURCE.GUIDE. 2018. Open source metrics — open source guides. https://opensource.guide/metrics/. (Accessed on 07/13/2018). (Cited on page 9.)

[105] PAPERT, S. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc. (Cited on page 18.)

[106] PESSOA, L., FERNANDES, P., CASTRO, T., ALVES, V., RODRIGUES, G. N., AND CARVALHO, H. 2017. Building reliable and maintainable dynamic software product lines: An investigation in the body sensor network domain. *Information and Software Technology 86*, 54–70. (Cited on page 50.)

[107] PIAGET, J. 1948. *The Child's Conception of Space: By Jean Piaget and Barbel Inhelder. Trans. from the French by FJ Langdon & JL Lunzer*. Norton. (Cited on page 18.)

[108] PULKKINEN, V. ET AL. 2016. Synthesizing perceived challenges in continuous delivery-a systematic literature review. (Cited on page 8.)

[109] REASON, J. 2000. Human error: models and management. *BMJ: British Medical Journal 320,* 7237, 768. (Cited on pages 16 and 64.)

[110] RESNICK, M. 2017. *Lifelong Kindergarten: Cultivating Creativity Through Projects, Passion, Peers, and Play*. MIT Press. (Cited on page 19.)

[111] ROSENMÜLLER, M., SIEGMUND, N., PUKALL, M., AND APEL, S. 2011. Tailoring dynamic software product lines. In *ACM SIGPLAN Notices*. Vol. 47. ACM, 3–12. (Cited on page 50.)

[112] ROSSI, C., SHIBLEY, E., SU, S., BECK, K., SAVOR, T., AND STUMM, M. 2016. Continuous deployment of mobile software at facebook (showcase). In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. ACM, New York, NY, USA, 12–23. (Cited on page 8.)

[113] RUSSO, B., DAMIANI, E., HISSAM, S., LUNDELL, B., AND SUCCI, G. 2008. *Open Source Development, Communities and Quality: IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, September 7-10, 2008, Milano, Italy*. Vol. 275. Springer Science & Business Media. (Cited on page 4.)

[114] SANTOS, C., KUK, G., KON, F., AND PEARSON, J. 2013. The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems 22,* 1, 26–45. (Cited on pages 2 and 3.)

[115] SEPÚLVEDA, S., CRAVERO, A., AND CACHERO, C. 2016. Requirements modeling languages for software product lines: A systematic literature review. *Information and Software Technology 69*, 16–36. (Cited on page 50.)

[116] SERRADOR, P. AND PINTO, J. K. 2015. Does agile work?a quantitative analysis of agile project success. *International Journal of Project Management 33,* 5, 1040–1051. (Cited on page 4.)

[117] SHAHIN, M., BABAR, M. A., AND ZHU, L. 2017a. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access 5*, 3909–3943. (Cited on pages 6 and 62.)

[118] SHAHIN, M., BABAR, M. A., AND ZHU, L. 2017b. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access 5*, 3909–3943. (Cited on page 8.)

[119] SHU-YI, W. AND KUO-KUANG, F. 2016. Survey into user's usage feeling towards material design button in the website. In *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*. 632–635. (Cited on page 39.)

[120] SLANY, W. 2014. Pocket code: a scratch-like integrated development environment for your phone. In *Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity*. ACM, 35–36. (Cited on pages 23, 51, and 62.)

[121] SMITH, S. L. 1986. Standards versus guidelines for designing user interface software. *Behaviour & Information Technology 5,* 1, 47–61. (Cited on page 38.)

[122] STORK, C., CALANDRO, E., AND GILLWALD, A. 2013. Internet going mobile: internet access and use in 11 african countries. *info 15,* 5, 34–51. (Cited on page 34.)

[123] SZCZEPANSKA, A. M., BERGQUIST, M., AND LJUNGBERG, J. 2005. 22 high noon at os corral: Duels and shoot-outs in open source discourse. *xvii*. (Cited on page 2.)

[124] TAKAHASHI, D. 2017. Sensor tower: Mobile game revenues grew 32% in q2 as asian titles surged — gamesbeat. https://venturebeat.com/2017/07/24/

mobile-game-revenues-grew-32-in-q2-as-asian-titles-surged/. (Accessed on 08/04/2018). (Cited on page 18.)

[125] TARA BARNETT, BEN LAWLESS, H. K. AND VISTA, A. 2017. Complementary strategies for teaching collaboration and critical thinking skills. https://www.brookings.edu/blog/education-plus-development/2017/12/12/complementary-strategies-for-teaching-collaboration-and-critical-thinking-skills/. (Accessed on 02/08/2018). (Cited on page 21.)

[126] TEAM, G. 2018a. Gradle user manual. https://goo.gl/bTvadr. (Accessed on 04/25/2018). (Cited on page 51.)

[127] TEAM, J. 2018b. Jenkins. https://jenkins.io/. (Accessed on 04/25/2018). (Cited on page 52.)

[128] TEDRE, M. AND DENNING, P. J. 2016. The long quest for computational thinking. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 120–129. (Cited on pages 18 and 21.)

[129] TUMLIN, N. 2017. Teacher configurable coding challenges for block languages. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 783–784. (Cited on page 22.)

[130] TURNER, C. W., LEWIS, J. R., AND NIELSEN, J. 2006. Determining usability test sample size. *International encyclopedia of ergonomics and human factors 3,* 2, 3084–3088. (Cited on page 47.)

[131] VERSLUIS, G. 2017. Why an automated pipeline? In *Xamarin Continuous Integration and Delivery*. Springer, 1–5. (Cited on page 67.)

[132] VOLODYA, B. 2017. Measuring the success of open source project productcoalition.com. https://productcoalition.com/measuring-the-success-of-open-source-project-a76b6fd5fe6b. (Accessed on 07/13/2018). (Cited on page 9.)

[133] WHITE, J., GALINDO, J. A., SAXENA, T., DOUGHERTY, B., BENAVIDES, D., AND SCHMIDT, D. C. 2014. Evolving feature model configurations in software product lines. *Journal of Systems and Software 87*, 119–136. (Cited on page 50.)

[134] WIKIPEDIA CONTRIBUTORS. 2018a. Agile software development — Wikipedia, the free encyclopedia. [Online; accessed 15-July-2018]. (Cited on page 4.)

[135] WIKIPEDIA CONTRIBUTORS. 2018b. Social age — Wikipedia, the free encyclopedia. [Online; accessed 12-July-2018]. (Cited on page 2.)

[136] WING, J. M. 2006. Computational thinking. *Communications of the ACM 49,* 3, 33–35. (Cited on page 18.)

[137] WU, B. AND WANG, A. I. 2012. A guideline for game development-based learning: a literature review. *International Journal of Computer Games Technology 2012*, 8. (Cited on page 21.)

[138] YE, Y. AND KISHIDA, K. 2003. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineering*. ICSE '03. IEEE Computer Society, Washington, DC, USA, 419–429. (Cited on pages 3 and 4.)

[139] ZAIMI, A., AMPATZOGLOU, A., TRIANTAFYLLIDOU, N., CHATZIGEORGIOU, A., MAVRIDIS, A., CHAIKALIS, T., DELIGIANNIS, I., SFETSOS, P., AND STAMELOS, I. 2015. An empirical study on the reuse of third-party libraries in open-source software development. In *Proceedings of the 7th Balkan Conference on Informatics Conference*. ACM, 4. (Cited on page 2.)

[140] ZLOTNICK, F. 2017. Github open source survey 2017. http://opensourcesurvey.org/2017/. (Cited on page 4.)