An Information Fusion based Evolution Requirements Acquisition Method for Mobile Applications

Yuanbang Li[1], Rong Peng[1]*, Bangchao Wang[1], Dong Sun[2]

School of Computer Science, Wuhan University, Wuhan, China

IT Management Department, Haitong Securities Co.Ltd, Shanghai, China

(lybang,rongpeng,wangbc)@whu.edu.cn, 13362999@qq.com

***Abstract*—User feedbacks and market changes are both important sources of requirements evolution. Accurately capturing the evolutionary demands from user feedbacks and market changes are extremely important for providers of mobile apps to adjust evolutionary strategies of products. However, many challenges, such as divergent demands and conflicting demands, hinder the process of evolutionary requirements acquisition from multiple sources. Thus, eliciting and merging information from multiple sources is vital to make intelligent evolution decisions. In this paper, an evolution requirements acquisition method based on information fusion is proposed, which comprehensively utilizes its functionality statements, its online comments and its similar apps' online comments to refine evolutionary requirements. By evolution point ranking and selection, it provides a feasible and reasonable way to recommend the evolutionary requirements for the next release of the mobile application.**

Keywords information fusion; evolutionary requirements; kernel concerns; mobile app; user comments.

I. INTRODUCTION

Since the launch of Apple App Store and Google Play in 2008, mobile applications have penetrated into multiple aspects of people's daily life such as communication, games, reading, shopping, social networking, scheduling, working and so on [1].

Users are increasingly interested in downloading and installing mobile applications (apps) to obtain convenient services as more and more apps are published in app stores. Meanwhile mobile apps are facing fierce market competition. On the one hand, the proliferation of homogenous apps brings great challenges to the sustainable development of mobile apps [2]; on the other hand, user expectations are becoming more and higher with the improvement of mobile apps [3]. If mobile apps fail to respond demand changes timely, they will be replaced by other apps soon. Therefore, continuously paying attention to user feedbacks, market changes and main competitors are vital for mobile apps to stand out in fierce competition.

However, tracing, integrating and prioritizing the demands elicited from user feedbacks, market changes and main competitors are always huge challenges for providers due to the cost and time constraints [4]. Hence, in this paper, an information fusion based evolution requirement acquisition method for mobile apps is proposed, which not only provides a merge algorithm to synthesize the information of its own feedback and similar apps' feedback, but also proposes a ranking method to evaluate the importance of the evolutionary points, which provides a feasible way to determine the priorities of the evolution requirements of certain mobile application.

In this paper, Section 2 introduces the relevant work. Section 3 to 5 introduces the whole method. Section 6 demonstrates the effectiveness of the method through a case study. Section 7 summarize the paper.

II. RELATED WORKS

There are two main ways for mobile applications to acquire evolutionary requirements: monitoring based evolutionary requirements acquisition and application market analyses based evolutionary requirements acquisition.

A. Monitoring Based Evolutionary Requirements Acquisition

Monitor the changes of user behaviors and system performance indicators, find problems or bottlenecks in time are important for mobile apps to analyze evolutionary requirements. Therefore, deploying various performance monitoring tools to detect performance indicators such as response time, resource utilization and data transmission rate in real-time has become important means to guide system improvement [6]. For example, Instagram deploys Munin to fulfil network resource monitoring, Dogslow to fulfil process monitoring, and Redis to fulfil database query traffic monitoring [7]. However, monitoring-based evolutionary requirements acquisition is prone to discovering system anomalies and performance bottlenecks, but not suitable for capturing evolutionary requirements arising from user experiences or expectation changes.

B. Application Market Analyses Based Evolutionary Requirements Acquisition

Basic information, technical information and market information of a mobile app are needed when it is submitted to app market to facilitate user retrieval. The basic information of the application mainly includes the developers, size, function description and characteristics description. The technical information includes function interfaces, class libraries and resource manifest files, which can be obtained by reverse analysis. The market information consists of the price, category, download records and reviews of the app.

The methods in this category can be divided into three kinds: feature analysis, version engineering and commentary analysis.

Feature analysis methods mainly focus on extracting applied features from all the available information sources include app descriptions and resource listing files, internal functions, permission and comments by NLP, topic modeling , clustering and other technologies[8-10]. The

DOI reference number: 10.18293/SEKE2020-026

methods have been widely used in app recommendation and version evolution.

Version engineering methods focus on version information and release strategies. The recommendation of app and the formulation of version strategy are realized based on the analysis of the relationship between version external function interfaces, download volumes, comments and sales volumes [11-13].

Comment analysis methods focuses on extracting useful information from online comments of apps. These methods categorize and summarize the comments with other information such as version and download to understand the concerns and complaints of users by using the technology of classification, topic extraction, affective analysis, association mining and regression analysis [14-16].

III. Acquisition Process of Evolutional Requirements

Evolutional requirements can be extracted from user comments. However, effective comments from which evolutionary requirements can be extracted are few because they were written spontaneously by ordinary users with the main purpose of describing their own feelings. Therefore, it is not enough. The acquisition should be broaden to gather the useful information from the comments of its similar apps, namely the apps with similar functions and the apps from competitors or potential competitors in the market.

As shown in Fig.1, a mobile application evolutionary requirements acquisition process is designed to integrate user requirements from multiple sources, which include the application information, its user comments and the user comments of its similar apps.

Firstly, Kernel Concerns (KCs) are automated extracted from its own comments and its similar apps' comments; and then, they are used to generate a specific Scenario Model Instance (SMI) for each comment; after that, Aggregated Scenario Models (ASMs) are generated and merged according to the similarity of the kernel concerns of SIMs and ASMs; then, an association establishment algorithm is utilized to establish the associations between the ASMs and the Functional Structure Tree (FST) created according to the application information; finally, a ranking strategy is employed to prioritize the potential evolutionary points.

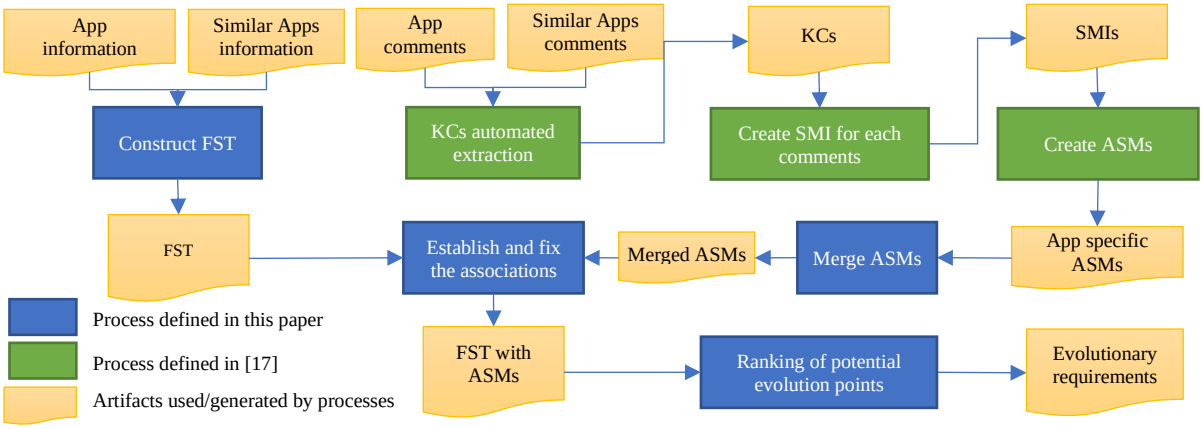

Fig. 1. Acquisition process of evolutional requirements

IV. Model Definition

A. Definition of Functional Structure Tree

The app information registered in App Store is represented in Functional Structure Tree (FST), which is defined as follows:

Definition 1 Functional Structure Tree is modeled as T= (N, R), where N = {N_0, N_1... N_n} denotes the set of function nodes in T (n >=0) and N_0 is the root node and represent the whole system; each functional node N_i contains two attributes: function name and function description, denoted as Ni:=<Name, Description>; R={<N_i, N_j>, $i \neq j$} represents the set of relationships among function nodes in T, and <N_i, N_j> indicates that N_j is a sub-function of N_i.

FST construction process is as following: firstly, construct the app's FST according to the basic and technical information provided to App Stores; after that, supplement the FST with the functions of its similar apps abiding by the following rules:

Traverse the function description of each similar app: for each function f_i in the description:

- If a matching node can be found in FST, record it as an alias if the function name is inconsistency with the node's name;
- Otherwise, if f_i is a sub-function of the function f_j which has a matching node N_j is FST, add a new node N_i for f_i and establish a relation < N_j, N_i >; otherwise, add a new node N_i for f_i and establish a relation < N_0, N_i >.

An example is shown as Fig. 4.

B. Definition of Scenario Model Instance

Definition 2 Scenario Model Instance (SMI) describes in which scenario the demand is needed or the defect happens. Its core element is KernelConcern, which has the attributes of HasTriggers, HasApperences, HasTerminal, HasOS and HasAppV [17].

For space limit, we only shown an example of SMI for the comment C_1 "Quit without prompt after open the positioning function" in Fig.2. The technical detail of how to construct SMI can be found in [17].

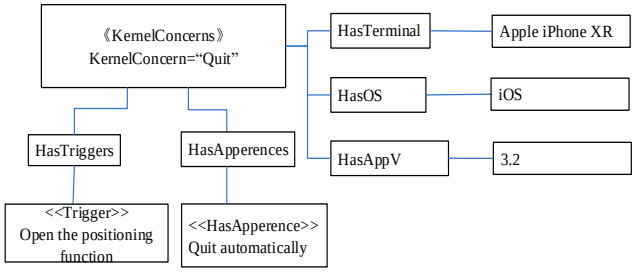

Fig. 2. SMI for the Comment C_1

C. Definition of Aggregated Scenario Model

Definition 3 Aggregated Scenario Model (ASM) is an aggregated model of multiple scenario model instances with the same kernel concerns.

The algorithm of ASM construction are also elaborated in [17]. Therefore, we only show an example of ASM with the kernel concern "Quit" in Fig.3. The model is integrated by 15 SMIs whose kernel concerns are all "Quit". And the number in each rectangle represents the frequency of the attribute appears. It is worth mentioning that the number of trigger is not 15 because some of the comments do not specify their trigger events.

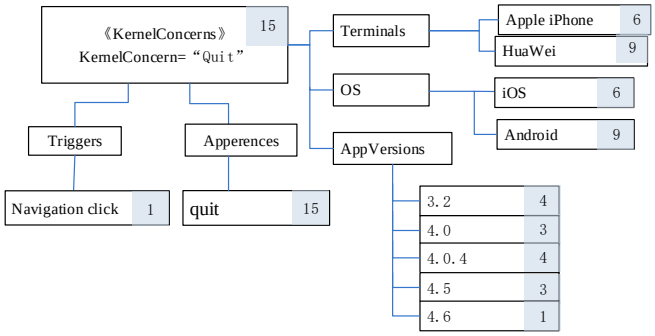

Fig. 3. Aggregated scenario model

V. Ranking and Selection of Potential Evolutionary Points

Ranking and selection of potential evolution points should be related not only to the importance of a function or the severity of a defect but also to the degree of the user attention to them. Therefore, the following evolution points

The step of ranking for defect feedback evolution point are as follows:

Step1: Construct FST for the app according to the process described in section 4.1.

Step2: Construct SMIs and ASMs for the app and its similar apps according to [17].

Step3: Merge the ASMs of all apps.

In this step, the ASMs of different apps are merged according to the similarity of their kernel concerns, which is judged by the requirements analyst. Once the analyst decides which two ASMs can be merged, the merge algorithm can be carried out automatically.

Algorithm 1: Merge algorithm of ASMs
Input: ASM M_1, M_2; M_1, M_2 are the ASMs to be merged
Output: ASM M
1: Initialize ASM M = M_1
2: Initialize the counter of the kernel concern of M:
M.times += M_2.times
3: for each a∈AttributesSet do
4: for each v∈M_2.a.V do
5: if (v∈M.a.V) then
6: M.a.v.times += M_2.s.v.times
7: else
8: M.a.V.add(v)
9: M.a.v.times= M_2.s.v.times
10: end if
11: end for
12: end for
13: return M

The input of the algorithm is two ASMs, M_1 and M_2, and the output is the merged M. M is initialize to M_1 (Line 1), and the counter of the KernelConcern of M is set to the sum of the counters of the KernelConcern of M_1 and M_2 (Line 2). For each attribute *a* in AttributesSet (Line 3), traverse each value v in M_2.s.V(Line 4): if v already exists in M.s.V, sum M.s.v.times and M_2.s.v.times (Line 5-6); otherwise, add v to M.s.V, and assign M.s.v.times to M_2.s.v.times (Line 7-9).

For example, as shown in Fig. 3,{Triggers, Appearances, Terminals, OS, AppVersions} are the AttributesSet of the ASMs; and the value set of the attribute "Terminals" is {"Apple iPhone", "HuaWei"}.

Step4: Establish the association between the ASMs and the FST according to the following algorithm:

Algorithm 2: Association establishment algorithm between ASMs and FST
Input: ASMSet MS, FST T;
Output: RelationSet RS
1: RS =NULL // Initialize the relation set RS to NULL
2: for each *asm*∈MS do
3: for each *lnode*∈T.leafNodeSet do
4: if (*asm*.KernelConcern.IsAssociatedWith(*lnode*))
5: then RS.add(*asm*.KernelConcern, *lnode*)
6: end if
7: end for
8: end for
9: return RS

Algorithm 2 aims to establish the association between ASMs and FST. For each *asm* in the ASM set MS, traverse each leaf node *lnode* of FST T (Line 2-3): if the kernel concern of *asm* is associated with *lnode* which is determined by analyst, add a relation between them to the relation set RS (Line 4-5). Finally, the relation set RS is returned (Line 9).

Step5: Ranking of the ASMs

The importance of an ASM_i $G(ASM_i)$ is measured by the product of the importance of the model's kernel concern $GA(ASM_i)$, the importance of the function associated with the model $Gf(ASM_i)$ and the user attention to the model $Gu(ASM_i)$, as shown in formula 1.

$$G(ASM_i) = GA(ASM_i) \cdot GF(ASM_i) \cdot GU(ASM_i) \quad (1)$$

$GA(ASM_i)$ is determined by requirement engineers and can be divided into three levels{0.5,1,2}, which represent{not serious, normal, serious}. $GF(ASM_i)$ is also divided into three levels{0.5,1,2}, which represents {unimportant, normal, important}. The value of $GF(ASM_i)$ that is not associated with any functional of the ASM is set to 1 by default. Of course, requirements engineers can also set other levels in the specific implementation process.

$GU(ASM_i)$ can also be divided into three levels{0.5,1,2}. It is calculated by the number of times the user pays attention to the concern, as shown in formulas 2- 4.

$$GU(ASM_i) = \begin{cases} 0.5 \ if(T_i < minp) \\ 1 \ if(T_i \geq minp \&\& T_i \leq maxp) \\ 2 \ if(T_i > maxp) \end{cases} \quad (2)$$

$$minp = min + (max - min)/3 \quad (3)$$

$$maxp = min + 2 \times (max - min)/3 \quad (4)$$

Where T_i indicates the number of times ASM_i is concerned; max and min indicates the maximum and minimum number of times a ASM is concerned, respectively; minp and maxp divide the range between min and max into three ranges on average.

Based on the score calculated of $G(ASM_i)$, the ASMs with TOP n scores are recommended as the evolution requirement points.

VI. CASE STUDY

Take Baidu Map with the version of 4.6 as the sample app, and take Amap with the version of 5.0 as its similar app. 840 user comments of Baidu Map and 960 user comments of Amap are crawled as feedbacks.

The evolution points are selected according to the process described in section V, which is specified as follows.

Step 1: The FST of Baidu Map was built as Fig.4.

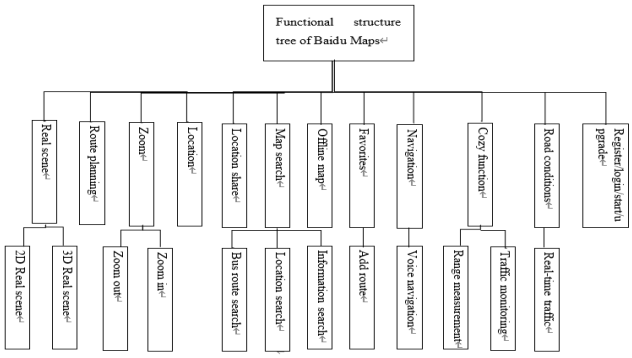

Fig. 4. FST of Baidu Maps

Step 2: Seven feedback ASMs are constructed through the analysis of the comments, namely "deviation", "inaccurate", "slow", "auto-exit, "flash-screen", "down-time" and "waste-of-slow".

Step 3: "Inaccurate" and "deviation" are merged as they indicate the same meaning. Finally, six ASMs are retained.

Step 4: The associations between these ASMs and the FST are established using algorithm 2. The associations between ASMs and function nodes of FST are built as follow:<deviation, location>, <slow, location>, <auto-exit, navigation>, <flash-screen. start>. No functions are associated to the ASM with concern "down-time" and "waste-of-flow".

Step 5: Set the GA, GF value of each ASM under the guidance of the analyst and calculate GU according to the formula 2-4 based on the times of the feedbacks. Finally, calculate the importance of each ASM by formula 1 and sort them in descending order. The result is shown in Table 1.

Table 1. The result of the ranking of evolution points

N	F	R	T	GU	GA	GF	$G(ASM_i)$
1	Deviation	Location	720	2	2	2	8
2	Slow	Location	490	1	1	2	2
3	Auto-exit	Navigation	150	0.5	2	2	2
4	Flash-screen	Start	190	0.5	2	2	2
5	Down-time	/	90	0.5	2	1	1
6	Network flow waste	/	160	0.5	1	1	0.5

Note: F indicates the kernel concern of the ASM; R indicates the function associate with the ASM; T indicates the feedback times of the kernel concern of the ASM; GU, GA and GF indicate $GU(ASM_i)$, $GA(ASM_i)$ and $GF(ASM_i)$, respectively.

As shown in the table, the most important evolutionary requirement is to settle the problem of "deviation" in the "location" function; and the least important one is to settle the problem of "Network flow waste".

VII. SUMMARY

In this paper, a multi-source information fusion based evolution requirements acquisition method for mobile apps is proposed. It synthesizes the information provided to App Stores, the feedbacks of the app and its similar apps to acquire evolutionary requirements. By ranking the ASMs according to their importance and user attention, most urgent evolutionary requirements can be found. The case study on Baidu Map show that the ranking is similar to its version history, which verifies the effectiveness of the method.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Plan of China (No. 2017YFB0503702).

REFERENCES

[1] William Martin, Federica Sarro, Yue Jia, et al. 2016. A Survey of App Store Analysis for Software Engineering[J]. Research Note of UCL Department of Computer:1-56.

[2] Cuiyun Gao, Hui Xu, Junjie Hu, et al. 2015. AR-Tracker: Track the Dynamics of Mobile Apps via User Review Mining[C]//Proceedings of the Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 284-290.

[3] Claudia Iacob, Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews[C]//Proceedings of the 10th Working Conference on Mining Software Repositories (MSR).41-44.

[4] Lawrence Bernstein, C. M. Yuhas. 2014. Software Requirements[M]. Apress, 73-106.

[5] Xie zhongwen, li tong, dai fei, etc. 2011. A paraconsistent meta-model of requirements for software evolution[J]. journal of Jiangsu university(natural science edition), 32(5):562-568. Doi:10.3969/j.issn.1671-7775.2011.05.013.

[6] Liyin Tang, Vinod Venkataraman, Charles Thayer. 2012. Facebook's Large Scale Monitoring System Built on HBase[C]//Proceedings of the Strata Conference.

[7] Dong Sun, Rong Peng, Wei-Tek Tsai. 2014. Understanding Requirements Driven Architecture Evolution in Social Networking SaaS: An Industrial Case Study[C]//Proceedings of the Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on. IEEE, 230-236.

[8] Borja Sanz, Igor Santos, Carlos Laorden, et al. 2012. On the Automatic Categorisation of Android Applications[C]//Proceedings of the 9th IEEE Consumer Communications and Networking Conference.149-153.

[9] Jieun Kim, Yongtae Park, Chulhyun Kim, et al. 2014. Mobile application service networks: Apple's App Store[J]. Service Business, 8(1):1-27.

[10] Lavid Ben Lulu David, Tsvi Kuflik. 2013.Functionality-based clustering using short textual description: helping users to find apps installed on their mobile device[C]//Proceedings of the International Conference on Intelligent User Interfaces.N/A.

[11] Diya Datta, Sangaralingam Kajanan. 2013. Do App Launch Times Impact their Subsequent Commercial Success? An Analytical Approach[C]//Proceedings of the International Conference on Cloud Computing and Big Data.205-210.

[12] Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, et al. 2016. Analyzing Ad Library Updates in Android Apps[J]. IEEE Software, 33(2):74-80.

[13] Gunwoong Lee, T. S. Raghu. 2014. Determinants of Mobile Apps' Success: Evidence from the App Store Market[J]. Journal of Management Information Systems, 31(2):133-170.

[14] Ning Chen, Jialiu Lin, Steven C. H. Hoi, et al. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace[C]//Proceedings of the 36th International Conference on Software Engineering.767-778.

[15] W. Maalej, H. Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews[C]//Proceedings of the Requirements Engineering Conference.116-125.

[16] Guzman E, Aly O,Bruegge B. Retrieving diverse opinions from app reviews//Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on. IEEE, 2015: 1-10.

[17] Sun D, Peng R. A Scenario Model Aggregation Approach for Mobile App Requirements Evolution Based on User Comments[M]//Requirements Engineering in the Big Data Era. Springer Berlin Heidelberg, 2015: 75-91.