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

1st Kirshan Kumar Luhana
Institute for Software Technology
Graz University of Technology
Graz, Austria
kirshan.luhana@student.tugraz.at

2nd Christian Schindler
Institute for Software Technology
Graz University of Technology
Graz, Austria
cschindler@ist.tugraz.at

3rd Wolfgang Slany
Institute for Software Technology
Graz University of Technology
Graz, Austria
wslany@ist.tugraz.at

***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.**

***Index Terms*—mobile application, continuous deployment, Catrobat, Pocket Code, open source, Google Play, Fastlane, internationalization (i18n)**

I. Introduction & Background

A. 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 [1] 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.

[1]https://catrobat.org

B. 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.

C. Continuous practices

Continuous practices [2] are emerging software development industry practices based on agile methods mitigating the gap [3] 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 [2], [4].

D. Deployment pipeline (DP)

A deployment pipeline is a way to progress through the release process in stages [5], [6]. Usually, the first stage of the pipeline is to download the latest codebase from the repository to build the binaries for further use [6]. 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?
- 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.

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

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.

E. Release strategy

A release could be a new app or an upgrade with new or modified features [7]. A release also could be a bug fix or a refactored version with better performance. Nayebi et al. 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 [8]. Whether time based or feature based, [9] 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 [10]. 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 [9]. 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) [11]. Therefore multi language apps in combination with a frequent release approach need deployment automation. The consumer IT market is rapidly growing [12] 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 [12]. Releasing software faster than competitors is also an important success factor [13].

F. 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 [14]. 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].

G. 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.

1) 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.

2) 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.

H. Challenges in Catrobat's Pocket Code deployment

Human factor researchers are increasingly concerned with developing tools for handling critical acts [15]. Automation improves performance with Fastlane, reliability, availability, and productivity hence it saves time and money. The main challenges in Catrobat's Pocket Code deployment 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.

[4]https://crowdin.com/pricing

[5]https://support.crowdin.com/github-integration

[6]https://goo.gl/SSJkQj

[7]https://jenkins.io

I. 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 languagecode-CountryCode for others, e.g., "ar" for all Arabic languages, "hr" for Croatian, "ca" for Catalan but "cs-CZ" for Czech, "en-US" for English United States, "en-UK" for English United Kingdom.

II. 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 [16] 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"[10] 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

[8]https://goo.gl/kbhSTV
[9]https://support.google.com/googleplay/android-developer/answer/113469
[10]http://nvie.com/posts/a-successful-git-branching-model

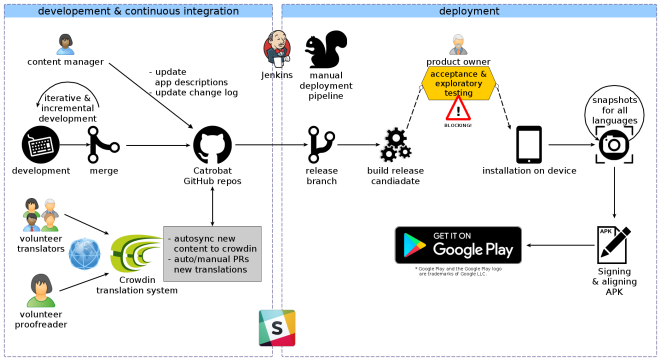

Fig. 1. Catrobat manual deployment workflow

manual interactions in the workflow (see Figure 1). 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 [17] 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 [18], [19].

A. 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.

III. 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 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 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,[11]. 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[12] 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.

A. 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.

B. 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 [20]. All android apps need to be digitally signed with a certificate in order to distribute via Google Play [21]. It is important to always sign all versions with the same certificate.

C. 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[13] 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.

D. 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[14] 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

[11]https://developers.google.com/android-publisher/tracks
[12]https://catrobat.slack.com
[13]https://developer.android.com/training/testing/espresso/index.html
[14]https://support.crowdin.com/cli-tool/

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.

E. 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.

F. 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.

G. Post build actions

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

H. 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.

I. 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.

IV. Deployment time

Manual deployment as it is implemented according to Figure 1 suffers from four main drawbacks, a.) the manual overhead human interaction poses, e.g., task switching, b.) the creation of 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

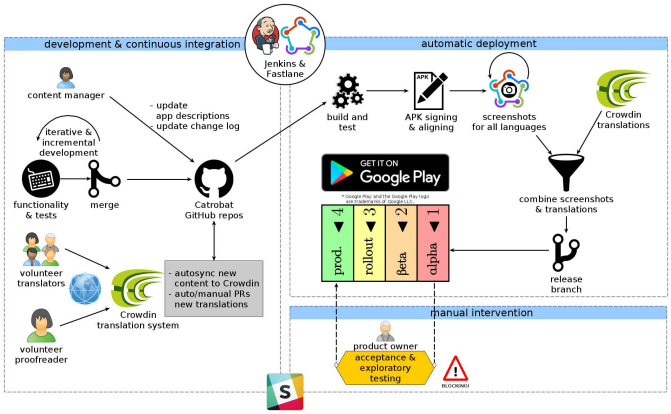

Fig. 2. Catrobat automatic deployment workflow

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 configuration 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.

V. 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.

References

[1] W. Slany, "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, 2014, pp. 35–36.

[2] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[3] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[4] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The highways and country roads to continuous deployment," *IEEE Software*, vol. 32, no. 2, pp. 64–72, 2015.

[5] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.

[6] M. Fowler. (2013) Deployment pipeline. Accessed: 2017-12-15. [Online]. Available: https://martinfowler.com/bliki/DeploymentPipeline.html

[7] M. Nayebi, B. Adams, and G. Ruhe, "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, 2016, pp. 552–562.

[8] M. Michlmayr, B. Fitzgerald, and K.-J. Stol, "Why and how should open source projects adopt time-based releases?" *IEEE Software*, vol. 32, no. 2, pp. 55–63, 2015.

[9] A. Cesar Brandão Gomes da Silva, G. de Figueiredo Carneiro, F. Brito e Abreu, and M. Pessoa Monteiro, "Frequent releases in open source software: A systematic review," *Information*, no. 3, 2017. [Online]. Available: http://www.mdpi.com/2078-2489/8/3/109

[10] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, July 2013.

[11] Google. (2014) 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. Accessed: 2017-12-05. [Online]. Available: https://www.statista.com/statistics/296304/mobile-app-abandoment-rate-due-to-lacking-localization/

[12] J. Hamunen, "Challenges in adopting a devops approach to software development and operations," G2 Pro gradu, diplomityö, 2016. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-201609083476

[13] A. Dyck, R. Penners, and H. Lichter, "Towards definitions for release engineering and devops," in *Proceedings of the Third International Workshop on Release Engineering*. IEEE Press, 2015, pp. 3–3.

[14] A. M. A. Awwad, C. Schindler, K. K. Luhana, Z. Ali, and B. 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*. ACM, 2017, p. 99.

[15] J. Reason, "Human error: models and management," *BMJ: British Medical Journal*, vol. 320, no. 7237, p. 768, 2000.

[16] F. Erich, C. Amrit, and M. Daneva, "Report: Devops literature review," *University of Twente, Tech. Rep*, 2014.

[17] G. Versluis, "Why an automated pipeline?" in *Xamarin Continuous Integration and Delivery*. Springer, 2017, pp. 1–5.

[18] J. Davis and K. Daniels, *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. "O'Reilly Media, Inc.", 2016.

[19] A. Fallis, *Effective DevOps*, 2013, vol. 53, no. 9.

[20] G. Developer. (2016) zipalign. Accessed: 2017-12-16. [Online]. Available: https://developer.android.com/studio/command-line/zipalign.html

[21] ——. (2017) Sign your app. Accessed: 2017-12-17. [Online]. Available: https://developer.android.com/studio/publish/app-signing.html

Final published version: https://doi.org/10.1109/ICIRD.2018.8376296

Citation in BibTex format:

@INPROCEEDINGS{8376296,
author={K. K. Luhana and C. Schindler and W. Slany},
booktitle={2018 IEEE International Conference on Innovative Research and Development (ICIRD)},
title={Streamlining mobile app deployment with Jenkins and Fastlane in the case of Catrobat's pocket code},
year={2018},
volume={},
number={},
pages={1-6},
keywords={computational linguistics;mobile computing;public domain software;Catrobat app;Catrobat's Pocket Code;Fastlane;Google Play;Jenkins;Pocket Paint;active installs;additional repetitive tasks;app store;continuous deployment;continuous integration server;deployment prolongation;deployment steps;left languages;mobile app deployment;mobile app release framework;mobile open source project;multiple languages;reliability;staged deployment approach;supported languages;Androids;Google;Humanoid robots;Manuals;Software;Task analysis;Tools;Catrobat;Fastlane;Google Play;Pocket Code;continuous deployment;internationalization (i18n);mobile application;open source},
doi={10.1109/ICIRD.2018.8376296},
ISSN={},
month={May},}