International Journal of Engineering & Technology, 7 (4.15) (2018) 333-339

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Building Pocket Code build-variants

Kirshan Kumar Luhana

Graz University of Technology Graz, Austria
**Corresponding author E-mail:kirshan.luhana@student.tugraz.at*

Abstrat

Pocket Code is an integrated development environment (IDE) targeted at smartphones. With this IDE users can create mobile apps for the block-based visual programming language Catrobat. Pocket Code is released in various flavors with custom features for partners and projects (e.g., Pocket Code, Create@School, Phiro, and Standalone). All flavors extend a single common project codebase according to flavor specific requirements. The Standalone variants (debug and release) convert a Catrobat project into an Android application to install it independently and execute it without the need for an installed Pocket Code on an Android smartphone. Furthermore, it can be published on app stores for reputational and also monetary benefits. The app resource files and the configuration are generated on the fly upon a user request via the Pocket Code sharing platform. In this paper, the approach of building a Pocket Code variant and transform a Pocket Code project into an Android application are described. Especially the Standalone build variants have the potential to bring many interesting apps to the market.

Keywords: *Catrobat; Pocket Code; Android Build variant, Dynamic Software Product Lines; Variability*

1. Background & Introduction

By 2018, eMarketer forecasts that over one-third of consumers worldwide that is more than 2.56 billion people use smartphones [1]. With increasing demand and popularity for mobile computing and mobile devices, the number of application development projects increased significantly in the last years. [2] [3]. In mobile application development, aspects such as short development lifecycle, mobility, visibility, screen size, user interface (UI) design and navigation of the application [2], process architecture, storage size, RAM capacity, and privacy policy are supposed to be handled carefully [2]. Moreover, users expect variability from an application in, e.g., features, location, resource awareness, and accessibility. The system needs to adopt various deployment and usage strategies (e.g., product lines and families, self-adaptive systems, configurable or customizable single systems, open platforms, context-aware mobile apps, plug-ins of web browsers, service-based, systems in clouds, and the Internet of Things) [4]. Therefore, development requires extra efforts to release different versions of the same application for meeting their hardware, accessibility, functionality, and compatibility requirements. Releasing different flavors of the same application with a different set of functionalities is a common problem faced by the software industries [5] [6]. In traditional software development practices, each flavor was treated as an individual product. Elicitation of requirements, design, implementation, testing, and maintenance for each product leads to high development and maintenance costs [7] [8]. Furthermore, if any potential feature or change across all variants is required, all projects will be updated manually, which is neither scalable nor efficient [8] [9] [5]. For this reason, different strategies are applied in software development to reuse components [7]. One approach to reuse core functionalities across all flavors is to build a library module for core functionalities and use it in all projects as a library component. The approach reduces refactoring efforts [10], but it actually turns an app development project into a library project. The library project architecture and development process are different than app development. Moreover, updating all flavor-specific projects with the latest version of the library requires extra efforts and creates "dependency hell" [11]. There is another approach that builds a system that contains all features and configurations inside a single package [10]. The system behavior is handled by a configuration file. The system behavior can be changed by configuration at build time, deploy time, and runtime [11] [12]. The challenge in developing the configurable system is a suitable mechanism for the dynamism [13]. An existing variability mechanism (e.g., if-then-statements) and/or underlying approaches increase complexity with an increase in the number of configurations and not suitable for a family of mass products [13]. The other disadvantage of this approach is that it includes all resources, assets, aesthetic details, and features for all variants, and increases package size with irrelevant resources, and features. This approach is rarely an economical solution [10].

Software Product Line Engineering (SPLE) is an emerging technology [7]. SPLE has attracted attention due to its ability to reuse requirements and components [14]. A software product line (SPL) is used to build software, families of products, i.e., similar products differentiated by certain characteristics from a common codebase with a set of features to satisfy the targeted market or project requirements [15] [7]. The SPL paradigm empowers companies to produce quality software and to mitigate costs and launch times [7]. Furthermore, it increases productivity, user-satisfaction, product quality, and development of moral [16] [17]. Thereby, SPLE is considered one of the major accelerators in the software development process [16] and promoting the industrialization of software development [10] [7]. SPLE has convinced the industry to develop a variety of similar software at low cost in short time with higher quality in contrast to single system development [5]. Mobile Software Product-line Engineering (MSPLE) is emerging as a promising research area [18]. Software product line (SPL) generates a set of tailor-made products from a common codebase (e.g., for different customers or application contexts) [19] reusing feature and exploiting variability and configurable options [6]. In configurable systems and product lines, variability is an important characteristic [20]. "Software variability is the ability of a software system or artifact to be changed, customized or configured for use in a particular context." [21]. Nowadays, systems

become variability-intensive to satisfy functional and non-functional characteristics of the application and for various business targets, e.g., budget efficiency, quality enhancement, and reduce time to market [20]. Variability can be both intentional or unintentional [4] and is used in various situations, e.g., custom user needs, dynamics in the availability of resources or external services, a variety of hardware. Thus, in software engineering, variability is addressed in broader context [4]. Research on variability specifically on the usage of variability mechanisms and techniques in Android is still unclear [20]. Modern applications need to be self-adaptive to change according to system environment and user demands. Systems that demand self-adaptive capabilities, including mobile applications that deal with different environments, enable or disable services on-the-fly [22]. The dynamic software product line engineering (DSPLE) approach for developing self-adaptive systems is based on commonalities and variabilities for a family of similar products [22]. The Dynamic Software Product Line (DSPL) is an extended version of the conventional Software Product Line [23].It generates product variants at runtime. The difference between SPL and DSPLis, in SPL variability analysis, decision making, and configuration that takes place at the design time; in contrast, DSPL emphasizes variability analysis at design time, postponing the decision of the variability and the application reconfiguration to be made at runtime [23] [24]. DSPL dynamicity allows the SPL to be reconfigured at runtime [23] [24].

The Pocket Code is a visual programming language environment that allows the creation of games, stories, animations, and many types of other apps directly on smartphones. The user can share these projects on the Catrobat sharing platform. Pocket Code is available for Android and iOS platforms (beta) [25]. The Pocket Code Android version is released in various flavors for different partners and projects (e.g., Create@School, Phiro, and Standalone). All flavors of Pocket Code share the common codebase with additional flavor specific design and features. To release all Pocket Code variants from a common codebase, Pocket Code DSPL uses Gradle as its build-tool and Jenkins server as a build and delivery server [10].

In this paper, Pocket Code build variants are discussed, particularly the Standalone build variant, which converts Catrobat user projects into Application Package Kits(APK). An APK can be installed as an independent application on Android phones and can be hosted on app stores. When users trigger a build Standalone job on the Catrobat sharing platform, the app configuration, assets, app title, app name, package ID, etc., are passed to Catrobat build server with the selected Catrobat project information. The build server reconfigures app metadata including selected Catrobat project details, icons, app permissions, etc., at build-time. On a successful build, it sends back the APK to the sharing platform for downloading the project as the standalone app.

1.1. Catrobat project

The Catrobat[1] is an independent free and open source software (FOSS) project [26]. From more than 20 countries, more than 500 volunteers contribute to the design, development, and translation of the Catrobat software [25]. Catrobat and the software developed by Lab [27]. With Pocket Code, a user can create and execute programs on smartphones. Catrobat projects are platform independent. A project developed on one platform can be directly executed on other platforms. These projects can be shared via the Catrobat sharing platform. Shared Catrobat projects are Affero General Public License (AGPL)[2] licensed. The license allows learning, remixing and sharing them freely with the community [26].

1.2. Pocket Code project

Pocket Code is a mobile-based integrated development environment (IDE) for visual block programming language. Android version is publically available via different app stores, and its iOS version is in beta testing phase. Pocket Code helps its users to learn how to code and create apps in a very short time with little or no programming experience [27]. The Pocket Code internationalization and localization features help users better understand and comfortably use the app in their chosen language [28] [10]. The Catrobat platform can be accessed directly from Pocket Code IDE for sharing projects easily on sharing platform or downloading other users program for learning and remixing purpose [10]. Pocket Code is also released in different variants for their partners and projects such as Create@School[3] and Phiro[4] [10].

1.3. Catrobat share community

Fig.1: Pocket Code project sharing platform

Catrobat maintains a web-platform where users can share their projects [10]. A user can share his project or download any project available on sharing platform for learning and remixing purpose. Sharing community offers many features such as HTML5 Catrobat project player, that helps running a project in any HTML5 compatible browser. Intelligent recommended system and remix graph check relevant project and explore more possibilities. Platform displays projects in different groups, e.g., latest, most download, most view etc to observe market trends. The Catrobat platform also offers users to transfer any of Catrobat project 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 [29]. User projects as APKs can be installed on Android devices and executed without the need of the Pocket Code IDE.

the Catrobat team are inspired by the Scratch programming system

[1]https://catrobat.org

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

[3]https://edu.catrob.at/

[4]https://www.phiro.science/

1.4. Gradle build system

Gradle [5] is an open-source build automation tool focused on flexibility and performance [30]. Android uses Gradle as their official build tool [30] [31]. When Gradle plugin for Android and Android studio were introduced, the focus of development was on code reusability, build variants, customization and configuration of build process [32]. All build tasks available to the Android project can be executed via Gradle wrapper[6] command line tool [33]. Each Android project provides the Gradle wrapper as part of a repository and enables the project to build on continuous integration system e.g., Jenkins without having Gradle runtimes [34]. The Gradle and Android plugin enable configuration of build type (debug, release types) or product flavors (paid, free version), and build variant (paid release version, free release version) of Android project [35].

1.5. Jenkins server

Jenkins is an open source, self-contained automation server used to automate all sorts of tasks related to building, testing, and delivering or deploying software [36]. Jenkins jobs can be triggered manually, by REST API calls or on a predefined schedule [37]. A Jenkins job can also receive parameters which can be used to customize the job [38] [10]

1.6.Catroid Repository

Pocket Code repository is hosted on GitHub under Catroid name [7] [10]. All Pocket Code flavors are developed and maintained in the single repository. Keeping all build variants in single repository helps easily maintaining of core functionalities across all build variants.

2. Pocket Code variants

As already mentioned in this paper, there are different Pocket Code flavors for, e.g. partners or projects and for Standalone build flavors [10]. Each flavor has its own acceptance criteria; additionally, there are two build-types (debug and release) of each flavor. When all build-type properties are added to the specific product flavor properties, it is called a build variant [10]. All variants slightly differ in design and functionality.

2.1. Pocket Code variants

The Pocket Code is the main flavor of the family with features targeting the general audience. The Pocket Code flavor is available in two variants i) debug for testing, and ii) release version for the public. Both variants have minor difference in acceptance criteria. In the release version, Crashlytic is used to log crash reports; also when a user uses Pocket Code first time, with Snackbar feature app shares a lightweight guide to introduce Pocket Code IDE. Crashlytic must be disabled in debug version, that it will not increase crash log while testing app. Snackbar feature is also not required and may disturb automatic testing or waste tester time [10].

2.2.Create@School

The Create@School variant is built with a focus on schools [10]. With all common functionalities of Pocket code, it has additional school-related features. The app must contain its own launch icon, unique app package name, app title to parallelly installable with other Pocket Code variants on the same device or publish on Google play store [10].

2.3. Phiro

The Phiro flavor is solely designed to control Phiro[8] robots wirelessly. This version of Pocket Code does not require any extra feature and must be excluded to maintain app size and usability. The Phiro flavor is also published on Google play under its own unique app package name, icon, app title etc.

2.4. Build-Standalone

When a user creates a game, animation or any application in Pocket Code IDE, Pocket Code creates a project file with extension " catrobat". A user can share their projects with friends or on sharing platform. These projects are platform independent that means if a project is developed on Android, it can run in Pocket Code for iOS or on Pocket Code player in any HTML5 compatible browser. The Build-Standalone is an atypical variant of Pocket Code, which transfers Catrobat user project into Android application package (APK) file. This APK file can be installed as an independent application and does not requires Pocket Code IDE. Each project converted as an Android application (known colloquially as "app") contains its own unique package name, app_title, icon. The build-Standalone variant is available in two variants [10].

2.4.1. Unpublishable build-standalone variant

From Catrobat share platform, any user can transfer any shared project into a standalone app (APK) by pressing "Prepare app" option as shown in figure 1. Unpublishable build-standalone must discourage intellectual property theft so users do not publish someone else work for reputation or monetary gains. by fixed version name, version code, package name, etc and placing a banner showing project affiliation with Pocket Code.

2.4.2. Publishable build-standalone variant

The Publishable build-standalone variant is still in beta-testing phase and is not available to public [10]. This build variant will enable the owner of the project to release the Catrobat project as an app on app stores like Google Play with the flexibility to customize properties, and optionally include mobile advertisements to earn money too as shown in figure 2.

3. Building Pocket Code variants

As mentioned in section 1.6, all flavors of Pocket Code use the same codebase and slightly different in design and functionality [10]. To release different variants of Pocket Code, Pocket Code product line uses Gradle as its build-tool and Jenkins-CI server as a build and delivery server [10]. The Gradle combines settings, code, and resource configuration using a specific set of rules to structure a build variant. Technically, a build variant is not configured directly, it is actually a cross product of build type and product flavor [39] [40]. By default, Gradle creates two build types (Debug and Release) when an app is created in Android studio [35]. The following code snippet 1 shows Pocket Code build-types.

```
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 1: Pocket Code build type configuration in build.gradle [10]

[5]https://gradle.org/
[6]https://docs.gradle.org/current/userguide/gradle wrapper.html
[7]https://github.com/Catrobat/Catroid

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

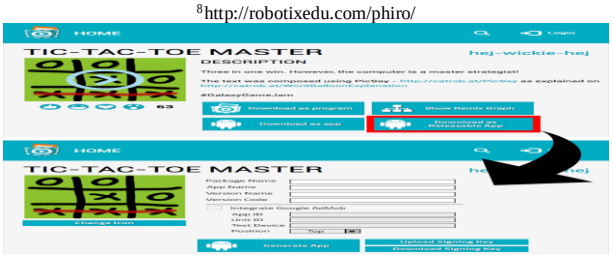

Fig. 2: Pocket Code project publishable build variant user interface

In build type section certain properties, typically related to different development lifecycle are configured. For all Pocket Code variants, Snackbar and Crashlytics are required only in release variant. Code snippet 1 shows in debug build type the Snackbar and Crashlytics are disabled; whereas, in the release version, they are enabled. Gradle creates the 'BuildConfig' Java class at build time with constants which are defined with Gradle method 'buildConfigField'. Similarly, Gradle creates resource variables defined by 'resValue' Gradle method. The constant in Java class can be accessed by importing the 'BuildConfig' Java class and a resource of the type specified e.g., String into 'res' (Resource) directory and could refer it via XML with '@string/SNACKBAR_HINTS_ENABLED' [10]. This variability mechanism allows deciding at build time and which features should be enabled or disabled. The mechanism not only helps in controlling app behavior but used to pass certain values which are not available in codebase or not feasible to store in a repository such as service credentials etc. In section 4 the 'buildConfigField' method is discussed in more detail. To build different features from a single codebase, Gradle 'productFlavor' feature allows easy configuration for each flavor [10]. The following code snippet 2 shows configurations for different flavors of Pocket Code [10]. This makes customization of flavor specific properties easy and manageable e.g., appID. In source code flavor specific code and resources (icon, logo, etc) are placed under its respective folder e.g "./src/[flavor-name]". On building specific variant, Gradle includes only files from the folder specified for that variant. All common resources and Java code are stored in the "main" directory. Flavor specific code and resources are located under flavor named folder and are used only by that flavor. Source code and resource files are handled differently [10]. Resource files, such as strings, icon, layout files override if they are placed in the same location and with the 'same name' as in the 'main' resources during the build. This makes it easy to replace logo or layout theme. For example, for a different app logo, the new image just needs to be placed under "./../src/flavor-name/res/drawable-****/ic launcher.png". In case, Java classes, if the flavor specific fold contains same Java class name as in the 'main' folder, Gradle will throw "duplicate class" error [40]. This can be handled by making an abstract base class and

then extend the class in the flavor-specific directory. This helps to include only resources which are required for a particular flavor and changing app's behavior by extending flavor specific code.

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

Code-Snippet 2: Pocket Code build Flavor build.gradle file [10]

In all Pocket Code variants except Standalone build variant, app properties, resources, configuration, etc are predefined in the codebase. A release engineer triggers build manually and monitor its progress. Wherease, Standalone build variant triggers from Catrobat sharing platform by a visitor. The resources, app properties, configurations are dynamically changed, based on visitor selection.

4. Standalone-Build variant

A Catrobat user project file with extension "catrobat" can be run within Pocket Code IDE or in the browser-based Pocket Code HTML5 player [10]. The Standalone-Build variant transfers Catrobat user project into Android application package (APK) file installable on an Android phone as an app. These apps do not require Pocket Code IDE and execute as independent apps. To transfer Catrobat project into standalone version of a project hosted on sharing platform, the user triggers the build via the web interface on the remote Catrobat build server. When a user press "Prepare App" button on Pocket Code community platform as in figure 1; internally, a job on Catrobat' CI server is triggered with parameters containing user's selected project information. Before build starts, app properties, app permission, resources, and launch icon are all unknown in the codebase. Information such as package name, version code, or AdMob user key is assigned via variables.

Catrobat uses Jenkins-CI server for integration, build and deploy of its Android Projects. When user triggers build standalone job, Jenkins runs a Gradle task with required parameters. The Gradle task downloads selected project which is to be built and placed at an appropriate location. A user Catrobat project file contains information about app permissions which are required to run on an Android device and icon of the app etc. The same Gradle task extracts permission information and icon for use in the build. The repository contains an Android Manifest file template for Standalone build variant. The Gradle task updates template with app permission, package name, app package name, version code, version name, etc. When Gradle finishes generating all resources and placing them in appropriate folders, Jenkins executes another Gradle task to assemble an APK. If both tasks execute correctly without any error, Jenkins deploys the APK back on the Sharing platform for use as an Android app. To discourage intellectual property theft, these app display the "Made with Pocket Code" banner when app closed normally. Android requires that all APKs be digitally signed with a certificate before they can be installed. When a debugging type of application is generated, Android automatically signs that APK file with a debug certificate generated by Android SDK [42]. App stores such as Google Play Store and other do not accept apps signed with debug certificate. The sharing platform generates debug build

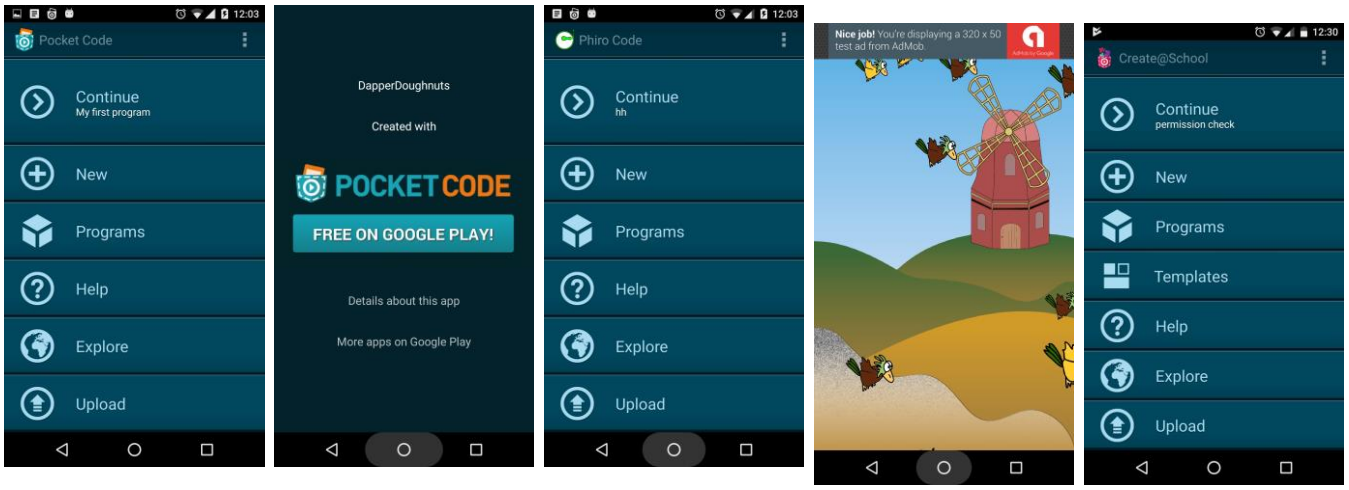

Fig. 3: Pocket Code flavors. Pocket Code, Standalone, Phiro, Publishable with Admob, and Create@School [10].

type of Standalone variant that cannot be published on Play store, also, "Prepare app" option creates APK with default app version and version name i.e 1.0 and creates unique package name in the format as org.catrobat.generatedXXX, where XXX is project ID on share platform.

```
                                        appIcon = '@drawable/icon'
./gradlew -Pdownload="${DOWNLOAD}" -Papk_generator_enabled=true
        -Psuffix="${SUFFIX}" buildStandalone

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

Code-Snippet 3: Gradle command with parameters from build server [10]

The Gradle task in code snippet 3 shows an unpublishable variant call with parameters. The 'assembleStandaloneDebug' task tells Gradle to assemble the Standalone flavor using the debug build-type [10]. The publishable variant mechanism is similar to unpublishable build variant. Publishable variant needs more information to sign APK with the private certificate and optional information for AdMob integration. The publishable feature will be available to registered users. A user can only publish his/her apps by logging in sharing platform. Before triggering publishable Standalone-build variant job, some information is required from the user as in figure 2. In case of releasable variant, the user can assign package name of his/her choice, change app version code and version name for app next version under same package name. When a build is triggered, Jenkins runs Gradle task as shown in snippet 4 to prepare releasable build type.

```
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: Gradle call with parameters for AdMob integration [10]

Changing source code is risky as compared to variable values [11]. Credentials for services or private information such as AdMob account details are not advised to store in the source code, especially in open source software [10]. In Pocket Code Standalone variant, AdMob account details are varying from user to user and must be dynamically included by the user. For different variants, if a value is different for example app ID, start project, etc, assign value via variable [10]. Following code snippet in 5 is taken from the build.gradle file which illustrates how in the Standalone build variables are used to pass AdMob user account details or set configuration for publishable APK. In all variants, the snackbar is enabled in release build-type except Standalone, as it is not designed for user projects introduction.

```
standalone {
applicationId getPackageNameSuffix()
versionCode getVersionCodeForStandAlone().toInteger()
versionName getVersionNameForStandAlone()
appName = $appName


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 5: Pocket Code Standalone build Gradle configuration [10]

Gradle method 'buildConfigField' creates the Java class named with 'BuildConfig' name. Inside the 'BuildConfig' class, Gradle declares constant at build time. These constants are used in the source code as shown in code snippet 6.

```
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 6: Admob inegration and conditional behavior handling [10]

In Pocket Code product line, Gradle not only makes it easy to manage source code, resources for different variants but also with variability enables build to customize software behavior and include or exclude resources dynamically [10].

5. Conclusion

Modern computer languages, build tools and build servers lead to all sorts of characteristics and techniques to maintain the project in efficient and scalable manner. If system configuration is properly applied, variability in a software product line can drastically reduce time spent on build configuration and risk. In this paper, we discussed the Pocket Code product line and its variability mechanism to release

different versions from a single common codebase. The Catrobat sharing platform helps users to share their creative work within the community. The Standalone build enables users to share these projects as standalone apps. With the end of April 2018, we counted over 52,000 projects available on the Catrobat sharing platform and as of June 2018 over 56,000. Users triggered the standalone APK creation job for over 42,000 times until April 2018 and as of June 2018, it was over 47,000 times. The interesting observed fact is that the generated APKs were downloaded over 205,000 times until April and as of June 2018, the number increased to over 238,000 APK downloads. The results not only validate our approach but also show the great demand of this Standalone build variant feature. The upcoming extended version of this feature, which is currently in beta testing phase, will allow users to prepare their project APKs to publish them on Google Play. The presented approach was already successfully applied to publish three Pocket Code generated applications on Google Play, 'Space Portal"[9], "Phiro Play"[10] "Tic-Tac-Toe Master"[11]. The "Tic-Tac-Toe Master" app has 50,000+ downloads on Google Play which hints that projects generated with Pocket Code can create an impact.

Referencs

[1] eMarketer, "2 billion consumers worldwide to get smart(phones) by 2016 - emarketer," https://www.emarketer.com/ Article/2-Billion-Consumers- Worldwide-Smartphones- by-2016/
[2] 1011694, 2016, (Accessed on 06/15/2018). [On- line]. Available: https://www.emarketer.com/Article/
[3] 2-Billion- Consumers-Worldwide- Smartphones-by- 2016/1011694 Aldayel and K. Alnafjan, "Challenges and best practices for mobile application development: Review paper," in Proceedings of the International Conference on Compute and Data Analysis, ser. ICCDA '17. New York, NY, USA: ACM, 2017, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/3093241.3093245
[4] H. K. Flora, S. V. Chande, and X. Wang, "Adopting an agile approach for the development of mobile applications," International Journal of Computer Applications, vol. 94, no. 17, 2014.
[5] M. Galster, U. Zdun, D. Weyns, R. Rabiser, B. Zhang, M. Goedicke, and G. Perrouin, "Variability and complexity in software design: To- wards a research agenda," ACM SIGSOFT Software Engineering Notes, vol. 41, no. 6, pp. 27–30, 2017.
A. Metzger and K. Pohl, "Software product line engineering and vari- ability management: achievements and challenges," in Proceedings of the on Future of Software Engineering. ACM, 2014, pp. 70–84.
[6] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Corte´ s, and M. Hinchey, "An overview of dynamic software product line architectures and tech- niques: Observations from research and industry," Journal of Systems and Software, vol. 91, pp. 3–23, 2014.
[7] N. R. Brisaboa, A. Cortin˜ as, M. R. Luaces, and M. Pol'la, "A reusable software architecture for geographic information systems based on software product line engineering," in Model and Data Engineering, L. Bellatreche and Y. Manolopoulos, Eds. Cham: Springer Intern tional Publishing, 2015, pp. 320–331.
[8] M. Usman, M. Z. Iqbal, and M. U. Khan, "A product-line model- driven engineering approach for generating feature-based mobile applications," Journal of Systems and Software, vol. 123, no. October 2013, pp. 1–32, 2017.
[9] J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, and
[10] D. C. Schmidt, "Evolving feature model configurations in software
[11] product lines," Journal of Systems and Software, vol. 87, pp. 119–136,
[12] 2014.
[13] K. K. Luhana, "Pocket code build variants," in 2018 IEEE International
[14] Conference on Innovative Research and Development (ICIRD), May
[15] 2018, pp. 1–6.
[16] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader). Pearson Education, 2010.
[17] S. Sepu´ lveda, A. Cravero, and C. Cachero, "Requirements modeling languages for software product lines: A systematic literature review," Information and Software Technology, vol. 69, pp. 16–36, 2016.
[18] T. Dinkelaker, R. Mitschke, K. Fetzer, and M. Mezini, "A dynamic software product line approach using aspect models at runtime," in Pro- ceedings of the 1st Workshop on Composition and Variability. CEUR Workshop, 2010, pp. 180–220.
[19] M. Li, A. Grigg, C. Dickerson, L. Guan, and S. Ji, "A product line systems engineering process for variability identification and reduction," arXiv preprint arXiv:1806.04705, 2018.
[20] K. Pohl, G. Bo¨ ckle, and F. J. van Der Linden, Software product line engineering: foundations, principles and techniques. Springer Science
[21] & Business Media, 2005.
[22] T. Du¨ rschmid, M. Trapp, and J. Do¨ llner, "Towards architectural styles
[23] for android app software product lines," in Mobile Software Engineer-
[24] ing and Systems (MOBILESoft), 2017 IEEE/ACM 4th International
[25] Conference on. IEEE, 2017, pp. 58–62.
[26] J. Martinez, T. Ziadi, T. F. Bissyande´ , J. Klein, and Y. L. Traon,
[27] "Bottom-up technologies for reuse: Automated extractive adoption
[28] of software product lines," in 2017 IEEE/ACM 39th International Con-
[29] ference on Software Engineering Companion (ICSE-C), May 2017, pp.
[30] 67–70.
[31] L. Pessoa, P. Fernandes, T. Castro, V. Alves, G. N. Rodrigues, and
[32] H. Carvalho, "Building reliable and maintainable dynamic software
[33] product lines: An investigation in the body sensor network domain,"
[34] Information and Software Technology, vol. 86, pp. 54–70, 2017.
[35] M. Rosenmu¨ ller, N. Siegmund, M. Pukall, and S. Apel, "Tailoring
[36] dynamic software product lines," in ACM SIGPLAN Notices, vol. 47,
[37] no. 3. ACM, 2011, pp. 3–12.
[38] N. Fußberger, B. Zhang, and M. Becker, "A deep dive into android's
[39] variability realizations," in Proceedings of the 21st International Sys-
[40] tems and Software Product Line Conference-Volume A. ACM, 2017,
[41] pp. 69–78.
[42] D. Beuche, H. Papajewski, and W. Schro¨ der-Preikschat, "Variability
[43] management with feature models," Science of Computer Programming,
[44] vol. 53, no. 3, pp. 333–352, 2004.
[45] J. D. A. S. Eleute?rio, B. B. N. de Franc?a, C. M. F. Rubira, and R. de Lemos, "Realising variability in dynamic software product line solutions," in Software Engineering for Variability Intensive Systems:
[46] Foundations and Applications. CRC Press, 2018. [Online]. Available:
[47] http://kar.kent.ac.uk/66574/
[48] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic soft-
[49] ware product lines," Computer, vol. 41, no. 4, pp. 93–95, April 2008.
[50] J. Eleute´rio and C. Rubira, "A comparative study of dynamic software
[51] product line solutions for building self-adaptive systems," 2017.
[52] K. K. Luhana, C. Schindler, and W. Slany, "Streamlining mobile app
[53] deployment with jenkins and fastlane in the case of catrobat's pocket
[54] code," in 2018 IEEE International Conference on Innovative Research
[55] and Development (ICIRD), May 2018, pp. 1–6.
[56] "Catrobat," http://developer.catrobat.org/, (Accessed on 06/21/2018).
[57] [27] W. Slany, "Pocket code: a scratch-like integrated development environ-
[58] ment for your phone," in Proceedings of the companion publication of
[59] the 2014 ACM SIGPLAN conference on Systems, Programming, and
[60] Applications: Software for Humanity. ACM, 2014, pp. 35–36.
A. M. A. Awwad, C. Schindler, K. K. Luhana, Z. Ali, and B. Spieler,
[61] "Improving pocket paint usability via material design compliance and
[62] internationalization & localization support on application level," in
[63] Proceedings of the 19th International Conference on Human-Computer
[64] Interaction with Mobile Devices and Services. ACM, 2017, p. 99.
[65] Google, "Apk expansion files — android developers," https:
[66] //developer.android.com/google/play/expansion-files.html, (Accessed
[67] on 04/25/2018). [Online]. Available: https://developer.android.com/
[68] google/play/expansion-files.html
[69] Gradle, "Gradle user manual," https://docs.gradle.org/current/
[70] userguide/userguide.html, (Accessed on 06/21/2018).
[71] ——, "Building android apps," https://guides.gradle.org/
[72] building- android-apps/? ga=2.192105194.1765589774.
[73] 1529583778-1316420249.1529583778, (Accessed on 06/21/2018). [32] G. Developer, "Android plugin for gradle release notes — android
[74] studio,"
[75] opers," https://developer.android.com/studio/build/building-cmdline,
[76] (Accessed on 06/21/2018).

[77] android, “Build your app from the command line — android developers,”https://developer.android.com/studio/build/building-cmdline, (Accessed on 06/21/2018)org/
[78] “Executing gradle builds on jenkins,”https://guides.gradle.org/
[79] executing-gradle-builds-on-jenkins/, (Accessed on 06/21/2018).
[80] “Configure your build — android developers,” https://developer.
[81] android.com/studio/build/, (Accessed on 06/21/2018).
[82] “Jenkins user documentation,” https://jenkins.io/doc/, (Accessed on
[83] 06/21/2018).
[84] J. team, “Jenkins,” https://jenkins.io/, (Accessed on 04/25/2018).
[85] [Online]. Available: https://jenkins.io
[86] K. Kawaguchi, “Parameterized build - jenkins - jenkins wiki,” https:
[87] //wiki.jenkins.io/display/JENKINS/Parameterized+Build, 2017, (Ac-
[88] cessed on 04/25/2018).
[89] G. Developer, “Configure build variants — android studio,” https:
[90] //developer.android.com/studio/build/build- variants.html, 2018, (Ac-
[91] cessed on 04/25/2018).
[92] ——, “Configure your build — android studio,” https://developer.
[93] android.com/studio/build/index.html, 2018, (Accessed on 04/25/2018). [41] ——, “Multiple apk support — android developers,” https://developer.
[94] android.com/google/play/publishing/multiple-apks.html, 2018, (Accessed on 04/25/2018). [Online]. Available: https://developer.android.com/google/play/publishing/multiple-apks.html
[95] Google, “Sign your app - android studio,” 2018, accessed: 4th April, 2018. [Online]. Available: https://developer.android.com/studio/ publish/app-signing.html