Optimized Functionality for Super Mobile Apps

Maleknaz Nayebi, Guenther Ruhe
SEDS Laboratory
University of Calgary
Calgary, Canada
Email: {mnayebi, ruhe}@ucalgary.ca

***Abstract*—Functionality of software products often does not match user needs and expectations. The closed set-up of systems and information is replaced by wide access to data of users and competitor products. This shift offers completely new opportunities to approach requirements elicitation and subsequent planning of software functionality. This is, in particular true for app store markets. App stores are markets for many small sized software products which provide an open platform for users to provide feedback on using apps. Moreover, the functionality and status of similar software products can be retrieved. While this is a competitive risk, it is at the same time an opportunity.**

In this paper, we envision a new release planning approach that leverages the new opportunities for decision making. We propose a new model using bi-criterion integer programming. We make suggestions for optimized super app functionality that are based on two key aspects: (i) the estimated value of features, and (ii) the cohesiveness between newly added features and cohesiveness between existing and the features to be added. The information on these attributes comes from reasoning on feature composition of existing similar apps. The approach is applicable to the development of new product releases as well as to the creation of completely new apps. We illustrate the applicability of our model by a small example and outline directions for future research.

***Keywords*-Release planning; app store mining; integer programming; new product design; super app design.**

I. Introduction

A paradigm shift in requirements engineering and software evolution towards data-driven processes in open environments was projected by Maalej et al. [12]. Ease of access to users feedback and usage data, the pro-active participation of users through social media, and the maturity of analytics tools and algorithms opened new opportunities for requirements engineering arise. This is particularly true for software app markets. Big data on similar products, together with access to customers' feedback, provide a unique way to develop new software products based on the success and failure of similar products. Our previous study with developers and users [15] showed that app developers look into their competitors to decide on their software releases and updates. All the available data openly accessible in the app stores can be used for planning apps' evolution.

Release management is a decision-centric process and part of incremental and iterative software evolution. In particular, *release planning* is the process of assigning features to upcoming releases (or iterations) such that the overall product evolution is optimized. Release planning approaches for answering *what* and *when* to release problems have been modeled and solved by different researchers and with different formulations, methods, and solutions. Aspects of release planning for mobile apps were studied in [25], [15], [14], [3], [17]. However, none of these studies provided an optimized solution approach with reference to data analytics for reuse of features and their cohesiveness.

In this paper, we outline the direction to compose "super" mobile apps by combining functionality available across existing products with similar purposes. Deciding on the functionality of a new product release is of fundamental importance for product success. The key dilemma is that often users do not know what they want or they changed their mind quite often and quite rapidly. In the case of widely accessible apps and their existing functionality, one way to approach this problem is to extract and reuse features that have proven successful and offer them in conjunction with the current features. We provide a new formulation for proposing optimized super app functionality. The formulation applies in the same way to the brown field and green field product planning of mobile apps:

Brown field design: Design of a new release for an existing app, and

Green field design: Design of a completely new product from scratch.

In both cases, the idea is to reuse existing functionality and to combine it in an innovative way. In particular, we answer the below research question:

> ***Research question:*** *How to determine value-cohesiveness-optimized app functionality by reuse of features from similar apps retrieved in the app store?*

The main contribution of the paper is a new method to find the best set of features for a new release of a mobile app using optimization techniques. We formulate this search for optimized apps as a bi-criterion optimization problem. Using quadratic integer programming [4], we search for the trade-off between the total value of a release and the total cohesiveness of the features selected for the new version of an app. We demonstrate these concepts and the related approach by an illustrative example.

II. Information Needs

Search for super app functionality is based on various sources of information. We describe them in the sequel.

This is authors version of the article, Nayebi, M., & Ruhe, G., "Optimized Design of Supper Apps", International Conference on requirements Engineering (RE), 2017.

A. Similar apps

App markets provide access to a pool of software products, some of them are designed for a similar purpose. We call them *similar apps*. This gives users the opportunity to use a similar product if they are dissatisfied with a particular app. The openness of these platforms on providing meta-data such as app descriptions, reviews, and app ratings makes it easier to perform a holistic search for finding similar apps.

When publishing an app in the market, the owners classify their apps in a specific category. Looking into these categories is one way to find similar apps. For example, the category "travel and local" includes a variety of apps for booking hotels, finding flights, restaurants, etc. The other way to identify similar apps is to search app stores with specific keywords pertaining to the desired purpose such as booking a hotel. To this end, one might search "booking AND hotel" in the app store.

B. Feature Extraction

Feature extraction of mobile apps was approached either by mining app descriptions or app reviews [13]. Analyzing app reviews to extract feature requests and bug reports were comprehensively approached by Maalej and Nabil [11].

Feature extraction from app descriptions has been studied in recent years [13], and a method was proposed by Harman et al. [8] to extract *featurelets* from app descriptions. The authors defined featurelets as "a set of commonly occurring co-located words using bi-grams and then aggregate similar featurelets into features using clustering [8]. For our proof-of-concept implementation, we used this method. However, alternative methods can be used [13]. Mining app descriptions and app store reviews do not provide a complete set of app features and looking into alternative information sources has been recommended [16].

C. Feature value

The question of what constitutes the value of a feature is difficult to answer. Value definition is context specific and user specific. A comprehensive software value-map is suggested by Khurum et al. [10]. Recent studies on mobile apps, used rating, the number of downloads, the content of reviews, and sentiment of reviews as the indicator of users' perceived value. In the illustrative example provided later in this paper, the value of app features is determined from crowdsourcing [12].

In the proof-of-concept example, by presenting the app features to Amazon Mechanical Turk workers (who are considered knowledgeable in using apps), we ask them to assign a value between 0 to 100 to each feature. The final feature value is the average of the values defined by all users. The value indicates how desirable the feature for a (potential) user is expected to be.

D. Cohesiveness between pairs of features

There are multiple technical and non-technical benefits from offering features in conjunction. While cohesiveness is difficult to measure, we propose a proxy measure combining feature co-occurrences and apps' rating. Two features $f(n)$, $f(m)$ are considered to be (functionally) connected if they often co-occurred within the set of existing apps, and even more so if this co-occurrence is happening for apps with high ratings.

In addition to the number of co-occurrences, we also consider the rating of apps. The app rating is provided by users in the app market. Rating is expressed on a five-point-scale ranging from 1 (lowest) to 5 (highest). The rating of an $app(k)$ is called $rating(k)$ which is the average of all customer ratings within the apps' life cycle. The product of average rating and number of co-occurrence is called *feature cohesiveness*. The cohesiveness between two features is the higher, the more often they co-occur among all the existing apps and the higher the average rating of these apps considered for co-occurrence. Formally, for any pair of features $f(n)$ and $f(m)$

- $\alpha(n, m)$ represents the number of co-occurrences of these features among all apps currently available in that category.
- $\beta(n, m)$ represents the average rating across all the apps offering $f(n)$ and $f(m)$ in conjunction.

For example, considering similar apps in the category of travel and local, the feature *search a location* and the feature *sort results base on distance from a location* often co-exist in these apps. However, the features *counting calories* and *searching a location* co-occurred only twice across all apps from the app store. We observed that two features *counting calories* and *searching a location* in the app category of *travel and local* co-occurred in two existing App 1 and App 2. App 1 with $rating(1) = 3.0$ and App 2 with $rating(2) = 3.5$. This means that $\alpha(n, m) = 2$, and $\beta(n, m) = 3.25$.

E. Effort estimation

Implementation of features consumes effort. We make the simplifying assumption of just looking at the total amount of (estimated) effort needed per feature. Different methods are applicable for providing estimates [23]. Often, effort estimation methods are hybrid in the sense that they combine formal techniques with the judgment and expertise of human domain experts. For our proof-of-concept example, we used three context expert app developers and each of them estimated the effort for implementing each feature in person hours.

III. Problem formulation

Inspired by state of the art release planning methods [19], we provide a novel formulation for proposing optimized super app functionality. The key idea is to look at two objectives and formulating the search for reuse-based app functionality as an optimization problem:

Objective 1: The total value of an app is defined as the sum of the features elicited from existing similar apps. This objective emphasizes the individual attractiveness of features for users which is the traditional linear value function used in release planning [21].

Objective 2: The degree of cohesiveness of features measured by their co-occurrence in existing apps. This objective is independent of the first one and looks at how often (quantity), and on which occasions pairs of features co-occurred in apps within similar context (quality).

The first optimization objective is purely value related. The actual value of new features is difficult to predict. In the case of brownfield development, one way to measure feature value is to look at its usage. Other alternatives were elaborated by Khurum et al. [10].

The second search objective emphasizes the cohesiveness of the features as observed from their former occurrence. Both objectives are independent and important. The projected value of features is what makes the new release valuable. The cohesiveness of features ensures that semantic connections between pairs of "successful" combinations of features should be maintained. This second objective is important as we are targeting semantically cohesive apps, not just a collection of individually high valued but unrelated features as studied for theme-based release planning [9].

Let $F1 = \{f(1)\dots f(N1)\}$ be a set of features implemented in the current product version called $version\ y.z$, and $F2 = \{f(N1+1)\dots f(N)\}$ the features extracted across a set of similar apps for being candidates for designing super app functionality of $Version\ y.z+1$. Each new app release $version\ y.z+1$ is characterized by an N-dimensional Boolean decision vector x with components $x(n)$ defined as:

$$x(n) = \begin{cases} x(n) = 1 \text{ if feature f(n) is offered, and} \\ x(n) = 0 \quad \text{otherwise} \end{cases} \tag{1}$$

By definition,

- each feature $f(n)$ from $F2$ has occurred in at least one of the existing apps, and
- $x(n) = 1$ for all $n = 1...N1$.

Each feature $f(n)$ has a perceived individual value called $value(n)$. Upcoming feature value is inherently difficult to predict. We consider crowdsourcing, user forums or stakeholder evaluation as possible options to estimate value. Our first objective is to maximize the total value of a release. It is expressed by value function $C1(x)$ and defined as:

$$C1(x) = \sum_{n=N1+1\dots N} value(n) \times x(n) \tag{2}$$

Each feature $f(n)$ has an estimated effort called $effort(n)$ needed for its implementation. With an assumed total effort capacity Cap available for implementation of version $y.z+1$, all new app releases encoded in vector x need to fulfill the effort constraint expressed as Equation (3):

$$\sum_{n=N1+1\dots N} effort(n) \times x(n) \leq Cap \tag{3}$$

This formulation so far looks like a traditional release planning problem [6] with just one release. Exclusively applying Equation (2) as an objective for optimization would result in a composition of features maximizing the total release value. What we potentially get is a collection of attractive features, but not necessarily a set of features that have been proven to be cohesive with regards to the experience.

To address this additional concern, we formulate our second objective $C2(x)$ with the target to maximize total cohesiveness of the features selected for the super app functionality. As outlined above, for all pairs $f(n)$, $f(m)$ of features we use the product of the number $\beta(n,m)$ of co-occurrences with its average rating $\alpha(n,m)$ as a proxy for their cohesiveness. In other words, the more often two features co-occurred, the higher their cohesiveness. Also, co-occurrence of two features in the apps with higher rating increases cohesiveness compared to apps with lower rating

The objective then is defined as Equation (4) with the summation taken over all pairs of features $f(n)$, $f(m)$:

$$C2(x) = \sum_{f(n),f(m)} \alpha(n,m) \times \beta(n,m) \times x(n) \times x(m) \tag{4}$$

Consideration of semantic cohesiveness between features was studied for the purposed theme-based release planning [9]). The novel part in our formulation is that for the selection of additional features, the cohesiveness between them, as well as the connectivity to existing features is taken into account. This is something that is ignored in all existing release planning approaches but should be considered important to ensure features' cohesiveness.

Both objectives $C1(x)$ and $C2(x)$ are independent and are competing. While we want to maximize the value, we also want to maximize the cohesiveness of the features. This is because best-valued apps are not automatically best in terms of cohesiveness and vice versa. In other words, we are looking for Pareto solutions for the bi-objective optimization problem to maximize $(C1(x), C2(x))$.

We investigate the question: *"For a given app release, which set of features should be added to increase the total value of the app and to maximize the cohesiveness between all pairs of features of the new release."*

All app releases fulfilling the effort constraint of Equation (3) are called *feasible app releases*. An app release x^* is a *Pareto solution* if no other feasible app release x' exists that is better in one criterion (value or cohesiveness) and at the same time not worse in the other. This is expressed in Equation (5):

$$\begin{aligned} &\text{(i)} \quad C1(x') \geq C1(x^*) \quad \text{and} \\ &\text{(ii)} \quad C2(x') \geq C2(x^*) \quad \text{and} \\ &\text{(iii)} \quad (C1(x'), C2(x')) \neq (C1(x^*), C2(x^*)) \end{aligned} \tag{5}$$

For simplicity, we do not look at additional (detailed effort or technology related) constraints. Note that this technically would not change the proposed solution approach as we keep value and cohesiveness as the optimization objectives.

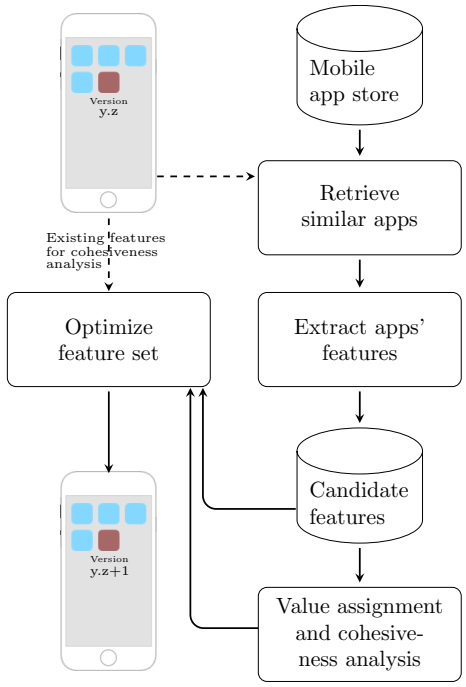

Fig. 1. Key steps of the process of finding optimized super app functionality. The two dashed arrows are only applicable for brown field development.

IV. Solution Approach

So far, we discussed possible methods for similarity analysis to find similar mobile apps as well as methods for extracting mobile app features, effort and value estimation in Section II and Section IV. Main steps of the solution approach are outlined in Figure 1.

For this paper, the emphasis is on the search process for finding an optimized set of feature for extending app functionality over releases. This problem formulation (Equations (1) to (5)) by nature is an Integer Quadratic Programming problem [7]. Each particular vector x with integer variables as defined in Equation (1) describes a specific composition of a new app release. The problem has a quadratic component as it contains combinations of two features as expressed in the second objective $C2(x)$.

To solve the problem, one can transforms the original bi-objective optimization problem into a sequence of single objective problems [4]. Following that, we have transformed our problem into a sequence of problems $G(\gamma)$:

$$\text{G}(\gamma, x) = Max\{\gamma \times C1(x) + (1-\gamma) \times C2(x) : \text{x fulfills (3)}\} \quad \text{for all } \gamma \text{ from (0,1)} \qquad (6)$$

The result of Equation (6) is a set of features for to be implemented in a new app release. From multi-criteria integer programming [4] it is known that the feature sets we receive from solving a series of the parametric single-criterion problems are Pareto optimal. For computing optimal (super) app releases for our proof-of-concept evaluation, we used an academic license of the proprietary Gurobi optimizer [1]. While the solution approach does not depend on this tool, Gurobi has proven to be both computationally efficient and scalable [1]. In a comparative analysis with top commercial solvers `CPLEX` (from IBM), `XPRESS` (from Fico) and also comparing to open source solvers such as `CBC`, `Gurobi` performed best concerning run time and the percentage of solved instances from a pool of benchmark problems[1].

V. Proof-of Concept Evaluation

We demonstrate the key idea of the approach by a small illustrative example which was taken from a more comprehensive case study formerly conducted [19]. The application context is *Over the Top TV (OTT)* apps (such as Netflix or Hulu) taken from the Android app store (Google Play). In this category, we gathered a set of 84 similar apps. Using the method proposed by Harman et al. [8], we extracted 10 features of OTT apps (which is a random subset of the total set of 34 features originally extracted). For all these features, the observed implementation effort and the feature value (based on crowdsourcing) are shown in Table I. We applied methods for crowdsourced feature evaluation and effort estimation as they were described in Section II. We investigate the question:

For a given OTT app and for given capacity to develop the next release, which features should be added to increase the total value of the app and to maximize the total feature cohesiveness?"

To illustrate our main idea, we assume that the first four features listed in Table I constitute the current version *Version 3.7* of the app under consideration, denoted by App^*. That means, App^* currently has features $f(1)$, $f(2)$, $f(3)$, $f(4)$. Different combinations of features $f(5)$, $f(6)$, ...,$f(10)$ are candidates for creating the next release *Version 3.8*.

Following the problem formulation in Section III to answer the above question we consider:

(i) The perceived individual value of each candidate feature $f(5)$, $f(6)$,...,$f(10)$, and

(ii) The degree of cohesiveness between all the pairs features. This includes the cohesiveness between existing features $f(1)$, $f(2)$, $f(3)$, $f(4)$ and newly added features as well as the cohesiveness of the newly added features with each other.

For (ii), the assumption is that features that have been occurred together in the past in (successful) competitor apps are good candidates to offer them in conjunction again. One reason might be that the two features depend on each other. Another one is that their co-occurrence creates value synergy. We provided the matrix of feature co-occurrence and average ratings as Table II. Therein, the values above the diagonal of the main matrix represent the average rating values ($\beta(n,m)$), the ones below the diagonal represent the number of co-occurrences ($\alpha(n,m)$). We observe that feature pairs $f(3)$,

[1] http://plato.asu.edu/ftp/milpc_tables/1thread.res

TABLE I

FEATURE SET OF THE ILLUSTRATIVE EXAMPLE FOR OPTIMIZED SUPER APP DESIGN AND ITS EVALUATION. FEATURES F(1), F(2), F(3) AND F(5) HAVE BEEN ALREADY IMPLEMENTED.

ID	Features	Value	Effort (in person hours)
f(1)	Live channel coverage	58.2	10.4
f(2)	Support of multi-screen	77.4	27.7
f(3)	Switching between horizontal and vertical views	49.1	10.3
f(4)	EPG	28.6	3.8
f(5)	Aspect ratio change	63.5	37.9
f(6)	Remote control	37.9	26.3
f(7)	Support for devices without a touch screen	43.8	20.4
f(8)	Video on demand	91.3	53.8
f(9)	YouTube integration	26.3	13
f(10)	Capability to select source signal	57.3	25.8

$f(8)$ and $f(5)$, $f(8)$ occur most frequently. At the same time, the pair $f(7)$, $f(8)$ co-occurs in one app with Rating 5.

Having feature value, cohesiveness and estimated implementation effort for features as discussed in Section II, we assumed $Cap = 58.3$ as the implementation effort budget for Version 3.8. Then, we applied formulation (6) and obtained four optimized alternatives for extending the current release Version 3.7 by adding features to create the new release Version 3.8:

Optimized Alternative O1: Adding f(9) and f(10) with (C1(x), C2(x)) = (296.9, 96.6)

Optimized Alternative O2: Adding f(7) and f(10) with (C1(x), C2(x)) = (314.4, 71.3)

Optimized Alternative O3: Adding f(4) and f(9) with (C1(x), C2(x)) = (303.1, 87.9)

Optimized Alternative O4: Adding f(4) and f(7) with (C1(x), C2(x)) = (320.6, 61.5)

All the four optimized feature sets are best in combining both value and cohesiveness for a new app release. That means, there is no alternative being better in one aspect and not worse in the other.

To visualize this, we formed all the possible feature combinations of $f(5)$, $f(6)$, $f(7)$, $f(8)$, $f(9)$, and $f(10)$ which needs implementation effort less than the assumed capacity of 58.3. We plotted the value and cohesiveness of these combinations along with the four optimized alternative solutions in Figure 2. In Figure 2, Optimized Alternative 1 to 4 are denoted by $O1$ to $O4$. $A1$, $A2$, $A3$ and $A4$ are non-optimal alternatives of feature combinations with required effort less than the available capacity ($= 58.3$). $A1$ is composed of $\{f(7), f(9)\}$, $A2$ is $\{f(6), f(9)\}$, $A3$ composed of $\{f(6), f(7)\}$, and $A4$ is composed of $\{f(6), f(10)\}$.

VI. RELATED WORK

Sophisticated analysis on the mobile apps, including reviews, ratings, the number of downloads, and dynamic changes in the competitors' apps provides opportunities for proactive analysis and decision making within this context [20], [13]. The results of different studies showed that the user's feedback

TABLE II

COHESIVENESS $\beta(n, m)$ (BELOW THE DIAGONAL) AND AVERAGE RATING $\alpha(n, m)$ (ABOVE THE DIAGONAL) FOR ALL PAIRS OF THE 10 FEATURES.

Features	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	f(7)	f(8)	f(9)	f(10)
f(1)		0	4.9	3	0	0	0	3.4	0	0
f(2)	0		3.2	0	4.6	3.8	0	4.9	0	0
f(3)	2	4		3.4	4.1	0	4.7	3.1	5	4.9
f(4)	2	0	8		4.5	4.6	0	4.5	3.3	0
f(5)	0	4	5	3		0	4.9	3.4	4.1	0
f(6)	0	1	0	2	0		0	0	0	0
f(7)	0	0	1	0	2	0		5	4.6	4.9
f(8)	1	7	14	2	15	0	1		4	4
f(9)	0	0	2	8	3	0	2	2		3.9
f(10)	0	0	1	0	0	0	2	1	9	

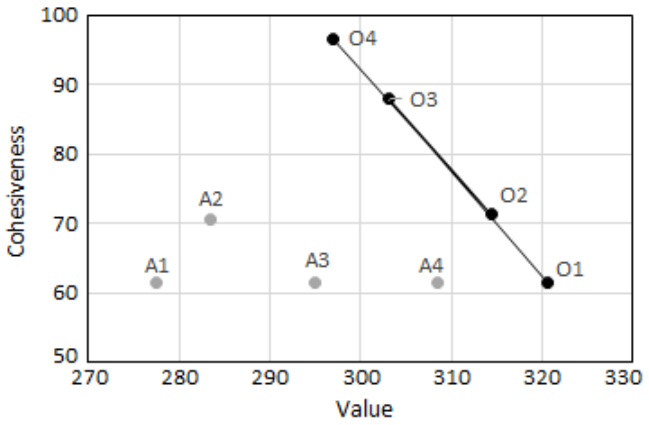

Fig. 2. Existing apps and optimized super app.

in the form of ratings and reviews have an impact on the app development decisions [20] and tools were investigated to assist app developers in reacting to their customers more efficiently [5]. App store mining and analysis has been widely studied in the context of software engineering from different aspects and reuse of code was discussed in number of studies [13]. However, requirement engineering and design of apps by reusing features from the other apps are rather untouched.

Seyeff et al. [22] studied the usage of Facebook to overcome current limitations of requirements' engineering (RE) tools in terms of the end user acceptance and involvement. They found that existing features of Facebook, e.g., the possibility to build dedicated groups, comment on posts, or like posts, support RE activities such as requirements elicitation, prioritization, and negotiation. We believe our proposed method, is the first step toward systematically integrating the voice-of-the customer into optimized mobile app design.

In a recent systematic literature survey on release planning models, Ameller et al. [2] confirmed the increasing trends to look at multiple objectives. In line with that, in the design of super app functionality, we looked into feature value and cohesiveness between features as the objectives. Release planning of mobile apps were studied [25], [15], [14], [3], [17]. However, none of these studies provide a formal formulation considering the established state of the art release planning or prioritization methods [21], [24] and often refer

to prioritization of bug fixes and features as release planning of mobile apps.

VII. Limitations

When looking at cohesiveness of feature combinations, we are currently limited to pairs of two features. However, additional synergies might occur for larger combinations (e.g., three or four features co-occurred often together). Also, detecting similar apps and their features is not an easy task and need further elaboration. Looking into the app store categories does not guarantee the similarity as the functionality range of apps in the same category is rather broad. However, searching for specific keywords, using domain experts, and snowballing the app store recommendations can be used in addition.

The scalability of the method needs to be proven. This is not only a computational but also an information retrieval and validity problem. Finding similar features and determining their value and cohesiveness is computationally intensive. We foresee the need for more comprehensive evaluation of the applicability of the proposed solution idea and have started applying the process to real-world app design [18].

VIII. Next RE Research

Offering the most attractive functionality to users is the ultimate goal of app release planning. We proposed a model to frame the functional design of apps as a systematic and optimization based decision-making process. So far, the limited proof-of-concept evaluation shows the applicability of our model to both new app development and extending existing apps with new functionality. The proposed model and its algorithmic implementation are flexible to accommodate more complex synergies and types of feature dependencies. With a powerful solver like Gurobi, solving quadratic integer programming is scalable and is expected to solve problems with several hundred of features.

Our initial idea was presented for brownfield app development considering an existing product in the market. However, the same idea can also be applied to enhance the functionality of a new app (Greenfield app design). The only difference is that the current set of existing features in the app is empty. We evaluate the above ideas trough fully-fledged real-word case studies as the next steps of this research.

Finally, the current model looks exclusively at features that have been occurred somewhere before. However, there might be new features coming from other information sources. As outlined and evaluated by Nayebi et al. [18], features can be extracted also from the mining of social media platforms such as Twitter. by adjusting the cohesiveness function $C2(x)$ to reflect "requirements co-occurrence" instead of features' co-occurrence, the formulation and solution approach can be applied in the same way for this more comprehensive application scenario.

Acknowledgments

This research was partially supported by Alberta Innovates Technology Futures AITF (first author) and by the Natural Sciences and Engineering Research Council of Canada, NSERC Discovery Grant 250343-12 (second author).

References

[1] Gurobi optimization. http://www.gurobi.com. Accessed: November 2016.

[2] D. Ameller, C. Farré, X. Franch, and G. Rufian. A survey on software release planning models. In *Proc. PROFES*. Springer LNCS, 2016.

[3] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. Gall. Analyzing reviews and code of mobile apps for better release planning.

[4] J. Clímaco, C. Ferreira, and M. E. Captivo. Multi-criteria integer programming: An overview of the different algorithmic approaches. In *Multicriteria analysis*, pages 248–258. Springer, 1997.

[5] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proc. FSE*, pages 499–510. ACM, 2016.

[6] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[7] R. Gupta and M. C. Puri. Bicriteria integer quadratic programming problems. *Journal of Interdisciplinary Mathematics*, 3(2-3):133–148, 2000.

[8] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: Msr for app stores. In *Proc. MSR 2012*, pages 108–111. IEEE Press, 2012.

[9] M. R. Karim and G. Ruhe. Bi-objective genetic search for release planning in support of themes. In *Proc. SSBSE*, pages 123–137. Springer, 2014.

[10] M. Khurum, T. Gorschek, and M. Wilson. The software value map—an exhaustive collection of value aspects for the development of software intensive products. *Journal of Software: Evolution and Process*, 25(7):711–741, 2013.

[11] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *Proc. RE2015*, pages 116–125. IEEE, 2015.

[12] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016.

[13] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering.

[14] S. McIlroy, N. Ali, and A. E. Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, 21(3):1346–1370, 2016.

[15] M. Nayebi, B. Adams, and G. Ruhe. Release practices for mobile apps–what do users and developers think? In *Proc. SANER*, volume 1, pages 552–562. IEEE, 2016.

[16] M. Nayebi, H. Cho, H. Farrahi, and G. Ruhe. App store mining is not enough. In *Proc. ICSE*. ACM, 2017.

[17] M. Nayebi, H. Farrahi, and G. Ruhe. Analysis of marketed versus not-marketed mobile app releases. In *Proc. RELENG*, pages 1–4. ACM, 2016.

[18] M. Nayebi, M. Marbouti, R. Quapp, F. Maurer, and G. Ruhe. Crowdsourced exploration of mobile app features: A case study of the fort mcmurray wildfire. In *Proc. ICSE Conference*, pages 552–562. IEEE, 2017.

[19] M. Nayebi and G. Ruhe. Analytical product release planning. In *The Art and Science of Analyzing Software Data*, pages 550–580. Morgan Kaufmann, 2015.

[20] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *Proc. ICSME*, pages 291–300. IEEE, 2015.

[21] G. Ruhe. *Product release planning: methods, tools and applications*. CRC Press, 2010.

[22] N. Seyff, I. Todoran, K. Caluser, L. Singer, and M. Glinz. Using popular social network sites to support requirements elicitation, prioritization and negotiation. *Journal of Internet Services and Applications*, 6(1):1, 2015.

[23] M. Shepperd. Software project economics: A roadmap.

[24] M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. B. Saleem, and M. U. Shafique. A systematic review on strategic release planning models. *Information and software technology*, 52(3):237–248, 2010.

[25] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *Proc. ICSE*, pages 14–24. ACM, 2016.