UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Onuche Akor Idoko

Automating the Release Planning of Mobile Apps by Including App-Reviews

Master's Thesis (30 ECTS)

Supervisor: Ezequiel Scott, PhD

Tartu, 2020

Automating the Release Planning of Mobile Apps by including App-Reviews

Abstract

Stakeholders constantly think of the best and sustainable approach in delivering new releases to their customers. Large software companies like Google and Facebook invest huge money in their release planning process. That is because release planning impacts the end-user. One goal in software engineering is to make most or all stakeholders happy. However, start-ups, open-source projects and other small software organizations focused mainly on mobile app development may not have enough resources to invest in their mobile release management; as a result, it is important to plan the releases of mobile apps. The development team makes decisions like who is the release intended for, what functionalities or features should the release have, when should the release happen and how much quality should the release have. In order not to lose customers to competitors, teams must make these decisions carefully. Therefore, it is our strong conviction that with user app-reviews from mobile app stores (e.g., Play Store and App Store), we can automate and optimize the release planning of mobile apps. In this paper, we introduce an approach that automatically plans and optimizes mobile releases for software development teams by combining app-review and issue tracker information.

Keywords:

Release Planning, Mobile App-Review, Issue Tracker, Machine Learning, Natural Language Processing, Software Engineering, Genetic Algorithm and Linear Programming.

CERCS:

P170 Computer science, numerical analysis, systems, control

P176 Artificial intelligence

Mobiilirakenduste väljalasete planeerimise automatiseerimine rakenduste arvustusi kasutades

Lühikokkuvõte

Sidusrühmad otsivad pidevalt parimat ja jätkusuutlikku lähenemist, kuidas toodete uusi täiustusi kliendini tuua. Suured tarkvarafirmad, nagu Google ja Facebook, investeerivad tohutuid summasid enda väljalasete planeerimise protsessi, kuna see mõjutab lõppkasutajat. Üks eesmärke tarkvaraarenduses on teha kõik või enamus sidusrühmad õnnelikuks. *Start-up'idel*, avatud lähtekoodiga projektidel ja muudel väikestel tarkvaraorganisatsioonidel, mis keskenduvad peamiselt mobiilirakenduste arendusele, ei pruugi olla küllalt ressursse, et investeerida enda mobiilirakenduste täiustuste juhtimisse. Tagajärjena on oluline seda planeerida. Arendusmeeskond otsustab, kellele on täiustus mõeldud, millised funktsioonid sellel olema peaksid, kuna selle peaks avalikustama ja kui kvaliteetne peaks täiustus olema. Meeskonnad peavad need otsused tegema ettevaatlikult, et mitte kaotada kliente konkurentidele. Seetõttu oleme veendunud, et rakenduste kasutajate arvustustega mobiilrakenduste poodidest (näiteks Play Store ja App Store), on võimalik automatiseerida ja optimeerida mobiilirakenduste väljalasete planeerimine. Selles teaduslikus artiklis tutvustame lähenemist, mis automaatselt planeerib ja optimeerib mobiilirakenduste väljalasked arendusmeeskondade jaoks, kombineerides rakenduste arvustuste ja probleemiraportite informatsiooni.

Võtmesõnad:

Vabastamise Plaanimine, Mobiilirakenduse Ülevaade, Probleemi Jälgija, Masinõpe, Loomuliku Keele Töötlemine, Tarkvaraarendus, Geneetiline Algoritm, ja Lineaarne Programmeerimine.

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

P176 Tehisintellekt

Acknowledgements

Foremost, I thank God Almighty for His grace, mercy and blessings upon my life because, without Him, this research will not be possible.

I am extremely grateful to the University of Tartu and everyone whose help was valuable during my MSc program and in this research.

In particular, I appreciate my Supervisor, Dr Ezequiel Scott for his sincere and valuable guidance throughout this research.

Also, I would like to extend my thanks to Wolfgang Slany, for his quick response to our interview invitation and for agreeing to take part in this research.

These acknowledgements are not complete, without appreciating my family for their patience, understanding and support. Without their prayers and sacrifices, I would not have gone this far. May God keep them and continue to bless them.

Table of Contents Page

1. Introduction

In software engineering, the use of good practices can have an impact on the success of a software product (Ebert, 2014). Regardless of the type of process used to develop software (agile, hybrid, lean or waterfall), the need to plan the releases of a product became eminent. There are many tools and strategies applied by teams in release planning. For example, using Bitbucket and Jira, developers can quickly ship fixes to users instantly. The main idea behind continuous integration and delivery is trying to simplify and improve the value derived from release planning.

Mobile applications which are a type of software products have unique characteristics. For example, mobile apps are usually made available by developers on app stores, where users can download and review the mobile app. This is usually available to end-users anywhere and anytime. Usually, these mobile app stores give users useful tips on how to post clear app-reviews. In general, software companies developing mobile apps make their users aware of new updates by implementing weekly or even daily releases.

The main challenge in release planning is that teams and stakeholders have to make difficult decisions about the release, as these decisions affect the end-users. They have to decide the functionalities, time, target users and quality of the release. Release planning addresses most of these challenges (Ruhe & Saliu, 2005). The goal of this research is to offer an approach that simplifies the release planning of mobile apps by optimizing not only the internal stakeholders' satisfaction but also the end user's opinions based on information extracted from app-reviews.

To achieve our research goal, we use a mixed-method approach that integrates both quantitative (e.g., experiments) and qualitative (e.g., interviews) analysis.

1.1. Problem Statement

Today, software companies developing mobile apps make their users aware of new updates by implementing weekly or even daily releases. Although practices such as continuous delivery aim to simplify release planning, the development team makes decisions, like who is the release intended for, what functionalities or features should the release have, when should the release happen and how much quality should the release have. In this research, we assign features to mobile releases and suggest the features for the next release. There are many approaches to plan releases (Saliu & Ruhe, 2005; Ruhe, 2010; Greer & Ruhe, 2003; Scalabrino et al., 2019). But the use of these approaches in the mobile context has little attention in the literature.

Although there are many studies in extracting app-reviews to help or support development practices, there are little studies on how app-reviews helps to support mobile release management. In this regard, we formulate three research questions. The first research question is about our release planning approach, while the second and third are questions about evaluating our release planning approach:

> RQ1: How can we combine information extracted from app-reviews and issue trackers to support the release planning of mobile apps?

RQ2: What is the performance of the existing release planning models in the context of mobile applications?

RQ3: To what extent is the proposed approach useful in practice as perceived by developers?

1.2. Outline

This research is organized into chapters as follows: Chapter 2 is the background of the release planning problem, where we discuss previous and current release planning issues. Chapter 3 is a review of the literature on release planning, where we described the method used in searching the literation and reviewing the literature. We asked and answered four literature review questions (LRQ). In Chapter 4**,** we discuss the release planning approach. In Chapter 5**,** we evaluate and report results from the release planning experiments. In Chapter 6**,** we discuss our findings and views. Chapter 7 identifies and discusses the limitations of our research. Finally, Chapter 8 summarizes and highlight our contribution to the release planning of mobile apps.

2. Background

In this chapter, we present the context and highlight the significance of our study. It includes describing mobile application development, user app-review and giving an overview of the release planning of mobile apps.

2.1. Mobile Application Development

Mobile apps are computer programs, usually developed by mobile app developers, which can run on any mobile device (e.g., tablets and phones). Unlike web and desktop apps, there exist generally accepted marketplaces (e.g., Play, App and Windows store), where end-users can download and install apps on their devices. Developers usually go through a series of steps to develop and publish a mobile app on the marketplace for commercial or non-commercial purposes. On the other hand, end-users install apps on their devices to use available services. Usually, any app providing some services competes with other similar apps available in the marketplace. Franch & Ruhe (2016) emphasize that the decision process of feature assignment to releases is one of the most critical activities in software product development. Thus, developers need to make difficult decisions before releasing a new version of an app.

2.2. User App-Review

Users also have the opportunity to rate (on a scale of 1 to 5 stars) and review mobile apps in the marketplace. Figure 1, shows an example of an app-review in the marketplace. The main aim of mobile app ratings and reviews is to give feedback about the experience of users when using the mobile app, in other to help other users make a decision about which mobile app product to use.

Mobile app discoverability can also be related to user ratings and reviews. Developers have the opportunity to address user app-reviews and increase user experience. Therefore, mobile app ratings and app-reviews can influence the willingness of new users to install and use the app.

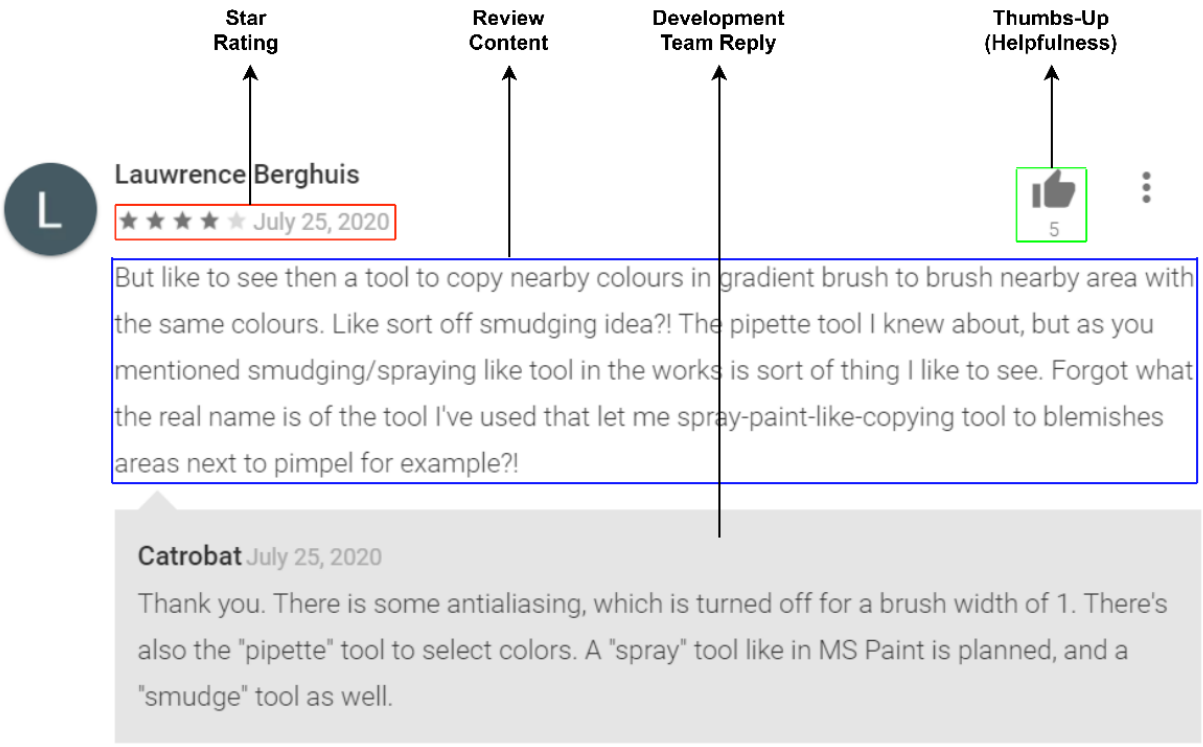

Figure 1. App-review Sample

2.3. Release Planning of Mobile Apps

Developers plan their releases through a process called release planning. This involves all the approaches, strategies and decisions employed by a team to ship features to the end-users (Nayebi et al., 2016a). In mobile development, many factors such as the mobile app marketplace, marketability, nature of availability and access to new updates makes the release planning of mobile apps different from traditional software apps and web services (Nayebi et al., 2017). Also, release planning in web and other traditional software product is well established compared to the release planning of mobile apps (Ruhe, 2010). Thus, it is important and necessary to fit general release planning approaches to the mobile context while considering unique factors in the mobile ecosystem.

Different aspects of release planning for mobile apps have already been studied in the past. Nayebi et al. (2016a) highlight certain release strategies employed by developers and how users react to new releases. Nayebi & Ruhe (2017) provide an optimized solution approach based on data analytics for reuse of features and their cohesiveness in mobile releases. CLAP heavily relies on app-reviews to recommend the next release. Nayebi et al. (2016b) perform a study on mobile releases using semantic versioning from open-source apps. Ciurumelea et al. (2017) investigate important topics written by users in app-reviews that may be important when developers are planning mobile releases. A detailed study of the continuous deployment process for the cloud-based software was conducted using Facebook (Rossi et al., 2016).

Some of these studies rely on user app-review information to drive release planning, but none of the current studies automates and simplifies the derivation of decision variables by combining app-review information (e.g., play store) and issue reports (e.g., Jira software) from the issue tracker to support release planning of mobile apps. Our release planning approach for mobile apps is well documented and its applicability has been verified using an open-source mobile app. We also provide a prototype of the mobile app planner to easily support planning in the mobile context for mobile development teams.

3. Related Work

This section presents a systematic literature review (SLR) that relies on guidelines proposed by Kitchenham et al. (2004). The SLR has three phases that include planning, conducting and reporting the review.

3.1. Planning the Review

We show the need for the review and develop the review protocol. Kitchenham et al. (2004) suggest that "the need for a systematic review arises from the requirement of researchers to summarise all existing information about some phenomenon in a thorough and unbiased manner." This is required as a prelude to our research in release planning. During the review protocol, an important aspect is formulating the research questions, which we shall address in the next section.

3.1.1. Literature Review Research Questions

To help us have an idea of the work that has been done in release planning, we have formulated four (4) literature review research questions (LRRQ) as follows:

LRRQ1: What are the current decisions and strategies employed by developers in release planning?

The decision variables, tools and strategies currently used by software engineers in release planning are discussed.

LRRQ2: How does release planning contribute to the success and quality of mobile apps?

The results obtained concerning success and quality gained as a result of release planning are summarized.

LRRQ3: What are the well-known methods of extracting and classifying user app-reviews into a bug, new feature or improvement?

Tools, ideas and approaches currently taken in mining, refining and classification of mobile app-reviews data are highlighted.

LRRQ4: What are the current methods for creating a release plan?

Methods currently used in the software industry to create release plans are described.

3.1.2. Review Protocol

The following databases were thoroughly searched for literature:

IEEE Xplore

ACM Digital Library

This is the final search query string gotten from the LRRQs:

(((mobile) AND (release) AND (planning)) OR ((mining) AND (user) AND (review)))

3.2. Conducting the Review

After finding relevant studies to help address our LRRQs, these are some of the inclusion and exclusion criteria applied.

IN1: The study is focused on the strategies and decisions used in release planning. This fulfils LRRQ1.

IN2: The study is focused on how release planning has contributed to the success and quality of mobile apps. This fulfils LRRQ2.

IN3: The study is focused on user review extraction and classification techniques. This fulfils LRRQ3.

IN4: The study is focused on methods of creating a release plan. This fulfils LRRQ4.

EX1: The study must be written in English.

As a result of the study searching process, two hundred and twenty-three (223) articles and publications were returned from the search query (IEEE Xplore database, for example). The selection process was based on the title of the publication and reading the abstracts. We also employed a snowboard approach to further get more related publications, by further looking into the references of the articles and publications.

3.3. Reporting the Review

The publications helped us answer our LRRQs; the next section provides a detailed report of these answers.

3.3.1. Results

LRRQ1: What are the current decisions and strategies employed by developers in release planning?

Software teams usually have to make decisions of what should be released, when the release should happen, who the release is for and how much quality the release should have. Usually, companies keep a backlog of prioritized requirements and make decisions of what requirements will be included in the next release. Teams have the opportunity to evolve based on information from previous decisions (past data) and discussion with all stakeholders. Some of the approaches and

strategies used by developers in release planning of mobile apps include; (1) Time-based approach; where releases are done weekly or bi-weekly or yearly. (2) Marketing consideration approach; where promotions and blog spill outs contribute to what is contained in a release. (3) Quality (test) driven approach; where value is placed on testing and quality assurance which drives the releases. (4) Feature-based approach; where releases usually contain implemented features. (5) Size-based strategy; where the release is based on the size. (6) Occasional strategy; where releases can be based on the launch of a new device. Nayebi et al. (2016a) report that developers follow the time-based, marketing, quality, feature-based, size-based and occasional approach at 80%, 40%, 25%, 20%, 10% and 5% respectively. It should be noted that some teams explicitly define these strategies at the beginning of the development phase; others follow a more intuitive approach. Saliu and Ruhe (2005) highlight key aspects of release planning which include the release scope, time horizon of the release, release plan objective, stakeholders involved, feature prioritization techniques and constraints. These aspects represent different areas where developers need to make decisions in release planning. Thus, it can be seen that to some extent software teams employ strategies in planning their releases.

LRRQ2: How does release planning contribute to the success and quality of mobile apps?

Software quality can be defined as the degree to which an application conforms to requirements. SWEBOK (2014), refers to software quality as "desirable characteristics of a software product, to the extent to which a particular software product possesses those characteristics and to processes, tools and techniques used to achieve those characteristics." Success can also be seen as the degree to which an application helps the user to fulfil goals. Fhang et al. (2018) emphasize that with a high frequency of releases, quality checks must be put in place to ensure releases conforms to industrial standards.

Furthermore, Rossi et al. (2016) highlight the importance of testing mobile applications at Facebook Inc. They explain that thousands of features are contained in each release and made available weekly. They also noted that the combination of devices and operating system (OS) versions that mobile apps run on are numerous. Thus, issues that can arise after the app has been released cannot be easily mitigated, so they employ different types of testing. At Facebook, they have seen that investing a lot in their mobile release management affects the quality and success of the app. As a result, the number of critical issues reported is low because of the quality checks put in place and also because critical issues are taken seriously, as this can affect overall success and quality of the app. Therefore, the quality of release planning contributes to success and tends to affect the quality of mobile applications.

LRRQ3: What are the well-known methods of extracting and classifying user app-reviews into a bug, new feature or improvement?

It is important to have an effective way of mining and classifying data from the IOS App Store and Play Store effectively and accurately. We have searched through GitHub repositories to discover work individuals have done to extract user app-review information and we came across a repository that uses HTML tags to mine app-reviews from Play Store[1]. Meng and Wang (2009)

[1] https://github.com/JoMingyu/google-play-scraper

propose a simple approach to extract features using product specification tree and user reviews (Figure 2).

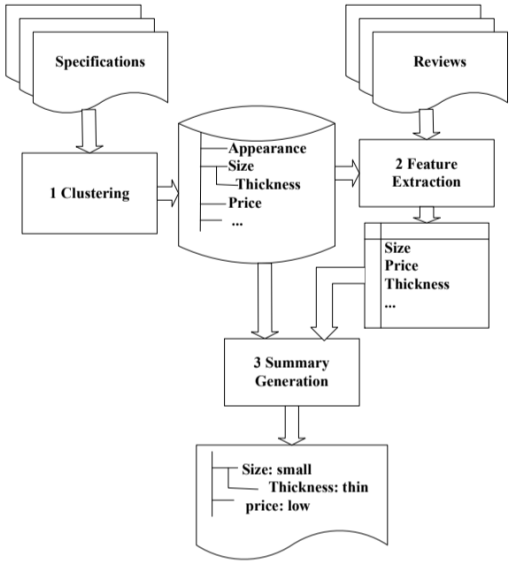

Figure 2. Extracting features from user review (Meng & Wang, 2009).

These authors choose to use product specification because it reduces noise in words; it is nicely structured and can be used as a unit of measure. Finally, CLAP is an approach that automatically categorizes user app-reviews into bugs, new features, performance issues, security issues and other. CLAP further cluster related app-reviews and suggest which cluster should be considered for the next release. A combination of text pre-processing, categorization using random forest, clustering using DBScan and prioritization using random forest was used (Villarroel et al., 2016; Scalabrino et al., 2019). Finally, AR-Miner is another approach of extracting valuable user app-review information and ranking these reviews based on importance (Chen et al., 2014). Here, it can be seen that a lot of research has gone into finding the best and most accurate approach to extract user app-reviews.

LRRQ4: What are the current methods for creating release plans?

Different methods, ideas and approaches exist in planning releases for software development projects. Agile release planning emphasizes communication and is usually limited to the next release. Jira provides tools that support agile practices. These tools are widely used across the software industry. However, release planning in agile development does not suggest ways to handle conflicts between multiple stakeholders (Ruhe & Saliu, 2005). Ruhe and Saliu (2005) suggest an approach that combines human and computational intelligence to design a model that formally defines release planning as a problem based on decision variables and an objective function for assigning features to multiple releases. Finally, Greer and Ruhe (2003) describe an evolutionary and iterative approach called EVOLVE, which generates multiple releases and offers decision support for software release planning. The approach also considers refinements and extensions from the previous iterations in the next iteration. (Ruhe) 2010 describes other methods and approaches. Thus, some release planning methods already exist.

Furthermore, the table below shows a comparison of the different methodologies involved in release planning (Table 1). These include;

- *Estimation-Based Management Framework for Enhancive Maintenance* (EMFEM): A mathematical based model for release planning (Penny, 2002).
- *Incremental Funding Method* (IFM): A business value driven model (Denne, 2004).
- *Cost-Value Approach for Prioritizing Requirements* (COVAP): A cost-value driven approach used in prioritizing features for the next release (Karlsson, 1997).
- *Optimizing Value and Cost in Requirements Analysis* (OVAC): Another cost-value driven approach for requirements selection to be included in the next release (Jung, 1998).
- *The Next Release Problem* (NRP): Model uses modern heuristics to formulate the next release, not compromising quality (Bagnall et. al., 2001).
- *Planning Software Evolution with Risk Management* (PSERM): An approach to support release planning decision making (Greer, 2004).
- EVOLVE* (Ruhe et. al., 2004).

Table 1. The comparison of various release planning methodologies; source: (Saliu & Ruhe, 2005).

RP Methods / Dimensions	PSERM	EMFEM	NRP	NRP	COVAP	IFM	EVOLVE*
Scope	1 release	1 release	1 release	1 release	1 release	Chunks of small releases	2 releases planned ahead
Time horizon	Fixed release	Fixed release	Fixed release	Fixed release	Fixed release	Fixed release	Fixed release
Objectives	Based on benefit of system changes	Based on benefit of features to customers	Based on weight of customers	Based on value of requirements	Based on customer satisfaction	Based on return on investment	Based on value, urgency, stakeholders weights and satisfaction
Stakeholders involvement	Project manager	Developers, project management	Project manager, customer	Involvement of project manager is implied	Project manager, customer, users	All major stakeholders	All major stakeholders
Prioritization mechanism	Optimization-based	No defined prioritization scheme	Optimization - based	Optimization	AHP	IFM Heuristics	Stakeholders prioritize features; Optimization heuristics used to balance conflicts
Technological constraints	Not available	Not available	precedence	Not available	Not available	Precedence (precursor)	Coupling and precedence
Resource constraints	Cost, risk	Effort	Cost	Cost	Cost	Cost, time-to-market	Effort, risk, schedule
System constraints	Operational risks	Not available	Not available	Not available	Not available	Not available	Not available
Character and quality of solutions	One solution plan	One solution plan	One solution plan by any chosen search algorithm	One solution plan generated	One solution plan provided	One plan spanning many release periods	Several alternative solution plans are provided. These plans are diversified and fulfill some target quality level.
Tool support	Not available	Time-tracking system	Not available	Not available	Not available	Partially available	ReleasePlanner ®

3.4. Conclusion

It is clear from the above systematic review, that there exists literature for (1) decisions made by developers in release planning, (2) success stories in release planning, (3) classification approaches for app-reviews and (4) release planning methods. But the area of simplifying release planning of mobile apps, by relying on app-review information and issue reports from the issue tracker have received little or no attention in the literature known to us.

4. Methodology

In this section, we first give an overview of our release planning approach. Then, we describe the main phases of the approach: Data collection and Optimization.

4.1. Overview of the Approach

Our release planning approach for mobile apps consists of four phases, which include (1) Data Collection, (2) Optimization, (3) Software Development and (4) Delivery. Figure 3 illustrates the phases of the approach to automatically create a release plan of a given mobile app.

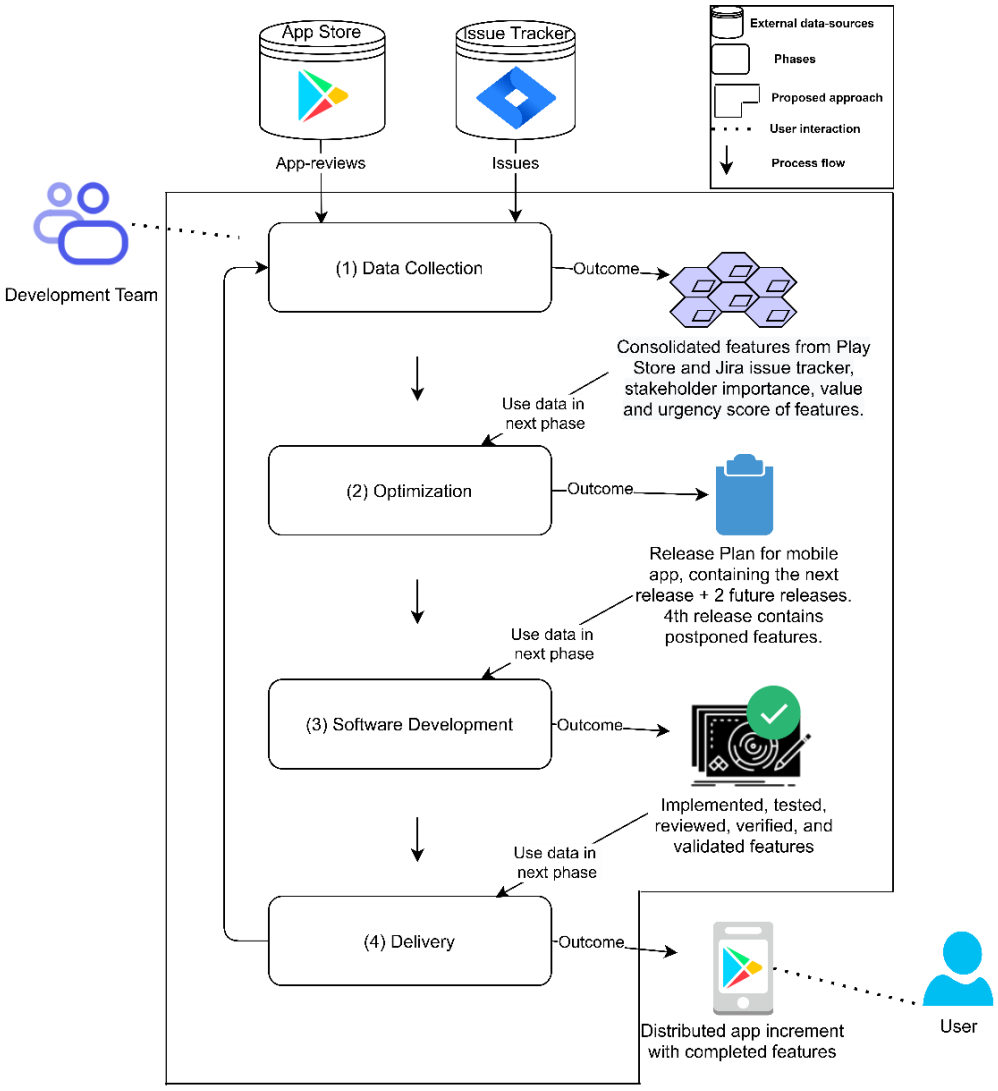

Figure 3. Mobile Release Planning Process

In the data collection phase (1), we use app-reviews extracted from the app store and issue reports from the issue tracker as our primary sources of information. After collecting the app-reviews, we combine several machine learning approaches to (1.1) classify the sentences of the app-reviews into features, bugs and other class. (1.2) extract the features from the review sentences in a human-readable form. For example, "*add pen tool*" extracted from "*Wow, this app very useful and user interface also, please add graph function to draw more easily and also add pen tool feature* 👌😁". Also, we collect features from the issue tracker to have decisions made by the development team. The issue reports are not classified, since each issue report is already manually labelled by the development team. The title/summary of each issue report is pre-processed to extract a feature description in a compatible format. For example, "*paste tool*" is a feature extracted from the issue report [PAINTROID-162] of type "Story". Figure 4, illustrates an issue report.

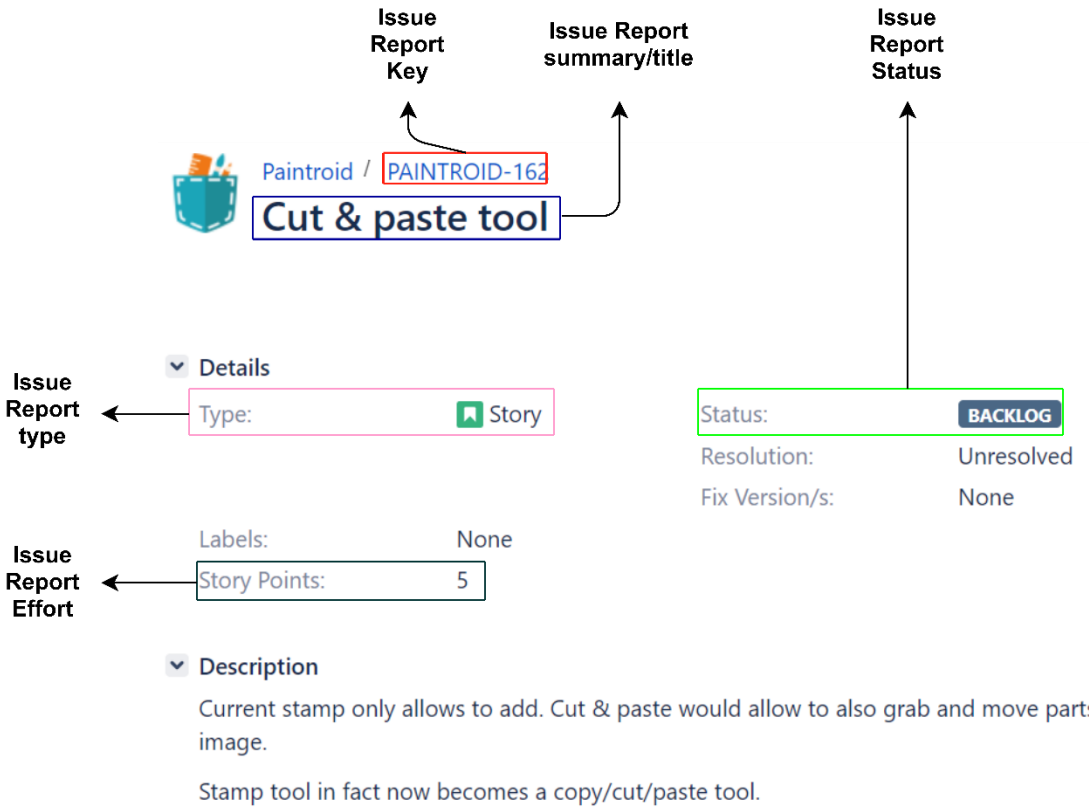

Figure 4. Issue Report Sample

Once the features are collected from both the app-reviews and the issue tracker, we consolidate the list of features by removing duplicated or already implemented ones and automatically calculate decision variables (priority, effort and value) using information from app-reviews and the issue tracker. The consolidated list of features is used as an input in the (2) Optimization phase to generate the optimal release plan regarding the satisfaction of all the involved stakeholders. In this phase, a genetic algorithm is used to create the plan.

Phases (1) and (2) of the proposed approach are fully automated and its final output (i.e., the generated release plan) can be used by the development team in phase (3) Software Development, to implement the features described by the release. In this phase, other software development practices such as verification, validation and code review can be carried out. Finally, during phase (4) Delivery, the development team can deploy the implemented features and make the mobile app available to the end-user. This process can be repeated until all features are implemented, as much as iterations are needed. Each phase is described in detail in the following sections.

4.2. Data Collection Phase

The main goal of the data collection the phase is to use machine learning approaches to extract and combine the features from the selected sources (i.e., the issue tracker and the app store). During this phase, the features are identified, cleaned, pre-processed and selected properly. The steps included in this phase are (1) Data extraction from both sources; (2) App-review sentences classification into Bug Report, Feature Request, Feature Evaluation, Praise and Other; (3) Feature extraction from app-reviews and issue reports; (4) Feature consolidation, including similarity check, deriving the values/scores of the decision variables and removal of repeated and already implemented features, since they should not be considered as part of the generated release plan.

These steps are performed in a way automatically. However, the approach allows the development team to manually set up several variables such as identifying and scoring the most important stakeholders, setting up the value and urgency of features from each stakeholder and assigning efforts to features. The result from this phase is an input in the optimization phase.

4.2.1. Data Extraction

To collect the app-reviews from the app store, we use Google App Scraper Python tool[2]. Each extracted app-review contains an app-review identification number, the username of the reviewer, the content of the review, the score of the review (rating in stars), the number of thumbs-up from other users on the review, the version of the app when the review was made, the time the review was created, the reply from the mobile app team and the time a reply was made on the review by the mobile app team. Our approach only uses the identification number, the content of the review, the score of the reviewer (rating in stars), the number of thumbs-up from other users on the review and the time the review was created.

For collecting the issue reports of the mobile app, we use a Python script[3] that connects with a Jira server instance and downloads all the issue reports in that server instance. Each issue report consists of an issue key, the created date, its description, summary, type name (e.g., Story or Bug), status (e.g., Backlog, Ready for Development, Rejected, or Merged) and key category (e.g., new or done) to mention a few. In this approach, we use the following fields: issue key, created date, summary, type name, status and the custom field that stores the effort estimate.

[2] https://pypi.org/project/google-play-scraper/
[3] https://github.com/ezequielscott/jiraextractor/

4.2.2. Classification of App-Reviews and Issue Reports

Once the app-reviews are extracted, we should find if their sentences contain descriptions of bugs, features, or any other praise. To do this, we use the solution proposed by Shah et al. (2018). The solution performs a sentence wise app-review classification experiment, classifying app-review sentences into a bug report (B), feature request (R), feature evaluation (E), praise (P) and other (O). In their work, the authors conducted several experiments to compare the performance of the bag of words (BoW) feature-based models with the CNN-based models. The maximum entropy (MaxEnt) model with BoW features was found to be very competitive, performing slightly better than the CNN-based model. In another experiment, they found the MaxEnt with Character N-Grams (BoC) and linguistic feature to have the best precision and f1-score (Shah, 2019). Therefore, we adopted this model for classifying the app-reviews. To ensure the model receives sentences, we tokenize every app-review text by using the natural language toolkit (NLTK). Once the sentences are classified into bug report (B), feature request (R), feature evaluation (E), praise (P) and other (O), the sentences related to the classes P, O and B were discarded. As a result, only the app-review sentences in class R and E are used during the next step.

For the issue report from the issue tracker, we take only issue report of type “Story”. Since they are already stories and contains information enter by the development team we do not classify them.

4.2.3. Feature Extraction from App-Reviews

To extract the features from the classified app-review sentences, we use SAFE (Johann et al., 2017). According to Johann et al. (2017), this technique outperforms other state-of-the-art approaches. A replication study was conducted by Shah et al. (2019), so we used the replicated model to extract features from the app-review sentences. The feature consists of a two-word sentence (or more) that represent a feature.

Regarding the issue reports extracted from the issue tracker, we only pre-process the field “summary” by removing pronunciations and stop words, for consolidation. This field was not further sent to SAFE for feature extraction, because this information source is directly from the development team.

4.2.4. Feature Consolidation by Similarity Check

The goal of this step is to merge the features extracted from the issue reports with the features extracted from the app-review sentences into a single set of features. The consolidation step requires; (1) automatically deriving the decision variables (priority, value and effort). (2) removing duplicated or implemented features as a result of merging features from both sources (app-store and issue tracker). For example, issue reports labelled as “Done” or “Complete” is not included for planning. The process of removing duplicated features is not simple since we are comparing sentences. For example, consider the following feature sentences (FSs):

- FS1 - “dynamic color scheme”
- FS2 - “colour shades”
- FS3 - “save dialog”
- FS4 - “add graph”
- FS5 - “image file size editing”
- FS6 - “resize pictures”

We consider FS1 and FS2 to be similar, FS3 and FS4 to be different and also FS5 and FS6 to be different. There are different ways to check the similarity between two sentences. One of the simplest ways is to use similarity measures such as cosine similarity (1) and Jaccard (2). More complex models can also be used to output sentence embeddings, which can be used to get a better cosine similarity score. For example, Sentence-BERT (Reimers & Gurevych, 2019) is a fine-tuned modification of the pre-trained state-of-the-art BERT (Devlin et al., 2019) model, which generates better sentence embeddings.

$$similarity = \cos(\theta) = \frac{A \,.\, B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

$$similarity = J(sentence_1, \quad sentence_2) = \frac{|sentence_1|}{|sentence_1| + |sentence_2| - |sentence_1 \cap sentence_2|} \tag{2}$$

4.3. Optimization Phase

The optimization phase is the second phase of the release planning process (Fig. 3) and its goal is to generate the best release plan that balances all the stakeholders’ satisfaction. Although the generated release plan is optimal, the plan can be further evaluated and modified by the team according to their needs. The optimization phase takes the result of the first phase as input and uses an optimization algorithm to generate a set of four release plans. The first plan contains the next release. The second and third plans are the two next releases. The fourth plan contains postponed features, which could not be planned due to limited resources or maybe have low satisfaction in the view of stakeholders.

Next, we highlight two optimization algorithms; Genetic Algorithm (GA) and Linear Programming (LP), the required inputs and the objective function used in this research.

4.3.1. Optimization Algorithms

There are several models that aim to create release plans (Saliu & Ruhe, 2005; Ruhe, 2010; Greer & Ruhe, 2003). Among them, we selected the following models for comparison:

1. Release Planner using Genetic Algorithm (GA); using the same fitness function as in (1) above, selection, crossover and mutation approach described in EVOLVE. In addition, we implemented partial matched and edge recombination crossover and tournament and proportionate selection.

2. Release Planner using Linear Programming (LP); as described in (Saliu & Ruhe, 2005). Features are assigned linearly to releases based on stakeholders' biggest weighted average satisfaction (WAS) of assigning the feature to the first release.

All implementations use the same fitness/objective function (4) below.

(1) Genetic Algorithm (GA)

Genetic algorithms (GAs) are evolutionary-based algorithms and an artificial intelligence technique (Perhinschi, 2017). This technique is used for searching near-optimal or optimal solutions. GAs was inspired by the concept of "survival of the fittest" and the process of evolution through reproduction by Charles Darwin (Elbeltagia et al., 2005).

Every GA problem must be represented with the appropriate encoding scheme since the scheme is the data structure that represents our problem in the GA space. We adopted permutation encoding in this research (Figure 5) because values or features (allele) in our chromosome cannot be repeated and our space is the total number of possible permutation. Other examples of encoding schemes include binary and list encoding.

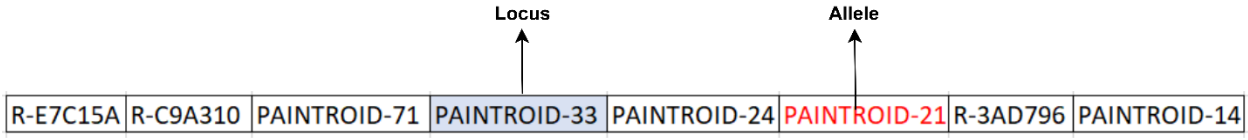

Figure 5. Chromosome or Solution

GAs require an initial population of chromosomes or solutions, generated stochastically. Greer et al. (2003) suggest that a large population size improves the probability of obtaining a better solution, but this comes at the expense of computational time.
Each chromosome in the GA space is scored for fitness, which is given by the objective or fitness function (4). Through the process of selection, crossover and mutation the best chromosome is obtained by running the GA n-times. Table 2 shows the additional parameter required by the GA.

Table 2. *GA parameters.*

Parameter Name	Type	Description
crossover_rate	float	Probability of performing crossover operation on two selected chromosomes.
mutation_rate	float	Probability of performing mutation operation.
inputs	dataframe	Set of features from csv file
max_cycles	int	Maximum number of iteration for the GA.

cross_type	string	Crossover operation to use. E.g. Ordered
select_type	string	Selection operation to use. E.g. Tournament.
population_size	int	Size of population in GA space.
auto_termination	bool	It determines whether auto termination functionality of the algorithm is turned on or not. If selected, the algorithm will terminate when the percentage of the scored and sorted population has the same fitness score. The percentage is given by the parameter population_percentage.
population_percentage	float	Percentage of optimal solutions to observe, when auto termination is turned on.

Below is the pseudocode for the GA model. The complete implementation and documentation for the GA model are available on our GitHub[4] repository.

GA model pseudocode

```
DEFINE FUNCTION solve():
    SET start TO current_time()
    initialize_new_population(seed)
    SET terminate_flag TO False
    SET cycles TO 0
    TRY:
      WHILE not terminate_flag:
        SET offspring TO perform_ga_operation()
        SET score TO evaluate(offspring)
       IF NOT exist(offspring):
          Append_to_seed_population(offspring)
          perform_cull_operation()
        SET terminate_flag TO check_termination()
        cycles += 1
    except KeyboardInterrupt:
      PASS
    SET end TO current_time()
    RETURN max()
```

[4] https://github.com/geegog/MobileReleasePlanner/blob/master/planner/ga.py

(2) Linear Programming (LP)

According to Lewis (2008), linear programming was developed during World War II, when there was a high need to maximize the efficiency of resources. The term "programming" was used by the military to refer to activities like planning schedules efficiently and optimally deploying soldiers. Linear programming consists of an expression to be optimized: this expression is called the objective function. It also has variables called the decision variables, in which the values of the variables must satisfy all constraints. Our release planning problem is to maximize the objective function defined in (4). The table below (Table 3) shows the parameter required by the LP model in addition to the model inputs defined in section 4.3.2.

Table 3. ***LP parameters.***

Parameter	Type	Description
is_sorted	bool	Sort features in descending order based on WAS of 1st release or shuffle.
highest	bool	Flag specified if feature should be chosen randomly or based on the highest WAS.
inputs	dataframe	Set of features from csv file

Based on the WAS of the first release, we sort the features in descending order and linearly assign each feature to a release considering all the defined constraints. Below is the pseudocode used for the feature assignment. The complete implementation and documentation for the LP model are available on our GitHub[5] repository.

LP model pseudocode

```
DEFINE FUNCTION assignment_function(feature_array_with_calculated_was):
    IF is_sorted:
      feature_array_with_calculated_was TO sorted(feature_array_with_calculated_was)
    ELSE:
      shuffle(feature_array_with_calculated_was)
    FOR feature_tuple IN feature_array_with_calculated_was:
      IF feature_tuple is not None:
        IF highest:
          SET max_feature TO get_max_was(feature_tuple)
        ELSE:
          SET max_feature TO get_random_was(feature_tuple)
        SET couple_key TO is_coupled_with(max_feature)
        IF couple_key is not None:
          handle_assignmnent_if_coupled_feature_logic()
        ELSE:
          assign(max_feature, feature_tuple)
```

[5] https://github.com/geegog/MobileReleasePlanner/blob/master/planner/lp.py

4.3.2. Model Inputs.

In this section, we rely on ideas presented by (Saliu & Ruhe, 2005; Ruhe, 2010; Greer & Ruhe, 2003). Table 4 gives a summary of all the inputs required by both models (LP and GA). Table 5 shows how a feature's value, priority and effort can be derived by combining app-review and issue tracker information. For deriving the value and priority of features extracted from app-review sentences, we solely rely on the explicit ratings and thumbs-ups of the users. Lak & Turetken (2014) suggest that sentiment analysis is not a strong alternative to user's explicit ratings and should not be used in place of explicit user ratings. Thus, we rely on these explicit ratings. Table 6 shows examples of issue report statuses. Finally, it should be noted that the app store represents the first stakeholder (user) and the issue tracker represents the second stakeholder (development team).

Table 4. ***Summary of required inputs: Description, example, formal definition and source where data is gotten.***

Input Definition	Input Description	Default Input value	Input Formal Definition	Source of Input
Scope (R)	Number of the releases to plan.	$R = 3$.	$R \in \{1, 2, 3, \dots, n\}$ n is a natural number.	Fixed value in the Models.
Duration (D)	Total effort for each release.	$D = 14$. Story points	$D \in \{1, 2, 3, \dots, \infty\}$.	Manually requested from the user or calculate velocity from the past releases. Velocity = sum (total number of story points for release) / number of releases
Features (F)	A set of features required to be implemented by the team.	None	$F = \{f(1), f(2), f(3), \dots, f(x)\}$ x is a natural number.	App or Play Store, Issue Tracker (Result from phase 2).

Stakeholder (S)	People who have a stake in the app (e.g., CEO, CTO, CIO, COO, Software Engineers, Customers).	(user, developers)	$S = \{s(1), s(2), s(3), \dots, s(b)\}$ b is a natural number.	User and the development team. Play store and issue tracker represents the user and development team respectively.
Importance ($\rho(i)$)	Importance of stakeholders in a team.	(user = 6, developers = 6)	$\rho(i) \epsilon \{2, 4, 6, 8\}$.	Manually requested from the user or use system default.
Release Relative Importance ($\xi(n)$)	Importance of each release.	(0.3, 0.3, 0.3)	$\rho(i) \epsilon \{0.1, \dots, 1\}$.	Manually requested from the user or use system default.
Value ($v(i,x)$)	Value a stakeholder places on a particular feature.	None	$v(i,x) \epsilon \{1, 2, 3, 4, 5\}$.	Manually requested from the user. Calculated from review star rating.
Urgency/Priority ($u(i,x)$)	Urgency a stakeholder places on a particular feature in respect to the scope. It is a vector, such that the first value is the priority a stakeholder place on the next release, the second represents priority placed on the second	None	$u(i,x) \epsilon \{1, 2, 3, 4, 5\}$.	Priority field in Issue Tracker or manually requested from the user. Calculated from thumbs up voting by other users on reviews.

	release and so on.			
Effort ($t(i,x)$)	Estimated effort taken to fully implement a feature.	None	$t(i,x)\ \epsilon\ \{\frac{1}{2}, 1, 2, 3, \dots, \infty\}$.	Effort estimate field in Issue Tracker (Story points assigned to issue).

Table 5. ***Deriving value, priority and effort values.***

	Exist only on Issue Tracker	**Exist only on Store**	**Exist on both Store and Issue Tracker**	**Does not exist on both**
Value	**Developer**: Manual input. **User**: Assign 1.	**User**: Map review star rating to value. If there are more than one similar features, we take the average. **Developer**: 0.5 * calculated user value (take ceiling) or request manually from development team.	Keep calculated user value. Keep calculated or manually request the value from the development team.	**User**: Assign 1. **Developer**: Assign 1 or request manually.
Priority	**Developer**: Extract from priority field in Issue Tracker for the next release and assign 0s for the remaining future releases. **User**: Assign (1, 0, 0).	**User**: Derive from thumbs up associated with review by splitting the maximum thumbs up value in the reviews into 5 parts such that we minimize the difference between the smallest and biggest part. E.g. split (30, 5) return an array of [5, 5, 5, 5, 5] and split (4, 5) returns -1. If the max cannot be split assign 1. If thumbs up count is greater than sum of the first 4 parts; assign 5, if greater than sum of the first 3 parts; assign 4, if greater than sum of the first 2 parts; assign 3, if greater than the first part; assign 2, else assign 1. Assign 0s for the remaining future releases. We take the average priority, if there are more than one similar	Keep priority value from Issue Tracker or manually request priority value from the development team. Keep calculated priority value for User.	**User**: Assign (1, 0, 0). **Developer**: Assign (1, 0, 0) or request from development team.

		features. **Developer**: 0.5 * calculated user priority (take ceiling, if 0 assign 1) and assign 0s for the remaining future releases or request manually from development team.		
Effort	Extract effort from Issue Tracker (Assuming it exists)	Median (effort): Assign 5 story points.	Effort from Issue Tracker.	-

Table 6. ***Summary of default feature statuses.***

Name	Description	Source
TODO	An initial status given to a feature awaiting release planning.	Issue Tracker.
BLOCKED	When external dependencies, limitations, or situations prevent a feature from being implemented.	Issue Tracker.
DONE	Final status given to an implemented feature..	Issue Tracker.

Scope and Total Effort

The scope (R) determines how many releases are generated. Usually, the scope of a release plan is limited only to the next release, but since there is the possibility of obtaining a lot of features from the extracted list of features from app-review sentences, multiple mobile releases can be created. The total effort (D) of each release is given as a whole number. Thus, we define the scope, $R \epsilon \{1, 2, 3, \ldots, n\}$, where n is a natural number. D can be given by the development team or calculated from past release information. The default number of release is $R = 3$.

Features

The model requires a set of prioritized features. $F = \{f(1), f(2), f(3), \ldots, f(x)\}$, where x is a natural number. Features are extracted using SAFE and their priority, value and effort are calculated as explained in Table 5.

Stakeholders

Stakeholders (S) are key drivers of a release plan. Developers, CEO and CTO of a company are examples of stakeholders because they have some stake in the company or product. The set of stakeholders is defined as $S = \{s(1), s(2), s(3), \ldots, s(b)\}$, where b is a natural number and represents the stakeholders. We assign an importance score $\rho(i)$ to all stakeholders involved in the release planning. We use even numbers to assign an importance to a stakeholder such that

$\rho(i) \epsilon \{2, 4, 6, 8\}$. The stakeholders' importance parameter is configurable; otherwise, the system assigns equal importance of 6 to the stakeholders.

Each involved stakeholder has an opportunity to assign a value, $v(i, x) \epsilon \{1, 2, 3, 4, 5\}$ and urgency, $u(i, x) \epsilon \{1, 2, 3, 4, 5\}$ to every prioritized feature. Table 5 shows what values are assigned to a feature's value, urgency and effort if they do not exist in both data sources. Finally, each stakeholder assigns an effort estimate, $t(i, x) \epsilon \{\frac{1}{2}, 1, 2, 3, \dots, \infty\}$, showing the effort required to complete a feature.

Constraints

It should be noted that it is important to engineer features and requirements in a way that they are not dependent on each other and can be implemented without waiting on other features. However, in release planning, we define a coupling constraint (Table 7). It makes sense for some features to be in the same release, for example, login and logout features should be in the same release. Thus, we define a set of coupled features C.

Finally, the sum of feature effort estimates made in a release should be less than or equal to the pre-determine number of duration D.

$$\sum_{n} t(i, x) \leq D \tag{3}$$

Table 7. ***Summary of constraints: Description, example, formal definition and source where data is gotten.***

Constraint	Description	Example	Formal Definition	Source
Coupling (C)	Features that must be in the same release.	$C = \{(f1, f2)\}$.	C	Can be configured in Issue Tracker (e.g., issue links).
Max Effort Capacity of the release.	Total sum of estimated effort in a release must be less than the duration	$13 < 21$.	$\sum_n t(i, x) \leq D$.	-

4.3.3. Model Objective

The goal of both models is to maximize the objective function. We use the objective function described by (Ruhe et al., 2005) in our research because it is easily adaptable to the mobile context. The objective function is given by:

$$F(x) = \sum_{n=1}^{R} \sum_{x} WAS(x, n) \tag{4}$$

Where,

$$WAS(x, n) = \xi(n) \left[\sum_{b}^{S} \rho(\mathrm{i}) * \mathrm{v}(\mathrm{i}, \mathrm{x}) * \mathrm{u}(\mathrm{i}, \mathrm{x}, \mathrm{n}) \right] \tag{5}$$

WAS = Weighted Average Satisfaction of all stakeholder priorities for all features $f(x)$ when assigned to release n.

5. Report and Evaluation of Results

To evaluate our approach, we conducted a case study on an open-source mobile app project available on Google Play Store: Paintroid[6].

The case study was selected as a result of looking for open-source mobile applications on the F-droid catalogue. We sought for projects that meet the following criteria: firstly, the project must have a published mobile app on Google Play Store. Secondly, the mobile app must have reviews posted by users. Thirdly, the development team must be using an issue tracking software to manage their issues and track issue reports (e.g., Jira or GitLab). Fourthly, there must be evidence of effort estimation in the issue tracker since the release planning requires having effort assigned to the issues. Out of about 3060[7] applications published on the F-droid catalogue, we were able to find only one application meeting the requirements: Paintroid.

Using the selected case study, we conducted the following experiments: (1) Feature consolidation by similarity check. (2) Hyper-parameter optimization. (3) Model performance comparison (For example, comparing model fitness, processing time and iterations taken to generate the optimal plan). (4) Comparing the generated plan from the best model with actual release. (5) Interview with Paintroid's release manager.

5.1. Case Study: Paintroid

We carried a study using Paintroid to demonstrate the applicability of our approach. Paintroid, which is also known as Pocket Paint, is associated with Catroid[8]. Catroid is an open-source visual programming language, programming environment, image manipulation program and website that enables teenagers to develop animations and games (Slany, 2012). Paintroid is a graphical paint editor application for the Android platform that, among others, allows setting parts of pictures to transparent.

Using the links to the app published on Google Play and to the issue tracker available on F-droid, we collected 501 app-reviews from Google Play, sorted based on the most relevant from 2013-09-26 to 2020-05-06. The development team uses Jira Software for issue tracking[9]. We collected 162 issue reports from the 2018-10-12 to 2020-05-10.

[6] https://play.google.com/store/apps/details?id=org.catrobat.paintroid&hl=en
[7] https://apt.izzysoft.de/fdroid/?repo=main
[8] https://github.com/Catrobat/Catroid
[9] https://jira.catrob.at/projects/PAINTROID/issues

5.1.1. Results of Classifying the App-Reviews

As described in Fig. 3, the approach requires a set of features. These features can be either extracted from the app-reviews or the issue tracker. To extract the features from the app-reviews, the sentences must be first classified into bug report (B), feature request (R), feature evaluation (E), praise (P) and others (O). We considered only app-reviews from 2019-07-24 to 2020-05-06, so we can work with more recent app-reviews, after the last release date in 2019.

We downloaded and ran the Stanford CoreNLP Server[10] tool on our system. When the server is started, it can be accessed by going to http://localhost:9000/? This tool is required by the evaluated and trained MaxEnt (using BoC) model by Shah (2019), to extract all linguistic textual features (e.g., part of speech) from app-review sentences. The process of obtaining textual features was carried out on app-review sentences, so we tokenized our app-review text using NLTK to obtain app-review sentences (191 sentences). The result from that process was saved in a CSV file. The result from extracting app-review sentences linguistic features can be found here[11]. Finally, the result was inputted into their model to predict the classes of the app-review sentences.

Table 8 contains the result from applying the classification model on the dataset. The resulting classification consists of 5 Bug Reports (B), 10 Feature Request (R), 38 Feature Evaluation (E), 34 Praises (P) and 104 Other sentences (O).

Table 8. ***Classes of app-review sentences.***

E	R	P	O	B
38	10	34	104	5

5.2.1. Extracted Features

Once the sentences were classified into E, R, P, O and B, we selected only those sentences in R and E categories and we passed them to SAFE. As a result, 54 features were extracted by SAFE. Note that SAFE did not extract features from all sentences and some sentences had more than one feature extracted. Table 9 shows a sample of features extracted using SAFE and the original app-review text and the sentence where the feature was extracted from.

To extract features from the issue reports, we only considered issue reports of type “Story” and key category “new”. We also pre-process the field “summary” by removing pronunciations and stop words. Note that we did not apply SAFE to the “summary” field of the issue reports since this information source is directly from the development team. Altogether, 25 issue reports were extracted.

[10] https://stanfordnlp.github.io/CoreNLP/corenlp-server.html
[11] https://github.com/geegog/MobileReleasePlanner/blob/master/data/reviews.csv

Table 9. ***Sample features extracted using SAFE technique for feature extraction.***

Feature Key	Feature Extracted by SAFE	App-Review Text
R-B35486	good range	The best simple paint\drawing app I've found. It's possible to do some quite complex picture editing with a good range of tools, much more than most simple draw apps offer. Minimal permissions, most importantly no Internet permission and no ads. It would be great to have options when saving. Unfortunately it reduces the resolution and removes the EXIF data (most similar apps do the same), else it would be perfect.
R-B737B7	quite complex picture	The best simple paint\drawing app I've found. It's possible to do some quite complex picture editing with a good range of tools, much more than most simple draw apps offer. Minimal permissions, most importantly no Internet permission and no ads. It would be great to have options when saving. Unfortunately it reduces the resolution and removes the EXIF data (most similar apps do the same), else it would be perfect.
R-8E7BE8	move image	This is a great drawing app. I thought it was just going to be one of those stupid little doodle apps that serves no purpose but you guys proved me wrong. The only thing I would improve is that you could select and move the image around. Other than that, it is SUCH a perfect drawing app. 100% would recommend. UwU -DragonTales
R-78EC8E	low resolution	Extremely low resolution when importing images.
R-61612B	infinite painter	Infinite painter is Better

5.2. Consolidation of Features

The goal of this experiment is to determine the best model to use in consolidating the features that were extracted from both sources (e.g., app store and issue tracker).

5.2.1. Set Up

To determine which model is the best in our context, we conducted a short experiment in which we compare the performance of four Sentence-BERT (Reimers & Gurevych, 2019) models combined with cosine similarity measure and Jaccard similarity measure. As a result, we determine bert-large-nli-stsb-mean-tokens + Cosine Similarity as the best model to be used to create a consolidated set of features.

The models are:

1. M1 - bert-base-nli-mean-tokens + Cosine Similarity.
2. M2 - bert-large-nli-mean-tokens + Cosine Similarity.
3. M3 - bert-base-nli-stsb-mean-tokens + Cosine Similarity.
4. M4 - bert-large-nli-stsb-mean-tokens + Cosine Similarity.
5. M5 - Sentence similarity comparison based on Jaccard metric.

In cases where a feature from issue tracker is similar to other features extracted from the app-review sentence, we keep the feature from issue tracker and discard all the other extracted features after we calculate the required field values. If they are different, we include both features.

For comparing the performance of the models, we first manually labelled all the pairwise combinations of features as similar or not. This dataset represents the ground truth. As a result, we obtained 19 pairs of features that are similar and 5381 different. A sample list of paired features can be found in Table 10.

Using the manual labelled dataset, we compare the performance of the models M1-M5 by using standard metrics such as accuracy (6), precision (7), recall (8) and f1-score (9). We consider the best model as the model with the highest f1-score because there is an imbalanced distribution of classes (19 similar vs. 5381 different) and we want a balance between precision and recall.

$$accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + True\ Negative} \tag{6}$$

$$precision = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Total\ Predicted\ Positive} \tag{7}$$

$$recall = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Total\ Actual\ Positive} \tag{8}$$

$$f1_{score} = 2 * \frac{precision * recall}{precision + recall} \tag{9}$$

We also tried different thresholds in other to determine whether a feature pair is similar or not. Thus, we consider two features as similar only if the model score is higher than the specified threshold. The thresholds values evaluated range from 0.50 to 0.95, in steps of 0.1 (e.g., 0.50, 0.51, 0.52, etc.)

5.2.2. Results of Consolidation

Table 10 shows a sample of the manually classified feature pair, which we used in our experiment. The completed dataset is available in the repository[12] and the complete evaluation results of each model is available in Appendix I.

[12] https://github.com/geegog/MobileReleasePlanner/blob/master/similarity/labelled-feature.csv

Table 10. ***Sample of manually classified feature pair.***

Feature 1 (Jira)	Feature 2 (Extracted from App-review sentences)	Label
dynamic color scheme	add more colour	different
dynamic color scheme	color grid	different
dynamic color scheme	color picker	different
dynamic color scheme	colour picker	different
dynamic color scheme	colour shades	similar
paste tool	copy paste	similar
copy paste symbols stamp tool selection application	copy paste	similar
image file size editing	editing large images	similar
export	export my pictures	similar
image file size editing	image editing	similar
correct user intended position tools	original position	similar
paste tool	paste a part	similar
paste tool	paste images	similar
image file size editing	resize picture	similar
image file size editing	resizing entire image	similar
copy paste symbols stamp tool selection application	selection tool	similar
confirm tool application selecting tool	selection tool	similar
hint change default settings tools	settings option	similar
image file size editing	shrinks image size	similar
confirm tool application selecting tool	simple area selection	similar
spray tool	spray tool	similar
dynamic color scheme	add transparency	different
dynamic color scheme	applying a color	different
dynamic color scheme	applying transparency	different

Figure 6 shows the precision, recall and f1-score of running the models M1-M5 along with the different threshold values. Table 11 summarizes the best f1-scores, showing that model M4 performs better than the other models. In consequence, we conclude that M4 with threshold value = 0.71 is the best model for checking the similarity description of two features.

Using M4, we compared the list of features extracted from the issue reports with the features extracted from the app-review sentences. As a result, we obtained a consolidated list of 60 features. The full list of features along with their calculated variables is presented in Table 12.

Table 11. ***Result from experiment.***

Models	Best F1 Score	Threshold
M1 (bert-base-nli-mean-tokens + cosine similarity)	**0.998608**	0.85
M2 (bert-large-nli-mean-tokens + cosine similarity)	**0.998793**	0.82
M3 (bert-base-nli-stsb-mean-tokens + cosine similarity)	**0.998794**	0.73
M4 (bert-large-nli-stsb-mean-tokens + cosine similarity)	**0.999072**	**0.71**
M5 (Jaccard)	**0.99833**	0.7

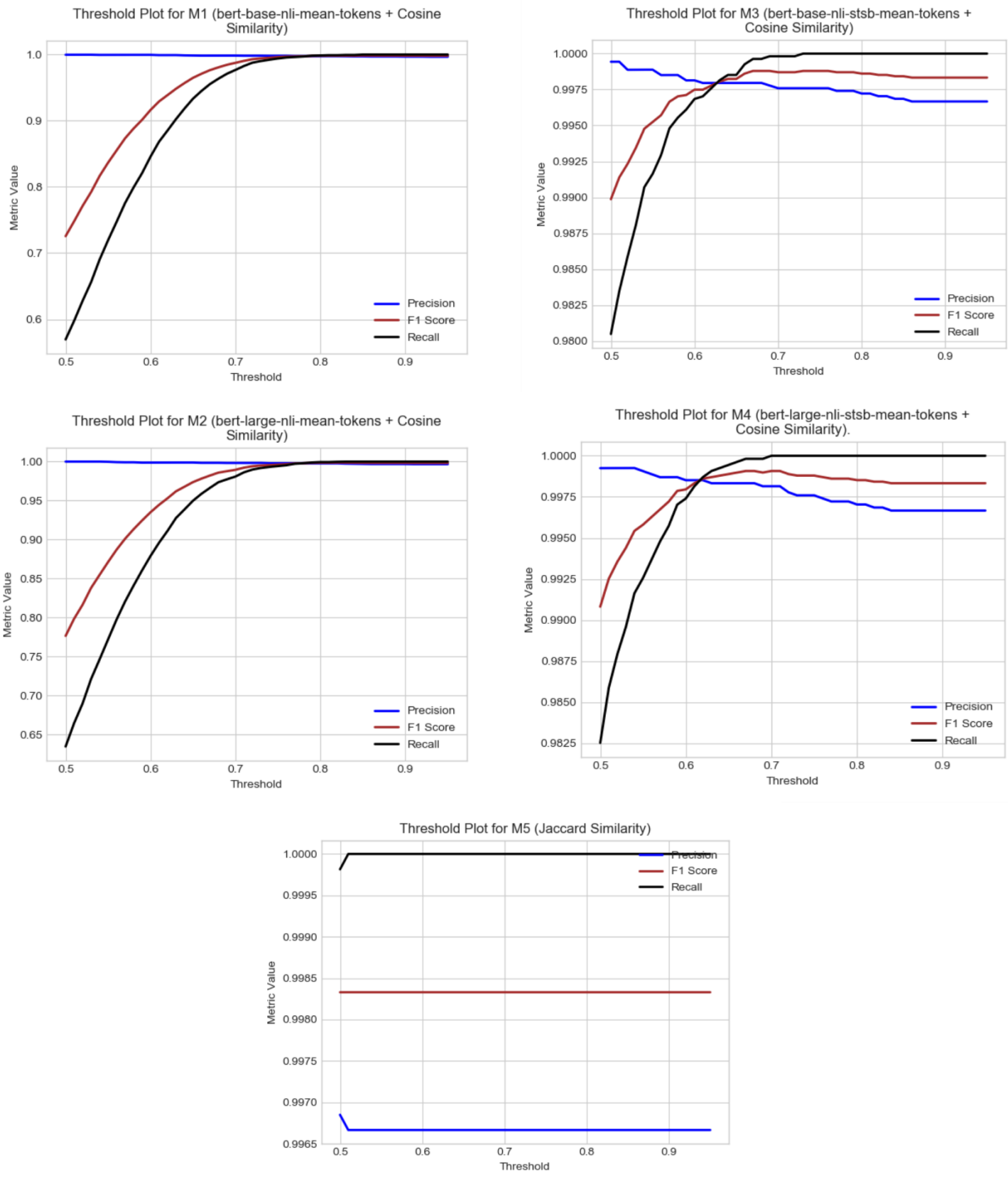

Figure 6. Threshold Plot

Table 12. Consolidated set of features.

Feature Key*	Feature f(i)**	Effort(Story Points) t(i,2)	Stakeholder S (1), Value v(1,i)	Stakeholder S (1), Urgency u(1,i)	Stakeholder S (2), Value v(2,i)	Stakeholder S (2), Urgency u(2,i)
PAINTROID-147	confirm tool application selecting tool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-122	image file size editing	20	1	(2, 0, 0)	1	(1, 0, 0)
R-1DC93C	like paint	5	3	(1, 0, 0)	2	(1, 0, 0)
R-7BE176	microsoft computers	5	3	(1, 0, 0)	2	(1, 0, 0)
R-0EB3CF	load photos	5	4	(1, 0, 0)	2	(1, 0, 0)
R-B84BCA	perfect substitute	5	4	(1, 0, 0)	2	(1, 0, 0)
R-1D5DFA	transparent color	5	4	(1, 0, 0)	2	(1, 0, 0)
R-8E7BE8	move image	5	5	(4, 0, 0)	3	(2, 0, 0)
R-D13C68	dead fast	5	5	(5, 0, 0)	3	(3, 0, 0)
R-4892DC	do some quite	5	5	(4, 0, 0)	3	(2, 0, 0)
R-B35486	good range	5	5	(4, 0, 0)	3	(2, 0, 0)
R-B737B7	quite complex picture	5	5	(4, 0, 0)	3	(2, 0, 0)
R-EF8B24	simple draw offer	5	5	(4, 0, 0)	3	(2, 0, 0)
R-10681F	removes exif data	5	5	(4, 0, 0)	3	(2, 0, 0)
R-80818F	reduces resolution	5	5	(4, 0, 0)	3	(2, 0, 0)
R-2D8EAD	unnecessary permissions	5	5	(1, 0, 0)	3	(1, 0, 0)
R-2BB9BA	wide verity	5	5	(1, 0, 0)	3	(1, 0, 0)
R-4226EF	main issue tools	5	5	(2, 0, 0)	3	(1, 0, 0)
R-94C4ED	resize tool	5	5	(2, 0, 0)	3	(1, 0, 0)
R-3CFAF6	shrink an image	5	5	(2, 0, 0)	3	(1, 0, 0)
R-6C3447	future update	5	5	(2, 0, 0)	3	(1, 0, 0)
R-205482	available font	5	5	(1, 0, 0)	3	(1, 0, 0)
R-84404B	gauging smaller sizes	5	5	(1, 0, 0)	3	(1, 0, 0)
R-7B41CF	paint bucket	5	1	(1, 0, 0)	1	(1, 0, 0)
R-5499A3	edit small things	5	4	(1, 0, 0)	2	(1, 0, 0)
R-78EC8E	low resolution	5	1	(1, 0, 0)	1	(1, 0, 0)
R-915433	pocket code	5	5	(1, 0, 0)	3	(1, 0, 0)
R-22693E	edit photos of gacha	5	5	(1, 0, 0)	3	(1, 0, 0)
R-DFB4D8	add pen tool	5	5	(2, 0, 0)	3	(1, 0, 0)
R-CABB93	user interface	5	5	(2, 0, 0)	3	(1, 0, 0)
R-78C4A2	add graph	5	5	(2, 0, 0)	3	(1, 0, 0)
R-B1C129	picky guy	5	5	(1, 0, 0)	3	(1, 0, 0)
R-174201	satisfy my experience	5	5	(1, 0, 0)	3	(1, 0, 0)
R-51B067	free and powerful	5	5	(1, 0, 0)	3	(1, 0, 0)
R-800B19	image edition	5	5	(1, 0, 0)	3	(1, 0, 0)

R-5E25F6	takes little time	5	5	(1, 0, 0)	3	(1, 0, 0)
R-61612B	infinite painter	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-164	close transform tool options clicking apply	2	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-163	save temporary copies automatically	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-162	paste tool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-161	save load pocketpaint format	20	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-160	save load pocketpaint format	13	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-159	hide unhide layer	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-158	share feature	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-157	feedback menu entry overflow menu	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-156	spray tool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-155	insert image catrobat media gallery	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-154	auto crop icon	2	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-153	hint change default settings tools	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-152	checkmark confirm tool applications	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-151	copy paste symbols stamp tool selection application	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-150	save dialog	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-149	better compression research implementation	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-148	export	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-146	center button transform tool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-145	fastlane support huawei store	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-131	github crowdin integration	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-130	build paintroid pull requests catroid develop jenkins	8	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-126	reuse drawingsurface pipette activity	8	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-121	upload mapping jenkins release builds	5	1	(1, 0, 0)	1	(1, 0, 0)

*** PAINTROID-XXX represents the issue report key, while R-XXXXXX represents the key for a feature extracted from an app-review sentence.**

**** Contains features extracted from issue reports and app-review sentences**

5.3. Hyper-parameter Optimization

Before comparing the performance of the optimization algorithms (LP and GA), it is recommended to search for the best parameters of the GA model. The parameters are mutation rate, crossover rate, selection type and crossover type.

We optimized the parameters of the GA model only since LP does not require optimization of the parameters.

5.3.1. Set Up

Table 13 shows the parameters and values used in this experiment. We explored all the possible combinations of the parameter values (grid search). For example, each crossover rate is matched with all the mutation rates, crossover types and selection types (i.e., 10 x 6 x 3 x 3). In total, we experimented with 540 combinations using two different termination criteria:

- (C1) Auto termination criteria. The algorithm terminates when it converges, that is, when a percentage of the scored and sorted population has the same fitness score.

  For these criteria, we used processing time and the fitness score to choose the best hyper-parameter values.

- (C2) Using a maximum number of iterations (1000), with 10 runs per iteration.

  For these criteria, we considered how many iterations are required for convergence.

These experiments were run on the University of Tartu's high-performance computers[13].

Table 13. Values of parameters used during this experiment.

Crossover rate	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
Mutation rate	0.05, 0.1, 0.15, 0.2, 0.25, 0.3
Crossover type	Ordered, Partially matched and Edge recombination
Selection	Fittest, Tournament and Proportionate
Encoding scheme	Permutation
Population size	50
Observed population used in determining convergence	0.3

[13] https://hpc.ut.ee/en/home/

5.3.2. Results

In our project repository, you can find the complete results for (C1)[14] and (C2)[15]. A total of 540 results were returned from C1. Figure 7 shows the two (2) best results from running all the parameter combinations, based on the processing time and fitness score. According to results from the first configuration, the best parameter values are fittest selection and partially matched crossover with 0.05 mutation rate and 0.7 crossover rate. Using these parameters, the GA model solved our release planning problem in 6 seconds and 142 iterations.

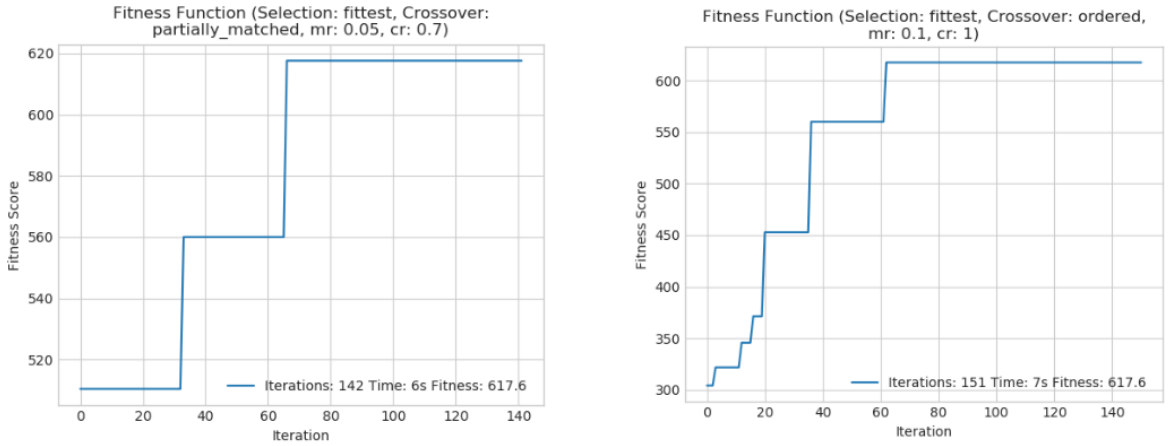

Figure 7. Best Fitness Plot from C1

A total of 90 results were returned from C2. In Figure 8, based on the number of iterations taken for the GA to converge, we selected the nine (9) best results, from each parameter combination. From the nine (9) results, the following performed the best;

(R1-C2) Fittest selection and partially matched crossover with a crossover rate of 0.9 and mutation rates of 0.05, 0.2, 0.25 and 0.3.

(R2-C2) Fittest selection and ordered crossover with a crossover rate of 0.5 and mutation rates of 0.1, 0.2, 0.25 and 0.3.

In R1-C2, only three (3) combinations (with mutation rates of 0.3, 0.25, 0.2 and 0.05) fully converged. From the graphs (Fig. 8), it can be seen that there can still be improvements in the fitness score, for the two (2) combinations with a mutation rate of 0.1 and 0.15. The mutation rate of 0.3, has the fastest convergence in about 250 iterations.

In R2-C2, all combinations, except the mutation rate of 0.05 fully converged. Also, the mutation rate of 0.3, has the fastest convergence in about 190 iterations.

[14] https://github.com/geegog/MobileReleasePlanner/tree/master/planner/exp1
[15] https://github.com/geegog/MobileReleasePlanner/tree/master/planner/exp2

Based on the results from C1 and C2, we chose fittest selection, ordered crossover, crossover rate of 0.5 and mutation rate of 0.3. We considered the result from C2 more, because the result is the average from 10 runs per iteration and the processing time difference between the results of C1 is insignificant.

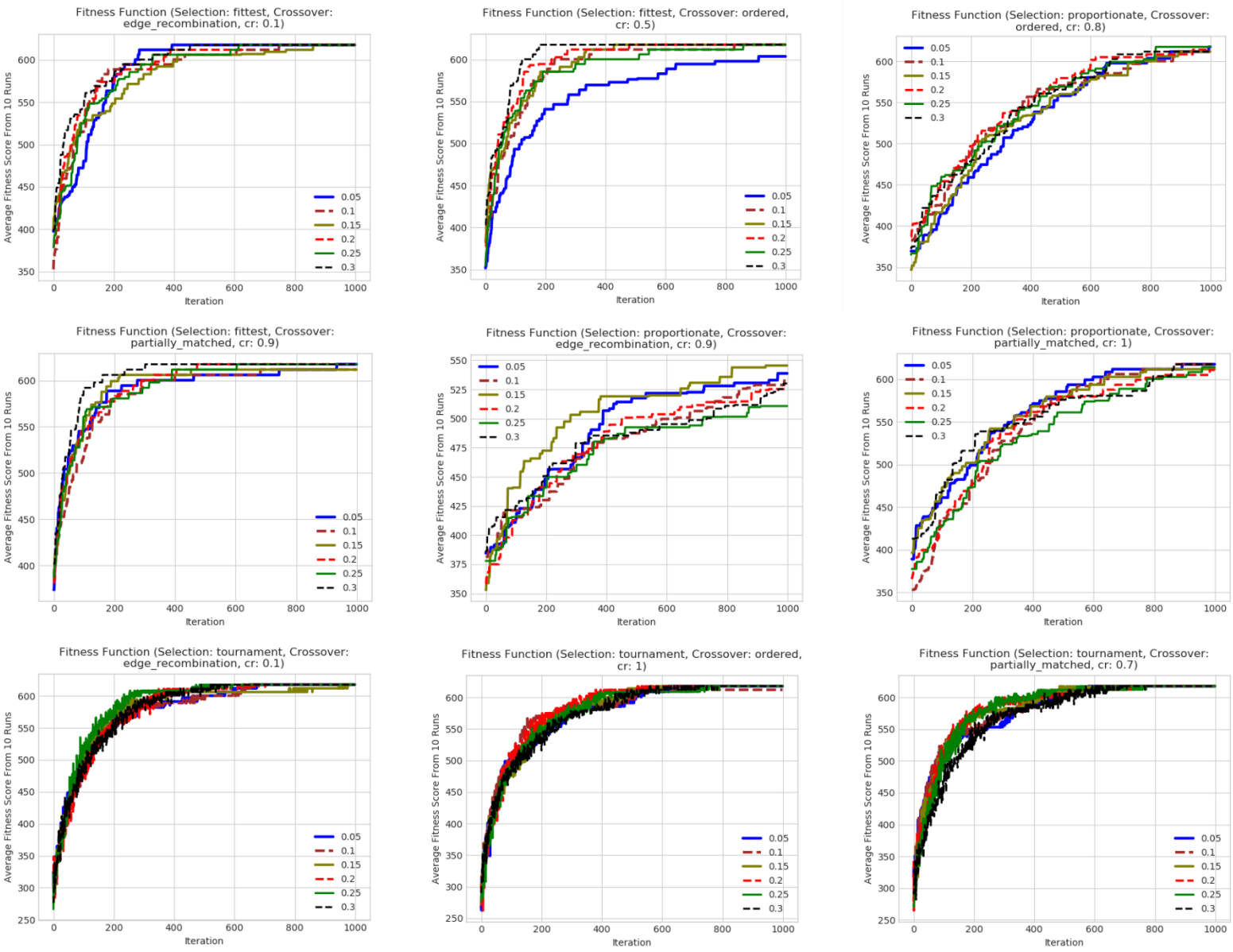

Figure 8. Best Fitness Plots from C2

5.4. Model Performance Comparison

The goal of this experiment is to compare the performance of the Linear Programming (LP) and Genetic Algorithm (GA) model.

5.4.1. Set Up

To compare the performance between the GA and LP model we ran the models using the optimized parameters. The GA model uses the parameter values shown in Table 14 and the LP model the ones shown in Table 15.

Table 14. GA Model

Crossover rate	0.5
Mutation rate	0.3
Crossover type	Ordered
Selection	Fittest
Population size	50
Population percentage	0.3
Auto termination	True
Max cycles	600
Stakeholders importance	(6, 4) user and development team respectively.
Duration	27 story points (calculated from past releases – Feb 7, 2019 to May 28, 2019)[16]
Relative release importance	(0.8, 0.1, 0.1) 1st, 2nd and 3rd release respectively.
Feature input	From Table 12.

Table 15. LP Model.

Is sorted	True
Highest	True
Max cycles	600
Stakeholders importance	(6, 4) user and development team respectively.
Duration	27 story points
Relative release importance	(0.8, 0.1, 0.1) 1st, 2nd and 3rd release respectively.
Feature input	From Table 12.

16 https://github.com/geegog/MobileReleasePlanner/blob/master/data/original_changelog_and_velocity_cal.xlsx

5.4.2. Results

The GA model takes about 170 iterations (5.8 seconds) to solve our mobile release problem with a fitness of 617.6 (Table 16). In contrast, the LP model solved our release problem in less than one second with a fitness of 300.8 (Table 17).

Table 16. Release plan solved by GA model.

Release*	WAS	Key	Feature	Effort (Story Point)
1	115.2	R-EF8B24	simple draw offer	5
1	115.2	R-10681F	removes exif data	5
1	148.8	R-D13C68	dead fast	5
1	115.2	R-8E7BE8	move image	5
1	115.2	R-B35486	good range	5
1	8	PAINTROID-164	close transform tool options clicking apply	2
2	0	PAINTROID-158	share feature	3
2	0	R-DFB4D8	add pen tool	5
2	0	R-0EB3CF	load photos	5
2	0	R-7BE176	microsoft computers	5
2	0	PAINTROID-152	checkmark confirm tool applications	3
2	0	R-78C4A2	add graph	5
3	0	PAINTROID-162	paste tool	5
3	0	R-2D8EAD	unnecessary permissions	5
3	0	PAINTROID-146	center button transform tool	5
3	0	R-800B19	image edition	5
3	0	R-4892DC	do some quite	5
4	8	R-61612B	infinite painter	5
4	57.6	R-6C3447	future update	5
4	33.6	R-51B067	free and powerful	5
4	8	PAINTROID-126	reuse drawingsurface pipette activity	8
4	115.2	R-B737B7	quite complex picture	5
4	25.6	R-B84BCA	perfect substitute	5
4	25.6	R-5499A3	edit small things	5
3	0	PAINTROID-154	auto crop icon	2
4	33.6	R-84404B	gauging smaller sizes	5
4	33.6	R-915433	pocket code	5
4	8	PAINTROID-156	spray tool	5
4	8	PAINTROID-163	save temporary copies automatically	5

4	8	PAINTROID-153	hint change default settings tools	3
4	8	R-78EC8E	low resolution	5
4	33.6	R-205482	available font	5
4	8	PAINTROID-149	better compression research implementation	5
4	57.6	R-3CFAF6	shrink an image	5
4	115.2	R-80818F	reduces resolution	5
4	25.6	R-1D5DFA	transparent color	5
4	33.6	R-2BB9BA	wide verity	5
4	8	PAINTROID-159	hide unhide layer	5
4	8	PAINTROID-121	upload mapping jenkins release builds	5
4	57.6	R-CABB93	user interface	5
4	8	PAINTROID-145	fastlane support huawei store	5
4	8	PAINTROID-155	insert image catrobat media gallery	5
4	57.6	R-4226EF	main issue tools	5
4	33.6	R-174201	satisfy my experience	5
4	33.6	R-B1C129	picky guy	5
4	8	PAINTROID-130	build paintroid pull requests catroid develop jenkins	8
4	8	PAINTROID-160	save load pocketpaint format	13
4	8	PAINTROID-147	confirm tool application selecting tool	5
4	33.6	R-5E25F6	takes little time	5
4	8	PAINTROID-148	export	3
4	12.8	PAINTROID-122	image file size editing	20
4	8	PAINTROID-131	github crowdin integration	3
4	57.6	R-94C4ED	resize tool	5
4	8	PAINTROID-157	feedback menu entry overflow menu	3
4	8	PAINTROID-150	save dialog	3
4	8	R-7B41CF	paint bucket	5
4	8	PAINTROID-151	copy paste symbols stamp tool selection application	3
4	33.6	R-22693E	edit photos of gacha	5
4	20.8	R-1DC93C	like paint	5
4	8	PAINTROID-161	save load pocketpaint format	20
	Effort R1: 27.0	Effort R2: 26.0	Effort R3: 27.0	F(x): 617.6

*** Release 1 contains features that should be released in the next release, while release 2 and 3 are future releases. Release 4 contains all the postponed features.**

Table 17. Release plan solved by LP model.

Release	WAS	Key	Feature	Effort (Story Point)
1	8	PAINTROID-153	hint change default settings tools	3
1	115.2	R-B737B7	quite complex picture	5
1	8	PAINTROID-155	insert image catrobat media gallery	5
1	8	PAINTROID-152	checkmark confirm tool applications	3
2	12.8	PAINTROID-122	image file size editing	20
1	8	PAINTROID-145	fastlane support huawei store	5
1	20.8	R-1DC93C	like paint	5
2	25.6	R-B84BCA	perfect substitute	5
3	8	PAINTROID-150	save dialog	3
3	8	PAINTROID-147	confirm tool application selecting tool	5
3	8	PAINTROID-158	share feature	3
3	25.6	R-5499A3	edit small things	5
4	8	PAINTROID-160	save load pocketpaint format	13
3	8	PAINTROID-157	feedback menu entry overflow menu	3
2	8	PAINTROID-154	auto crop icon	2
3	20.8	R-7BE176	microsoft computers	5
4	115.2	R-B35486	good range	5
4	8	PAINTROID-163	save temporary copies automatically	5
4	8	PAINTROID-149	better compression research implementation	5
4	8	PAINTROID-121	upload mapping jenkins release builds	5
4	33.6	R-915433	pocket code	5
4	115.2	R-10681F	removes exif data	5
4	33.6	R-84404B	gauging smaller sizes	5
4	8	R-78EC8E	low resolution	5
4	8	PAINTROID-156	spray tool	5
4	57.6	R-4226EF	main issue tools	5
4	8	PAINTROID-161	save load pocketpaint format	20
4	8	PAINTROID-130	build paintroid pull requests catroid develop jenkins	8
4	8	PAINTROID-126	reuse drawingsurface pipette activity	8

3	8	PAINTROID-164	close transform tool options clicking apply	2
4	33.6	R-174201	satisfy my experience	5
4	33.6	R-205482	available font	5
4	8	R-7B41CF	paint bucket	5
4	8	R-61612B	infinite painter	5
4	33.6	R-B1C129	picky guy	5
4	33.6	R-800B19	image edition	5
4	33.6	R-22693E	edit photos of gacha	5
4	8	PAINTROID-131	github crowdin integration	3
4	57.6	R-DFB4D8	add pen tool	5
4	115.2	R-80818F	reduces resolution	5
4	57.6	R-6C3447	future update	5
4	8	PAINTROID-148	export	3
4	25.6	R-0EB3CF	load photos	5
4	8	PAINTROID-151	copy paste symbols stamp tool selection application	3
4	33.6	R-2BB9BA	wide verity	5
4	8	PAINTROID-146	center button transform tool	5
4	115.2	R-4892DC	do some quite	5
4	57.6	R-CABB93	user interface	5
4	33.6	R-5E25F6	takes little time	5
4	57.6	R-3CFAF6	shrink an image	5
4	115.2	R-EF8B24	simple draw offer	5
4	33.6	R-2D8EAD	unnecessary permissions	5
4	33.6	R-51B067	free and powerful	5
4	115.2	R-8E7BE8	move image	5
4	148.8	R-D13C68	dead fast	5
4	8	PAINTROID-162	paste tool	5
4	25.6	R-1D5DFA	transparent color	5
4	8	PAINTROID-159	hide unhide layer	5
4	57.6	R-78C4A2	add graph	5
4	57.6	R-94C4ED	resize tool	5
	Effort R1: 26.0	Effort R2: 27.0	Effort R3: 26.0	F(x): 300.8

5.5. Comparison with Actual Plans

The goal of this experiment is to compare the release plan generated from the GA model with an actual release in the past. This will help us to understand if our approach addresses the needs of both the user and the developers. Also, it will help us to determine the accuracy of feature assignments to releases.

5.5.1. Set Up

We extracted the actual plan from April 18, 2019 to May 28, 2019 from the GitHub[17] repository where the features are planned (Table 18).

In order to compare the actual and the generated plans, we ran our approach using the app-reviews created before the date when the actual release was made, that is, from January 1, 2019 to April 18, 2019. These app-reviews were classified and their features were extracted. We also extracted Jira issue reports from October 1, 2018 to April 18, 2019. We further filtered the issue reports to keep only the issue reports that were created before the actual release. After collecting the data, we generated the plans using the best parameters determined in Section 5.3 and compare the plans. We describe the differences between the plans in the following section.

5.5.2. Results

The sentences of the extracted app-reviews were classified into 1 Bug Reports (B), 2 Feature Request (R), 6 Feature Evaluation (E), 14 Praises (P) and 0 Other sentences (O). Out of twenty-three (23) sentences, five (5) features were extracted using SAFE. In addition, we extracted 47 features from Jira. In total, we obtained a consolidated list of 52 features that were planned (Table 19). The generated plan using the GA as described above is listed in Table 20.

Table 18 shows the actual release plan that was extracted from the GitHub repository. In addition, an alternative plan was generated to see how it differs from the first plan that was generated. Table 21 shows the alternative plan.

When comparing the release plan generated by the GA model (Table 20) and the actual release (Table 18), features PAINTROID-34, PAINTROID-41, PAINTROID-63, PAINTROID-64, PAINTROID-68, PAINTROID-66 and PAINTROID-35 are included in both release plans, meaning that 87.5% of the features in the generated release plan were included in the actual plan. The GA model recommends postponing PAINTROID-67, however, this feature was included in the actual release. The GA model also recommends including R-E1649C, R-E7C15A, R-C9A310, R-3AD796 and R-E1649C, which were not included in the real release. These features were extracted from the app-review sentences and assigned to the first release. The model indicates that the stakeholders will gain a higher level of satisfaction if this features where included in the mobile plan and eventually implemented. The generated plan shows that the feature **background eraser** would have had the highest satisfaction if it was implemented, followed by **made transparent** and **eg layers**. It can be seen that the feature **rotating icons example shape tool** was easily assigned to the plan, possibly because it requires little effort. The total estimated effort of the first release is maximized (27 story points), while the total estimated efforts of the second (26 story points) and

[17] https://github.com/Catrobat/Paintroid/releases

third (26 story points) releases requires one (1) more story point to maximize work. If there were features with one (1) story point, the algorithm would have easily assigned it to releases 2 and 3.

Comparing the alternative plan (Table 21) and the first generated release plan (Table 20), shows a few differences. The first difference is that feature PAINTROID-64 and PAINTROID-63 is postponed in the alternative plan, while they were included in the first plan. The alternative plan also included a high effort feature (PAINTROID-15) in the release, this feature was postponed in the first plan. Finally, the effort required for the third release in the alternative plan is 25 story points, while the first plan required 26 story points.

This show that different plans can be generated from the GA model, even when run with the same parameters.

Table 19. Consolidated set of features from this experiment.

Feature Key	Feature f(i)	Effort(Story Points) t(i,2)	Stakeholder S (1), Value v(1,i)	Stakeholder S (1), Urgency u(1,i)	Stakeholder S (2), Value v(2,i)	Stakeholder S (2), Urgency u(2,i)
R-C9A310	background eraser	5	5	(5, 0, 0)	3	(3, 0, 0)
R-3AD796	eg layers	5	1	(5, 0, 0)	1	(3, 0, 0)
R-E1649C	made transparent	5	1	(5, 0, 0)	1	(3, 0, 0)
R-E7C15A	settings option	5	2	(1, 0, 0)	1	(1, 0, 0)
R-D1901F	lowers resolution	5	2	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-71	create block library	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-68	resize canvas tool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-67	hand tool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-66	rename crop transform tool	0	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-65	color picker fullscreen	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-64	color picker library	8	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-63	color input	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-62	text stroke option	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-53	remove camera option	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-52	improve permission reasoning texts	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-51	remove native code	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-47	merge jenkins	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-45	update strings crowdin	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-44	merge jenkins	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-43	remove options tools options	2	1	(1, 0, 0)	1	(1, 0, 0)

PAINTROID-42	remove long press tools	1	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-41	rotating icons example shape tool	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-40	shape feedback	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-39	menus slide buggy	8	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-37	proper icon tools screen	2	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-36	auto crop functionality feedback	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-35	color picker apply cancel buttons	3	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-34	text tool confirm	2	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-33	merge jenkins limit resources paintroid docker container	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-31	jenkins release pipeline	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-30	merge jenkins	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-29	jenkins avoid gradle clean	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-28	jenkins track statistics	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-27	refactor tools transformtool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-26	refactor tools cursor	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-25	refactor tools pipette	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-24	refactor tools filltool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-23	refactor tools importtool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-22	refactor tools texttool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-21	refactor tools stamptool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-20	refactor tools brush	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-19	refactor tools linetool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-18	refactor tools eraser	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-17	refactor tools shapetool	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-16	refactor tools basetool observable	13	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-15	refactor tools modification tools	20	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-14	refactor tools public static variables	13	1	(1, 0, 0)	1	(1, 0, 0)

PAINTROID-13	refactor tools separate application business logic	40	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-12	refactor tools public static tool state	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-11	revert texttool size options removal	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-10	handle android dependencies docker directly	5	1	(1, 0, 0)	1	(1, 0, 0)
PAINTROID-9	replace save icon fitting alternative	5	1	(1, 0, 0)	1	(1, 0, 0)

Table 18. Actual release plan from GitHub.

Key	Feature	Type	Effort
PAINTROID-68	Add a resize to canvas tool	Story	5
PAINTROID-67	Add a hand tool	Story	5
PAINTROID-66	Rename crop in transform tool	Story	0
PAINTROID-64	Publish the color picker as a library	Story	8
PAINTROID-63	Add HEX color input to the color picker	Story	3
PAINTROID-41	No Rotating Icons in for example shape tool	Story	3
PAINTROID-35	Add apply and cancel buttons to the color picker	Story	3
PAINTROID-34	Add a confirm button to the text tool	Story	2

Table 20. Optimal release plan solved using the GA model.

Release	WAS	Key	Feature	Effort (Story Point)
1	33.6	R-3AD796	eg layers	5
1	33.6	R-E1649C	made transparent	5
1	12.8	R-E7C15A	settings option	5
1	12.8	R-D1901F	lowers resolution	5
1	148.8	R-C9A310	background eraser	5
2	0	PAINTROID-22	refactor tools texttool	5
2	0	PAINTROID-47	merge jenkins	5
2	0	PAINTROID-26	refactor tools cursor	5
2	0	PAINTROID-18	refactor tools eraser	5
2	0	PAINTROID-63	color input	3
3	0	PAINTROID-27	refactor tools transformtool	5
3	0	PAINTROID-62	text stroke option	5
3	0	PAINTROID-68	resize canvas tool	5
3	0	PAINTROID-64	color picker library	8
4	8	PAINTROID-10	handle android dependencies docker directly	5

4	8	PAINTROID-29	jenkins avoid gradle clean	5
4	8	PAINTROID-33	merge jenkins limit resources paintroid docker container	5
4	8	PAINTROID-44	merge jenkins	5
1	8	PAINTROID-34	text tool confirm	2
4	8	PAINTROID-20	refactor tools brush	5
4	8	PAINTROID-17	refactor tools shapetool	5
4	8	PAINTROID-71	create block library	5
4	8	PAINTROID-21	refactor tools stamptool	5
4	8	PAINTROID-12	refactor tools public static tool state	5
4	8	PAINTROID-30	merge jenkins	5
4	8	PAINTROID-28	jenkins track statistics	5
4	8	PAINTROID-19	refactor tools linetool	5
4	8	PAINTROID-14	refactor tools public static variables	13
4	8	PAINTROID-13	refactor tools separate application business logic	40
4	8	PAINTROID-15	refactor tools modification tools	20
4	8	PAINTROID-39	menus slide buggy	8
4	8	PAINTROID-67	hand tool	5
2	0	PAINTROID-41	rotating icons example shape tool	3
4	8	PAINTROID-25	refactor tools pipette	5
4	8	PAINTROID-23	refactor tools importtool	5
4	8	PAINTROID-31	jenkins release pipeline	5
4	8	PAINTROID-65	color picker fullscreen	5
4	8	PAINTROID-24	refactor tools filltool	5
4	8	PAINTROID-16	refactor tools basetool observable	13
1	8	PAINTROID-66	rename crop transform tool	0
3	0	PAINTROID-35	color picker apply cancel buttons	3
	Effort R1: 27.0	Effort R2: 26.0	Effort R3: 26.0	F(x): 257.6

Table 21. Alternative optimal release plan solved using the GA model.

Release	WAS	Key	Feature	Effort (Story Point)
1	148.8	R-C9A310	background eraser	5
1	12.8	R-E7C15A	settings option	5
1	33.6	R-3AD796	eg layers	5
1	12.8	R-D1901F	lowers resolution	5
1	33.6	R-E1649C	made transparent	5

2	0	PAINTROID-44	merge jenkins	5
2	0	PAINTROID-35	color picker apply cancel buttons	3
2	0	PAINTROID-33	merge jenkins limit resources paintroid docker container	5
3	0	PAINTROID-15	refactor tools modification tools	20
2	0	PAINTROID-68	resize canvas tool	5
2	0	PAINTROID-20	refactor tools brush	5
1	8	PAINTROID-66	rename crop transform tool	0
3	0	PAINTROID-47	merge jenkins	5
4	8	PAINTROID-30	merge jenkins	5
4	8	PAINTROID-24	refactor tools filltool	5
4	8	PAINTROID-31	jenkins release pipeline	5
4	8	PAINTROID-25	refactor tools pipette	5
4	8	PAINTROID-14	refactor tools public static variables	13
4	8	PAINTROID-26	refactor tools cursor	5
4	8	PAINTROID-13	refactor tools separate application business logic	40
4	8	PAINTROID-17	refactor tools shapetool	5
2	0	PAINTROID-41	rotating icons example shape tool	3
4	8	PAINTROID-23	refactor tools importtool	5
4	8	PAINTROID-21	refactor tools stamptool	5
4	8	PAINTROID-63	color input	3
1	8	PAINTROID-34	text tool confirm	2
4	8	PAINTROID-67	hand tool	5
4	8	PAINTROID-28	jenkins track statistics	5
4	8	PAINTROID-19	refactor tools linetool	5
4	8	PAINTROID-12	refactor tools public static tool state	5
4	8	PAINTROID-22	refactor tools texttool	5
4	8	PAINTROID-62	text stroke option	5
4	8	PAINTROID-71	create block library	5
4	8	PAINTROID-16	refactor tools basetool observable	13
4	8	PAINTROID-39	menus slide buggy	8
4	8	PAINTROID-18	refactor tools eraser	5
4	8	PAINTROID-29	jenkins avoid gradle clean	5
4	8	PAINTROID-10	handle android dependencies docker directly	5
4	8	PAINTROID-65	color picker fullscreen	5
4	8	PAINTROID-64	color picker library	8
4	8	PAINTROID-27	refactor tools transformtool	5
	Effort R1: 27.0	Effort R2: 26.0	Effort R3: 25.0	F(x): 257.6

5.6. Qualitative Analysis

To better understand whether the proposed process is useful for the selected case study, we conducted a qualitative study based on interviews. In the following, we describe the methodology applied and the results obtained.

5.6.1. Methodology

We designed an interview aimed at determining the extent our approach is suitable in the context of the case study. In particular, we aim to get contextual information about the development team, qualitatively evaluate the release plans generated by the GA model (Table 20) and to analyse the differences in the plans, as perceived by the team. The interview consists of 25 open questions. The questions of the interview can be found in Appendix II. Next, we contacted the main developers via email on the 14th of July, 2020. As a result, the product owner of Paintroid, accepted to participate in the study. The product owner is in charge of the release planning of Paintroid and has been working on this project for the past 10 years. The interview was conducted remotely on the 17th of July, 2020. The interview last approximately 1,5 hours. The interview was recorded and transcribed and the important points extracted. A consent form was also signed by the interviewee.

5.6.2. Results

The first and second part of the interview asked about the roles of developers and the context in which Paintroid is developed. The interviewee noted the roles of the people involved in the release planning process, he describes, "… *I'm the product owner. Then there is a pro uh, project coordinator, which, who is a senior developer. And then there are so more senior developers in our team. We also have a specialist for a user experience and we also have us consultants, some, um, designers who helped us for, for icon design.*" Next questions about their release planning were asked. He states that "… the *whole process involves a lot of discussion*." He also highlights, "*it's always a team decision also about which tickets should go into the ready for development. But the final decision is with me*." During planning the team uses some "*online planning poker software to register the estimates*" for requirements. The interviewee states that "… *the main decision during the release planning process is the order of the stories, the priority*…"

Furthermore, regarding the software development methods and practices applied by the team, he expressed: "*We are using an Agile software development process with a test first approach, test driven development. We are using a Kanban system. Um, we are using many practices from one Agile approach, which is called extreme programming if you know it."*

The third part of the interview asks about the suitability of the approach for their organization. The participant said: "*Yeah. I mean, for me, the release plan is we don't have a real release plan. We*

only have a priority list. So for me, it's important to have this priority because it tells us what to implement next. Uh, but otherwise, yeah, sure. I mean, it's, it's very similar, I think. Yes."

The fourth part of the interview asked the participant to compare the release plan generated by the model (Table 20) with the actual plan extracted from GitHub (Table 18). The model excluded PAINTROID-67 (**hand tool**), but this feature was included in the actual release published on May 28, 2019. The interviewee discussed the reason why this feature was included: "… *I think this was because we did some usability studies with the users and we found that it's confusing for them, that we didn't have an explicit, explicit hand tool because this, like, for instance, moving around the screen or zooming in and out was possible and is still possible with any tool. But some people didn't understand that. And were looking for this hand tool that they were used from our apps and complaining that, you know, like complaining that they don't see it and they don't find it. And that's frustrating for them. So actually this was very simple because it's actually anyway, already in the app and it didn't cost a lot to include a specific, uh, this feature, um, to satisfy those users who were, uh, who, who were unsure about how to use the feature, even though it was already included, we wanted, wanted to make it more explicit based on the user studies that we did."* He noted that the decision was "… *based on, on user studies with kids*…"

On the other hand, the model recommended including five features extracted from the app-review sentences: **eg layers**, **made transparent**, **settings option**, **lowers resolution** and **background eraser**. These features were not included in the actual plan. Regarding the feature **eg layers**, the interview stated that it already existed even before that release, "… *it's possible that it was because it was at the top of the screen instead of like the audit tools at the bottom, it was maybe difficult to find. Okay. And this is one of the reasons why we moved it down now. So if you're compared to the interface from one week ago to the new interface from today, you will see that that the icon for the layers has moved down first. It was at the top. Now it's at the bottom. So now all those action things are at the bottom together to make them easier to discover."* For feature **made transparent**, the interviewee also said it "… *existed already and it hasn't changed in any way."* The interviewee highlights that for the **settings option** feature: "*… it concerns all those settings here, like the colour tolerance for the filling and so on now. So we did some changes here, like allowing to also enter a numerical value here. Since that could be one thing. So this is now something where you basically have this, these settings everywhere*" For the **lowers resolution** feature. The interviewee noted, "*… I fully understand this. ... let's say you make a picture with your camera and this has a very high resolution. ... as soon as you import it into pocket paint, we automatically reduced the resolution. And this was not very nice. And many users complained about it. ... . So this was exactly such a feedback, like you mentioned from the, from the play store and we reacted to it. ... I think with the resolution, it's a complex problem. Yeah. No easy solution, but we are working on it. And there are a number of news on new tickets, which relate to this, but it's not easy to change. We would have to, uh, re-implement a large number or, I mean, a major part, the code base to do this correctly*." Finally, for the **background eraser** feature, the interviewee expressed, "… *that's already existing, but I guess what the users really want is you take a picture and arbitrary picture*

with your foot, uh, with your camera. And now there's something in front of it, say a person for instance and you want to make the rest transparent."

Next, we asked questions regarding the effort required to implement features. For features that require high effort to implement, the interviewee stated, *"... during the planning, we must weigh the effort against the desirability. So if there's a lot of effort, like for instance, for the lowers resolution, uh, either we try to find some low hanging fruits that are easy to achieve and are not so expensive so that we can more realistically achieve them. Or we try to separate it into several, uh, tickets and solve them step by step.*" Tickets with very high effort are split up by the team, because they did not "… *want to solve it as one because it simply was too big*."

Again, we asked the participant about the features that made the most difference in our approach. In this regard, the interviewee expressed, "*… all [the phases] are important for instance, the only thing that for me really counts of course, is the delivery. So the final step, that's the only thing that really counts because everything else is a preparation as, as long as it's not delivered, it doesn't count. Alright. Uh, most thing is real until it's released to see. And I mean, from the outer parts, uh, of course the software development is a large part. So the software development itself is one of the major. I mean, that's the part where the most work goes into it, the phase three. So this is really important implementation that testing the, reviewing the verifying and validating of the features. Yes. And from the creativity point of view, uh, this phase one is also very interesting to get with phase two because here we have to decide what to do. Right. And that's something that is really important for the final result because we, we want to do the right thing. We don't want to waste the time. So we have to focus on the right thing. And that's both one and two. So I think all of those steps are important. We cannot skip any of them*." Finally, we asked the participant whether the generated alternative is ok or not, the interviewee stated: "*it's impossible to say for me now because, um, I don't see so much the difference*"

Finally, the interviewee also discussed other sources besides app-review, where they get features from. He said there is "*a Google mailing list. And on that mailing list, there's this thread about feature suggestions.*" The interviewee thinks our approach is suitable for the current Paintroid project: "*… yeah, sure. I mean, it's, it's very similar, I think. Yes*." On the other hand, the interviewee describes how happy the team is with some Jira features like workflow. He states that "*… this system works very well. Everyone is very happy about it because the workflow is quite simple but efficient*."

6. Discussion of Findings

In this section, we discuss all the results obtained from section 5 and answer the research questions (RQ1, RQ2 and RQ3).

RQ1: How can we combine information extracted from app-reviews and issue trackers to support the release planning of mobile apps?

From our release planning process for mobile apps (Fig. 3), it can be seen how our approach relies on issue reports and app-review sentences to derive some decision variables like priority, value and effort. Features were also extracted from the app-review sentences; thus, our approach generates release plans based on the user's need and the information provided by the developers. This is done automatically and required less effort from the development team. One drawback of the approach is that developers may not update feature priorities and effort on the issue tracker. In that case, we made assumptions (Table 5).

In our approach, we also determined that the M4 (bert-large-nli-stsb-mean-tokens + Cosine Similarity) model, is the best model to use in consolidating features extracted from issue reports and app-review sentences. The main advantage of M1, M2, M3 and M4 over M5 (Jaccard) is that they generate sentence embeddings, which gives a more accurate cosine similarity score (1). The result from the M5 model (Table 26) shows that for every threshold the accuracy, precision, recall and f1-score remains the same. This is because only three (3) Jaccard similarity scores of the feature pair are above 0.5 for the M5 model[18]. Thus, the value of the metrics will remain unchanged. This shows that M5 will perform poorly if, for instance, we had a balanced dataset. The high metrics value is because the majority of classes in our manually labelled dataset are "different". The best model was chosen based on the highest f1-score.

Lastly, we concluded that fittest selection and ordered crossover with a crossover rate of 0.5 and a mutation rate of 0.3 are the best parameters for the GA model. Also, partially matched and fittest selection with a crossover rate of 0.9 and a mutation rate of 0.3 has good performance. Umbarkar & Sheth (2015) show that partially marched crossover technique performs great for problems where the absolute position elements matters. These best parameters were chosen based on some metrics like the number of iterations taken to converge, processing time and fitness score (Fig 7 & Fig 8).

Thus, our release planning approach for mobile apps, shows how to utilise user information from the app store and issue reports from the issue tracker, to simplify release planning practices in the mobile context, will maximizing the satisfaction of the users and stakeholders.

[18] https://github.com/geegog/MobileReleasePlanner/tree/master/similarity/results

RQ2: What is the performance of the existing release planning models in the context of mobile applications?

Based on our quantitative analysis from the performance experiment, the GA model outperforms the LP model with a fitness score of 617.6 (Table 16) compared to 300.8 (Table 17). It took about 170 iterations (in 5.8 seconds) for the GA model to solve the release planning problem. The LP model generates the plan instantly. Because the GA model has the highest fitness score and our object is to maximize the satisfaction of all stakeholders, we choose the GA model. Also, the difference in processing time between both models is negligible.

The GA model also performs better than the LP model when we compared the generated plan with the actual release. Using the GA model, 87.5% of features in the generated release plan, was included in the actual plan, but in the case of the LP model (Table 27), only 62.5% of features generated was included in the actual release. The fittest of the LP and GA model is 145.6 and 256 respectively.

RQ3: To what extent is the proposed approach useful in practice as perceived by developers?

The answer to this research question is given by the interview we conducted. The reason why the GA model postpones some features may be related to limited resources or very low satisfaction from the viewpoint of all stakeholders. This is supported by the interview conducted where the team tries to include a feature that is more realistic to achieve. The GA model stochastically assigns features to releases to maximize each release's satisfaction. Some features with low satisfaction can be included in the release if the effort required is very small and no other feature with high satisfaction can be included. Also, the models will never assign features whose effort is greater than the maximum effort (3) allocated to a particular release. In that case, such a feature will always be postponed and never planned by the approach. Thus, it is important to split feature with large effort into smaller ones.

In the release plan generated by the GA model (Table 20), it seems that the features (**setting options** and **lowers resolution**) extracted from class Feature Request (R) appeared to be understandable by the interviewee. The other features like **eg layers** and **made transparent** belong to the class Feature Evaluation (E), that is why the interviewee said that the features already existed. Thus, more contexts are required to be able to establish the improvement the user wants. Our approach provides a link to the actual app-review sentences, where the feature was extracted from.

The interviewee highlights that there is not much difference between the actual plan (Table 18) and the plans generated from the GA model (Table 20 & Table 21). The reason could be because their release planning is based on some decision variables like the priority and effort of features, which our approach also relies on. The difference in the actual and generated plans could be as a

result of the team including other information sources like Google mailing list, physical observation of the user and conducting user studies. For example, the **hand tool** feature was included in the actual release based on user studies. Our approach relies on app store and Jira issue tracker for information to drive the release planning of mobile apps. Finally, the interviewee noted that all phases in the release planning approach are important and no phase can be skipped.

7. Limitations

The first limitation applies to the consolidation experiment (Section 5.2.), as the result may not be generalized. We only used one open-source mobile project that uses Jira to track issue reports. This is because only one project met our criteria.
Another limitation is in terms of the reliability of the generated results from both the LP and GA model. Although our approach is mostly automated, most of the features priorities and effort values were assigned based on assumptions (Table 5), thus affecting the reliability of the results. The effort played a huge role in feature assignment to releases, since the actual value could be extracted from the issue reports. A more reliable result could have been obtained if we directly got feature priority values from the issue reports. Some of this information is not present in the data.

8. Conclusion

In conclusion, release planning can be a challenging and complex activity (Ruhe, 2010). Our release planning approach for mobile application automatically helps development teams to plan their releases by generating the best plan that supports the stakeholders' needs. The model automates data collection from the app store and issue tracker in the release planning of mobile apps. It further extracts features from app-review sentences keeping stories from the Jira source as it is. Thus, ensuring information entered by the development team is not modified or falsified. The model also automates (to some extent) the derivation of decision variable (like priority, value and effort) by combining both the app store and issue tracker information. Our approach maximizes the satisfaction of the stakeholders (users and the development team), by weighing their priorities for all features being planned (Saliu & Ruhe, 2005). The LP model is deterministic, meaning if the same decision variables are given as input to the model, the result will always be the same even if it is run 10 different times. On the other hand, the GA generates different solutions even with the same parameters. It is important to note that even though the solution changes, they are still optimal or near-optimal (Greer & Ruhe, 2003).

In this study, we explored how the release planning of mobile apps can be automated, by including app-review and issue tracker information. We conducted five experiments to demonstrate the applicability of our release planning approach in mobile development (Fig. 3). We answered quantitatively how we can rely on app-review and issue tracker information in the release planning of mobile apps. We also answered quantitatively what is the best hyper-parameters for the GA model. We quantitatively and qualitatively answered how similar the generated plans are to the actual release. We compared both release plans generated from the LP and GA model. Finally, we conducted an interview to qualitatively evaluate the release planning approach for Paintroid app.

Our main contributions in this study include: (1) Combining app-review and issue information to drive mobile releasing planning. (2) An approach to consolidate features from both data sources while automatically deriving some decision variables like priority, value and effort. (3) GA experiment using additional selection and crossover type. (4) A detailed description of our release planning process (Fig. 3). (5) Implementation of the approach[19].

We believe that our findings will help support the release planning of mobile apps and offer a structured process for obtaining release plans that increases the satisfaction of stakeholders involved in the planning process.

[19] https://github.com/geegog/MobileReleasePlanner

References

Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. *Information and Software Technology*, 43(14), 883-890. https://doi.org/10.1016/S0950-5849(01)00194-X

Bourque, P., & Fairley, R. E. (Eds.). (2014). Guide to the Software Engineering Body of Knowledge, version 3.0. *IEEE Computer Society*. www.swebok.org.

Chen, N., Lin, J., Hoi, S. C. H., Xiao, X., & Zhang, B. (2014). AR-miner: mining informative reviews for developers from mobile app marketplace. *Proceedings of the 36th International Conference on Software Engineering*, 767-778. https://doi.org/10.1145/2568225.2568263

Ciurumelea, A., Schaufelbuhl, A., Panichella, S., & Gall, H. C. (2017). Analyzing reviews and code of mobile apps for better release planning. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 91-102. https://doi.org/10.1109/SANER.2017.7884612

Denne, M., & Cleland-Huang J. (2004). The incremental funding method: Data-driven software development. *IEEE Software*, 21(3), 39-47. https://doi.org/10.1109/MS.2004.1293071

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171-4186 http://dx.doi.org/10.18653/v1/N19-1423

Ebert, C. (2014). Software Product Management. *IEEE Software*, 31(3), 21-24. https://doi.org/10.1109/MS.2014.72

Elbeltagia E., Hegazyb T., & Griersonb D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, 19(1), 43-53. https://doi.org/10.1016/j.aei.2005.01.004

Fhang, M. C. S., & Swamy, R. (2018). Best Practices in Release Management of Large Projects. *Proceedings of Proceedings of the 2018 7th International Conference on Software and Computer Applications*, 51-55. https://doi.org/10.1145/3185089.3185112

Franch, X., & Ruhe, G. (2016). Software Release Planning. *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 894-895. https://doi.org/10.1145/2889160.2891051

Greer, D. S. (2004). Decision Support for Planning Software Evolution with Risk Management.

Greer, D., & Ruhe, G. (2003). Software release planning: An evolutionary and iterative approach. *Information and Software Technology*, 46(4). 243-253. https://doi.org/10.1016/j.infsof.2003.07.002

Johann, T., Stanik, C., Alizadeh, B. A. M., & Maalej, W. (2017). SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 21-30. https://doi.org/10.1109/RE.2017.71

Jung, H. (1998). Optimizing value and cost in requirements analysis. *IEEE Software*, 15(4), 74-78. https://doi.org/10.1109/52.687950

Karlsson, J., & Ryan K. (1997). A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5), 67-74. https://doi.org/10.1109/52.605933

Kitchenham, B. (2004). Procedures for performing systematic reviews. Software Engineering Group, Department of Computer Science, Keele University.

Lak, P., & Turetken, O. (2014). Star Ratings versus Sentiment Analysis -- A Comparison of Explicit and Implicit Measures of Opinions. *2014 47th Hawaii International Conference on System Science*, 796-805. https://doi.org/10.1109/HICSS.2014.106

Lewis, C. (2008). Linear programming: Theory and applications. *Whitman College*.

Meng, X., & Wang, H. (2009). Mining User Reviews: From Specification to Summarization. *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, 177-180. https://dl.acm.org/citation.cfm?id=1667637

Nayebi, M., & Ruhe, G. (2017). Optimized Functionality for Super Mobile Apps. *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 388-393. https://doi.org/10.1109/RE.2017.72

Nayebi, M., Adams, B., & Ruhe, G. (2016a). Release Practices for Mobile Apps -- What do Users and Developers Think? *In 2016 IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 552-562. https://doi.org/10.1109/SANER.2016.116

Nayebi, M., Farahi, H., & Ruhe, G. (2016b). Analysis of Marketed versus Not-Marketed Mobile App Releases. *Proceedings of the 4th International Workshop on Release Engineering*, 1-4. https://doi.org/10.1145/2993274.2993281

Penny, D. A. (2002). An estimation-based management framework for enhancive maintenance in commercial software products. *International Conference on Software Maintenance, 2002. Proceedings*, 122-130. https://doi.org/10.1109/ICSM.2002.1167759

Perhinschi, M. G. (2017). An introductory Course on Computational Artificial Intelligence Techniques for Engineering Students. *Computers in Education Journal*, 8(3), 1-9. https://www.asee.org/documents/papers-and-publications/papers/CoEd_Journal-2017/Jul-

Sep/PERHINSCHI_INTRODUCTORY.pdf

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://arxiv.org/abs/1908.10084

Rossi, C., Shibley, E., Su, S., Beck, K., Savor, T., & Stumm, M. (2016). Continuous Deployment of Mobile Software at Facebook (Showcase). *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 12-23. https://doi.org/10.1145/2950290.2994157

Ruhe, G. (2010). *Product release planning: Methods, tools and applications*. CRC Press.

Ruhe, G., & Ngo-The, A. (2004). Hybrid Intelligence in Software Release Planning. *International Journal of Hybrid Intelligent Systems*, 1(2), 99-110. https://dl.acm.org/doi/10.5555/1232788.1232800

Ruhe, G., & Saliu, M. O. (2005). The art and science of software release planning. *IEEE Software*, 22(6), 47-53. https://doi.org/10.1109/MS.2005.164

Saliu, O., & Ruhe, G. (2005). Supporting Software Release Planning Decisions for Evolving Systems. *29th Annual IEEE/NASA Software Engineering Workshop*, 14-26. https://doi.org/10.1109/SEW.2005.42

Scalabrino, S., Bavota, G., Russo, B., Penta, D. M., & Oliveto, R. (2019). Listening to the Crowd for the Release Planning of Mobile Apps. *IEEE Transactions on Software Engineering*, 45(1), 68-86. https://doi.org/10.1109/TSE.2017.2759112

Shah, F. A. (2019). Extracting Information from App Reviews to Facilitate Software Development Activities (Doctoral dissertation). https://dspace.ut.ee/bitstream/handle/10062/66904/shah_faiz_ali.pdf?sequence=1&isAllowed=y

Shah, F. A., Sirts K., & Pfahl D. (2018). Simple App Review Classification with only Lexical Features. *In Proceedings of the 13th International Conference on Software Technologies*, 1, 112-119. https://doi.org/10.5220/0006855901460153

Shah, F. A., Sirts K., & Pfahl D. (2019). Is the SAFE Approach too Simple for App Feature Extraction? A Replication Study. *In Requirements Engineering: Foundation for Software Quality*, 11412, 21-36. https://doi.org/10.1007/978-3-030-15538-4_2

Slany, W. (2012). A mobile visual programming system for Android smartphones and tablets. *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Innsbruck*, 265-266. https://doi.org/10.1109/VLHCC.2012.6344546

Umbarkar, A. J., & Sheth, P. D. (2015). Crossover Operators in Genetic Algorithms: A Review. *ICTACT Journal on Soft Computing*, 6(1), 1083-1092. http://doi.org/10.21917/ijsc.2015.0150

Villarroel, L., Bavota, G., Russo, B., Oliveto, R., & Penta, D. M. (2016). Release Planning of Mobile Apps Based on User Reviews. *Proceedings of the 38th International Conference on Software Engineering*, 14-24. https://doi.org/10.1145/2884781.2884818

Appendix

I. Results from Consolidation Experiment

Table 22. Similarity check experiment result from M1 - bert-base-nli-mean-tokens + Cosine Similarity model.

	Threshold	Accuracy	Precision	Recall	f1 Score
0	0.5	0.570741	0.999674	0.569411	0.725551
1	0.51	0.598704	0.999689	0.597473	0.747935
2	0.52	0.628889	0.999704	0.627764	0.771233
3	0.53	0.657037	0.999717	0.656012	0.79219
4	0.54	0.690741	0.999462	0.69002	0.816403
5	0.55	0.72	0.999484	0.719383	0.836611
6	0.56	0.747593	0.999503	0.747073	0.855046
7	0.57	0.775926	0.999521	0.775506	0.873378
8	0.58	0.799444	0.999535	0.799108	0.888154
9	0.59	0.820926	0.999547	0.820665	0.901316
10	0.6	0.845926	0.999561	0.845754	0.916247
11	0.61	0.867963	0.999145	0.86824	0.929104
12	0.62	0.884815	0.999161	0.885151	0.938707
13	0.63	0.902037	0.999177	0.902434	0.948345
14	0.64	0.917593	0.998787	0.918417	0.956917
15	0.65	0.932222	0.998608	0.933284	0.964841
16	0.66	0.943704	0.998429	0.944992	0.970976
17	0.67	0.954074	0.998446	0.955399	0.976448
18	0.68	0.962222	0.998459	0.963576	0.980707
19	0.69	0.96963	0.998471	0.971009	0.984549
20	0.7	0.97537	0.99848	0.97677	0.987506
21	0.71	0.981111	0.998301	0.982717	0.990448
22	0.72	0.986111	0.99831	0.987735	0.992994
23	0.73	0.988519	0.998314	0.990151	0.994215
24	0.74	0.99037	0.99813	0.992195	0.995154
25	0.75	0.992222	0.998134	0.994053	0.996089
26	0.76	0.993704	0.998137	0.99554	0.996837
27	0.77	0.994074	0.997767	0.996283	0.997024
28	0.78	0.994815	0.997768	0.997027	0.997397
29	0.79	0.995926	0.997586	0.998327	0.997957
30	0.8	0.996296	0.997587	0.998699	0.998143
31	0.81	0.996667	0.997588	0.999071	0.998329
32	0.82	0.996667	0.997588	0.999071	0.998329

33	0.83	0.996852	0.997404	0.999442	0.998422
34	0.84	0.996667	0.997219	0.999442	0.998329
35	**0.85**	**0.997222**	**0.99722**	**1**	**0.998608**
36	0.86	0.997037	0.997035	1	0.998515
37	0.87	0.997037	0.997035	1	0.998515
38	0.88	0.997037	0.997035	1	0.998515
39	0.89	0.997037	0.997035	1	0.998515
40	0.9	0.996852	0.996851	1	0.998423
41	0.91	0.996852	0.996851	1	0.998423
42	0.92	0.996852	0.996851	1	0.998423
43	0.93	0.996667	0.996666	1	0.99833
44	0.94	0.996667	0.996666	1	0.99833
45	0.95	0.996667	0.996666	1	0.99833

Table 23. Similarity check experiment result from M3 - bert-base-nli-stsb-mean-tokens + Cosine Similarity model.

	Threshold	Accuracy	Precision	Recall	f1 Score
0	0.5	0.98	0.999432	0.980487	0.989869
1	0.51	0.982963	0.999433	0.98346	0.991383
2	0.52	0.984815	0.99887	0.985876	0.992331
3	0.53	0.987037	0.998873	0.988106	0.99346
4	0.54	0.98963	0.998876	0.990708	0.994775
5	0.55	0.990556	0.998877	0.991637	0.995244
6	0.56	0.991481	0.998505	0.992938	0.995714
7	0.57	0.993333	0.998508	0.994797	0.996649
8	0.58	0.994074	0.998509	0.99554	0.997022
9	0.59	0.994259	0.998138	0.996097	0.997117
10	0.6	0.995	0.998139	0.996841	0.99749
11	0.61	0.995	0.997954	0.997027	0.99749
12	0.62	0.995556	0.997955	0.997584	0.99777
13	0.63	0.996111	0.997956	0.998142	0.998049
14	0.64	0.996481	0.997957	0.998513	0.998235
15	0.65	0.996481	0.997957	0.998513	0.998235
16	0.66	0.997222	0.997958	0.999257	0.998607
17	0.67	0.997593	0.997959	0.999628	0.998793
18	0.68	0.997593	0.997959	0.999628	0.998793
19	0.69	0.997593	0.997774	0.999814	0.998793
20	0.7	0.997407	0.997589	0.999814	0.998701
21	0.71	0.997407	0.997589	0.999814	0.998701
22	0.72	0.997407	0.997589	0.999814	0.998701
23	**0.73**	**0.997593**	**0.99759**	**1**	**0.998794**
24	0.74	0.997593	0.99759	1	0.998794
25	0.75	0.997593	0.99759	1	0.998794

26	0.76	0.997593	0.99759	1	0.998794
27	0.77	0.997407	0.997405	1	0.998701
28	0.78	0.997407	0.997405	1	0.998701
29	0.79	0.997407	0.997405	1	0.998701
30	0.8	0.997222	0.99722	1	0.998608
31	0.81	0.997222	0.99722	1	0.998608
32	0.82	0.997037	0.997035	1	0.998515
33	0.83	0.997037	0.997035	1	0.998515
34	0.84	0.996852	0.996851	1	0.998423
35	0.85	0.996852	0.996851	1	0.998423
36	0.86	0.996667	0.996666	1	0.99833
37	0.87	0.996667	0.996666	1	0.99833
38	0.88	0.996667	0.996666	1	0.99833
39	0.89	0.996667	0.996666	1	0.99833
40	0.9	0.996667	0.996666	1	0.99833
41	0.91	0.996667	0.996666	1	0.99833
42	0.92	0.996667	0.996666	1	0.99833
43	0.93	0.996667	0.996666	1	0.99833
44	0.94	0.996667	0.996666	1	0.99833
45	0.95	0.996667	0.996666	1	0.99833

Table 24. Similarity check experiment result from M2 - bert-large-nli-mean-tokens + Cosine Similarity model.

	Threshold	**Accuracy**	**Precision**	**Recall**	**f1 Score**
0	0.5	0.635926	1	0.63464	0.776489
1	0.51	0.665741	1	0.66456	0.798482
2	0.52	0.690926	1	0.689835	0.816452
3	0.53	0.722222	1	0.721241	0.838048
4	0.54	0.747222	1	0.74633	0.854741
5	0.55	0.772593	0.999759	0.771975	0.871225
6	0.56	0.797593	0.999301	0.797435	0.887028
7	0.57	0.82037	0.999095	0.820479	0.90102
8	0.58	0.840741	0.999117	0.840922	0.913219
9	0.59	0.859815	0.998706	0.860435	0.924428
10	0.6	0.878333	0.998733	0.879019	0.93506
11	0.61	0.895185	0.998757	0.89593	0.944553
12	0.62	0.91037	0.998778	0.911169	0.952964
13	0.63	0.926852	0.9988	0.927709	0.961942
14	0.64	0.937778	0.998814	0.938673	0.96781
15	0.65	0.948889	0.998827	0.949823	0.973709
16	0.66	0.957407	0.998452	0.958744	0.978195
17	0.67	0.965	0.998464	0.966363	0.982151
18	0.68	0.972222	0.998475	0.973611	0.985886

19	0.69	0.975741	0.998292	0.977328	0.987698
20	0.7	0.979259	0.998298	0.980859	0.989501
21	0.71	0.98463	0.998307	0.986248	0.992241
22	0.72	0.988148	0.998313	0.989779	0.994028
23	0.73	0.990185	0.998316	0.991823	0.995059
24	0.74	0.991111	0.997946	0.993124	0.995529
25	0.75	0.992222	0.997948	0.994239	0.99609
26	0.76	0.993333	0.99795	0.995354	0.996651
27	0.77	0.995185	0.997954	0.997212	0.997583
28	0.78	0.995926	0.997956	0.997956	0.997956
29	0.79	0.996667	0.997772	0.998885	0.998328
30	0.8	0.997222	0.997774	0.999442	0.998607
31	0.81	0.997222	0.997774	0.999442	0.998607
32	**0.82**	**0.997593**	**0.997774**	**0.999814**	**0.998793**
33	0.83	0.997407	0.997405	1	0.998701
34	0.84	0.997222	0.99722	1	0.998608
35	0.85	0.997037	0.997035	1	0.998515
36	0.86	0.996852	0.996851	1	0.998423
37	0.87	0.996852	0.996851	1	0.998423
38	0.88	0.996852	0.996851	1	0.998423
39	0.89	0.996852	0.996851	1	0.998423
40	0.9	0.996852	0.996851	1	0.998423
41	0.91	0.996667	0.996666	1	0.99833
42	0.92	0.996667	0.996666	1	0.99833
43	0.93	0.996667	0.996666	1	0.99833
44	0.94	0.996667	0.996666	1	0.99833
45	0.95	0.996667	0.996666	1	0.99833

Table 25. Similarity check experiment result from M4 - bert-large-nli-stsb-mean-tokens + Cosine Similarity model.

	Threshold	**Accuracy**	**Precision**	**Recall**	**f1 Score**
0	0.5	0.981852	0.999244	0.982531	0.990817
1	0.51	0.985185	0.999247	0.985876	0.992516
2	0.52	0.987222	0.999248	0.98792	0.993552
3	0.53	0.988889	0.999249	0.989593	0.994398
4	0.54	0.990926	0.999251	0.991637	0.99543
5	0.55	0.991667	0.999065	0.992566	0.995805
6	0.56	0.992593	0.998879	0.993681	0.996274
7	0.57	0.993519	0.998694	0.994797	0.996741
8	0.58	0.994444	0.998695	0.995726	0.997208
9	0.59	0.995741	0.998697	0.997027	0.997861
10	0.6	0.995926	0.998512	0.997398	0.997955
11	0.61	0.996667	0.998513	0.998142	0.998327

12	0.62	0.997222	0.998514	0.998699	0.998606
13	0.63	0.997407	0.998329	0.999071	0.9987
14	0.64	0.997593	0.998329	0.999257	0.998793
15	0.65	0.997778	0.998329	0.999442	0.998886
16	0.66	0.997963	0.99833	0.999628	0.998979
17	0.67	0.998148	0.99833	0.999814	0.999071
18	0.68	0.998148	0.99833	0.999814	0.999071
19	0.69	0.997963	0.998145	0.999814	0.998979
20	**0.7**	**0.998148**	**0.998145**	**1**	**0.999072**
21	**0.71**	**0.998148**	**0.998145**	**1**	**0.999072**
22	0.72	0.997778	0.997775	1	0.998886
23	0.73	0.997593	0.99759	1	0.998794
24	0.74	0.997593	0.99759	1	0.998794
25	0.75	0.997593	0.99759	1	0.998794
26	0.76	0.997407	0.997405	1	0.998701
27	0.77	0.997222	0.99722	1	0.998608
28	0.78	0.997222	0.99722	1	0.998608
29	0.79	0.997222	0.99722	1	0.998608
30	0.8	0.997037	0.997035	1	0.998515
31	0.81	0.997037	0.997035	1	0.998515
32	0.82	0.996852	0.996851	1	0.998423
33	0.83	0.996852	0.996851	1	0.998423
34	0.84	0.996667	0.996666	1	0.99833
35	0.85	0.996667	0.996666	1	0.99833
36	0.86	0.996667	0.996666	1	0.99833
37	0.87	0.996667	0.996666	1	0.99833
38	0.88	0.996667	0.996666	1	0.99833
39	0.89	0.996667	0.996666	1	0.99833
40	0.9	0.996667	0.996666	1	0.99833
41	0.91	0.996667	0.996666	1	0.99833
42	0.92	0.996667	0.996666	1	0.99833
43	0.93	0.996667	0.996666	1	0.99833
44	0.94	0.996667	0.996666	1	0.99833
45	0.95	0.996667	0.996666	1	0.99833

Table 26. Similarity check experiment result from M5 - Jaccard Similarity model.

	Threshold	Accuracy	Precision	Recall	f1 Score
0	0.5	0.996667	0.99685	0.999814	0.99833
1	0.51	0.996667	0.996666	1	0.99833
2	0.52	0.996667	0.996666	1	0.99833
3	0.53	0.996667	0.996666	1	0.99833
4	0.54	0.996667	0.996666	1	0.99833
5	0.55	0.996667	0.996666	1	0.99833

6	0.56	0.996667	0.996666	1	0.99833
7	0.57	0.996667	0.996666	1	0.99833
8	0.58	0.996667	0.996666	1	0.99833
9	0.59	0.996667	0.996666	1	0.99833
10	0.6	0.996667	0.996666	1	0.99833
11	0.61	0.996667	0.996666	1	0.99833
12	0.62	0.996667	0.996666	1	0.99833
13	0.63	0.996667	0.996666	1	0.99833
14	0.64	0.996667	0.996666	1	0.99833
15	0.65	0.996667	0.996666	1	0.99833
16	0.66	0.996667	0.996666	1	0.99833
17	0.67	0.996667	0.996666	1	0.99833
18	0.68	0.996667	0.996666	1	0.99833
19	0.69	0.996667	0.996666	1	0.99833
20	0.7	0.996667	0.996666	1	0.99833
21	0.71	0.996667	0.996666	1	0.99833
22	0.72	0.996667	0.996666	1	0.99833
23	0.73	0.996667	0.996666	1	0.99833
24	0.74	0.996667	0.996666	1	0.99833
25	0.75	0.996667	0.996666	1	0.99833
26	0.76	0.996667	0.996666	1	0.99833
27	0.77	0.996667	0.996666	1	0.99833
28	0.78	0.996667	0.996666	1	0.99833
29	0.79	0.996667	0.996666	1	0.99833
30	0.8	0.996667	0.996666	1	0.99833
31	0.81	0.996667	0.996666	1	0.99833
32	0.82	0.996667	0.996666	1	0.99833
33	0.83	0.996667	0.996666	1	0.99833
34	0.84	0.996667	0.996666	1	0.99833
35	0.85	0.996667	0.996666	1	0.99833
36	0.86	0.996667	0.996666	1	0.99833
37	0.87	0.996667	0.996666	1	0.99833
38	0.88	0.996667	0.996666	1	0.99833
39	0.89	0.996667	0.996666	1	0.99833
40	0.9	0.996667	0.996666	1	0.99833
41	0.91	0.996667	0.996666	1	0.99833
42	0.92	0.996667	0.996666	1	0.99833
43	0.93	0.996667	0.996666	1	0.99833
44	0.94	0.996667	0.996666	1	0.99833
45	0.95	0.996667	0.996666	1	0.99833

II. Interview

Contact Email: contact@catrobat.org and support@catrobat.org

Invitation Letter:

An Invitation to participate in my research project

Dear,

Christian Schindler,

Matthias Müller,

Thomas Schwengler,

I am an MSc student at University of Tartu, Estonia and I am conducting a study on release planning of mobile open-source apps, where I aim to maximize the satisfaction of all important stakeholders such as users and developers by including app-review information.

As the team's project managers and active senior developer, you are in an ideal position to give valuable first-hand information of a project I included in my study, Paintroid. Would you like to participate in an interview as part of the study?

I will like to inform you that the interview will be conducted separately and takes about 45 minutes each if you don't mind. Some of the questions are about the actual release planning process you use. All the information collected will be used exclusively for research purposes and we can anonymize the data or participant's information if you request it. We have also prepared a consent form which explains how we will use the information from the interview.

Your participation in this research is really important for my study and I will appreciate your participation. Please in the event you are willing to participate, suggest a day and time that works best for you and I will make myself available.

If you have any questions, please do not hesitate to ask.

Thank you in advance,

Onuche Akor Idoko

Signed Interview Consent Form:

Interview Consent Form

Research Project Title: Automating the Release Planning of Mobile Apps by Including App-Reviews: A Case Study of Paintroid Mobile App.

Interviewer: Onuche Akor Idoko

Interviewee: Wolfgang Slany

The interview will take about 45 minutes. There is no risk in participating in this research, but you have the right to end the interview at any time.

Thank you for accepting to participate in this research. Ethically, the consent of interviewees is required, to show that they understand how the information in the research is used. Please read the following information and then sign at the end to show you approve of the interview:

1. We will record the interview and produce a transcript from the interview.
2. The transcript will be sent to you for review.
3. The transcript will be analyzed by Onuche Akor Idoko as the interviewer.
4. The access to the transcript will be limited to Onuche Akor Idoko, my supervisor, Ezequiel Scott.
5. A direct quotation from you during the interview, will be kept anonymous during the publication of this academic work, if you so wish.
6. The actual recording will be destroyed, after the transcript is generated and reviewed.
7. Any variation of the conditions above will only occur with your further explicit approval.

Interviewee's Signature _____ Date __16.7.2020_____

Interviewer's Signature: signed digitally Date: signed digitally

Interview Process and Questions:

1- Step 1: Get contextual information about the person.
   a. What's your current role?
   b. How long have you been working at Catrobat for the project Pocket Paint Android?
2- Step 2: Get contextual information about the current planning process
   a. How many developers are active?
   b. How do you plan the next release?
   c. What is the experience of the people involved during the planning?
   d. Do you include information of the final users during the planning process? Which type of info? Do you use app-review info?
   e. Do you use agile methods? Do you use any specific release planning practice (agile or not)?

    f. What other practices do you apply for software development?
    g. Do you somehow evaluate the release plans you create?
    h. What are the main decisions you have to make during the release planning process?
3- Step 3: Introduce the release planning approach shown in Fig. 3 (thesis document) and briefly explain it (explain the general goal). **Goal**: To automate (as much as possible) the release planning of mobile apps by including app-reviews to maximize the satisfaction of all stakeholders.
    a. How suitable is the approach (Fig. 3) for your organization? How good is it for the current software development process?
4- Step 4: Briefly describe the comparison between the best generated plan and the release plan published on GitHub. You can show the plan so the interviewee can remember it. Then, you ask:
    a. We can see that some features that are in the actual plan were postponed by our approach… could you tell us why you included them in the actual plan? Were there any issues with them during the development (regarding complexity, effort, or something else)?
    b. We can see that some features that are NOT in the actual plan were included by our approach, could you tell us why you postponed them?
    c. We also see that features with high effort are usually postponed by the algorithm. Would have you done the same? Would you include high-effort features in the next release?
    d. Which features in our release planning approach made the most difference?
    e. Do you think the generated plan would have been a better alternative or not? Why?
5- Step 5: Generate an alternative plan and ask the interviewee if it is also ok or not.
    a. The tool is currently a prototype but, would you use this tool if it is available?
    b. Would you recommend this in future projects?
    c. Finally, would you like to add something else?
6- Step 6: Follow-ups if necessary
7- Step 7: Thank the participants / closing remarks
8- Step 8: Transcribe the data
9- Step 9: Analyse findings / bring out results

III. Generated Release Plan from the LP Model

Table 27. Release plan from LP model.

Release	WAS	Key	Feature	Effort (Story Point)
1	33.6	R-E1649C	made transparent	5
1	8	PAINTROID-17	refactor tools shapetool	5
1	8	PAINTROID-30	merge jenkins	5
1	8	PAINTROID-41	rotating icons example shape tool	3
1	8	PAINTROID-71	create block library	5
1	8	PAINTROID-34	text tool confirm	2
2	8	PAINTROID-18	refactor tools eraser	5
2	8	PAINTROID-63	color input	3
4	8	PAINTROID-13	refactor tools separate application business logic	40
2	8	PAINTROID-29	jenkins avoid gradle clean	5
2	8	PAINTROID-44	merge jenkins	5
3	8	PAINTROID-15	refactor tools modification tools	20
1	8	PAINTROID-66	rename crop transform tool	0
2	8	PAINTROID-28	jenkins track statistics	5
3	8	PAINTROID-27	refactor tools transformtool	5
4	8	PAINTROID-25	refactor tools pipette	5
4	8	PAINTROID-12	refactor tools public static tool state	5
4	8	PAINTROID-19	refactor tools linetool	5
4	8	PAINTROID-23	refactor tools importtool	5
4	8	PAINTROID-21	refactor tools stamptool	5
4	148.8	R-C9A310	background eraser	5
4	12.8	R-D1901F	lowers resolution	5
4	8	PAINTROID-20	refactor tools brush	5
4	12.8	R-E7C15A	settings option	5
4	8	PAINTROID-65	color picker fullscreen	5
4	8	PAINTROID-67	hand tool	5
4	8	PAINTROID-39	menus slide buggy	8
2	8	PAINTROID-35	color picker apply cancel buttons	3
4	8	PAINTROID-68	resize canvas tool	5
4	8	PAINTROID-26	refactor tools cursor	5
4	8	PAINTROID-64	color picker library	8
4	8	PAINTROID-10	handle android dependencies docker directly	5

4	33.6	R-3AD796	eg layers	5
4	8	PAINTROID-16	refactor tools basetool observable	13
4	8	PAINTROID-33	merge jenkins limit resources paintroid docker container	5
4	8	PAINTROID-47	merge jenkins	5
4	8	PAINTROID-14	refactor tools public static variables	13
4	8	PAINTROID-31	jenkins release pipeline	5
4	8	PAINTROID-62	text stroke option	5
4	8	PAINTROID-24	refactor tools filltool	5
4	8	PAINTROID-22	refactor tools texttool	5
	Effort R1: 25.0	Effort R2: 26.0	Effort R3: 25.0	F(x): 145.6

IV. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Onuche Akor Idoko,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Automating the Release Planning of Mobile Apps by Including App-Reviews,

supervised by Ezequiel Scott.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Onuche Akor Idoko
10/08/2020