Towards Mobile Twin Peaks for App Development

Towards Mobile Twin Peaks for App Development

Giovanna Avellis*, Julian Harty†, Yijun Yu‡
*InnovaPuglia SpA Bari, Italy
†CommerceTest Ltd. High Wycombe, UK
‡School of Computing and Communications, The Open University, UK

***Abstract*—Requirements of mobile apps are often hard to elicit from massive numbers of users, although it is important for the solution architecture to meet them. Mobile Twin Peaks approach is proposed as a process of developing apps concurrently and iteratively that incorporates bidirectional communications within a mobile app. The communications allow both requirements engineers and software architects to reach a consensus on functionalities and quality constraints and to adapt architectural design decisions appropriately. To recommend architectural design decisions to the developers, we aim to obtain architecture-critical requirements from a set of general apps by combining, for example, analytics, ethnographic study, and information retrieval. We argue that the effectiveness of these techniques could be evaluated by experimental case studies and by engaging with industry partners to perform action research.**

***Index Terms*—Twin Peaks, app development, mobile analytics, information retrieval, ethnographic studies**

I. Introduction

"What makes a good app?" Developers often ask this question because it is hard to know exactly what users really want. Similarly, "which app is good for me?" Users often cannot articulate their exact needs before using the product, either. App stores enable users to choose something that might fulfil their needs from their millions of apps, "for anything imaginable there got to be something there". However, the dilemma is that developers often do not know which features anonymous users desire, and users often do not know which app offers the qualities and features they can only express vaguely.

Figure 1, for example, shows the app store ratings for the same app in both Apple App Store and Google Play Store. Furthermore, Figure 2 shows some detailed comments in Apples App Store; those from Google Play Store are similar. Reviews tend to be terse and seldom provide sufficient information about requirements, so it is challenging for developers to decide on the perceived qualities of the app. Although it may be useful for a developer to know statistics such as the percentage of users who rate the app with five stars, and how many rate it with one, etc., they are not very useful for them to understand how well the users like this app. Similarly, the users may find many apps with similar ratings; but it is hard for them to know which app would best suit their needs. The rating is just an indication of the impressions voiced by a small percentage of the user base (only 1% of users may bother to provide such feedback), and users may often get confused how to compare seemingly similar apps.

In a nutshell, although app stores offer a channel for users to leave feedback such as comments and ratings, they are not effective when (1) comments are tersely expressed, (2) ratings are opinionated or inaccurate, (3) few users provide feedback, (4) users cannot memorize the contexts to provide timely feedback, and (5) most community feedback does not align with developers' mission [3].

The real challenge lies in establishing effective communications between the two sides, e.g., adding something similar to an interactive TV channel [8] to mobile apps, which has not been addressed in literature. Twin Peaks [21] is a recognised approach to the requirements engineering and software architecture research communities. It interweaves software requirements and architectures to achieve incremental development and speedy delivery in order to satisfy both developers and users. The name "Twin Peaks" emphasises the equal status given to requirements and architectures in incremental software development, an iterative process, which produces progressively and concurrently more detailed requirements and design specifications. Inheriting from the general Twin Peaks, our proposed Mobile Twin Peaks approach is inherently iterative to support incremental development of mobile apps by combining tried-and-tested components derived from established architectural patterns in mobile apps.

Methodologically, it is also more focused than the general Twin Peaks approach. To bridge the Twin Peaks between developers (for architectures) and users (for requirements) in the mobile world, we propose an implementation of the communication channel in a tangible conceptual product, similar to a chat room, to facilitate bidirectional communications, between users and developers, about the requirements for that mobile apps. Central to this mission is investigating live representations of satisfaction arguments [25] which capture the rationale on both sides of the communication channel.

To address these communications challenges, for the benefit of mobile software engineers, we have set out three objectives for this joint community:

1) to *understand* the factors that matter to app users by information analytics and market analysis;
2) to *localise* requirements to recommended design decisions in architecture by combining reverse engineering and structural traceability retrieval; and
3) to *evaluate* the effectiveness of communications by ethnographic user studies and experimental case studies.

To achieve these objectives, the research community needs to work together in the following sub-areas:

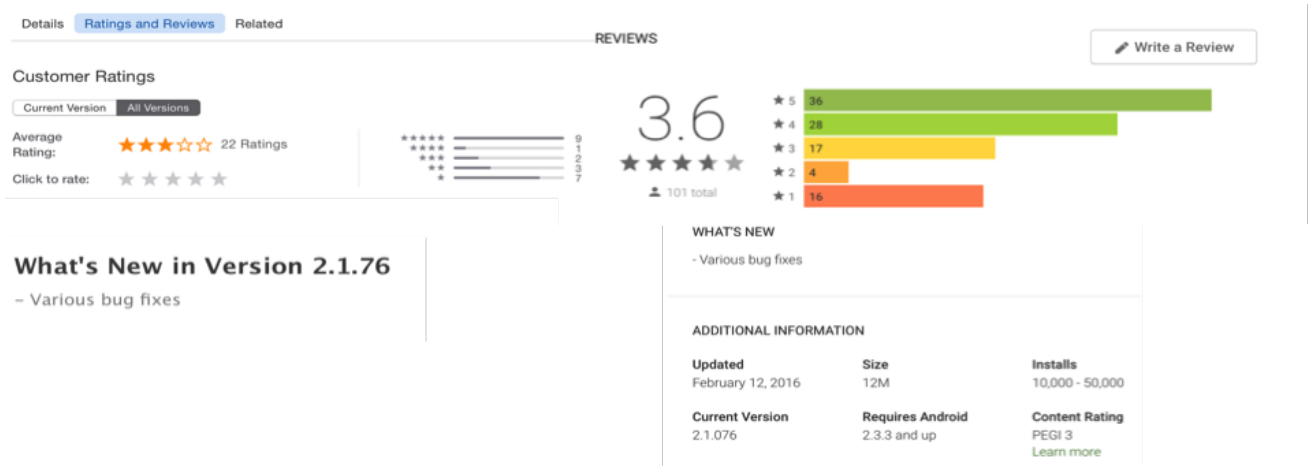

Fig. 1. Mobile App Ratings are not very informative on requirements or architecture

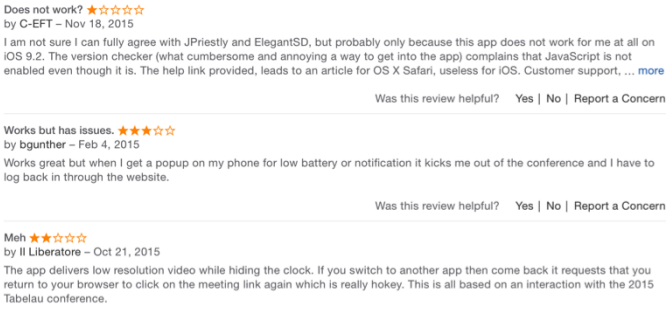

Fig. 2. Mobile App Reviews from the Apple App Store

1) **Mobile analytics and market analysis** to find out which requirements matter to the developers and users of mobile apps [13]. To facilitate these aims, app artefacts from software repositories and metadata will be collected from app stores, which has been happening through the mining software repository community. Developers can provide feedback on third-party libraries used in mobile apps, for instance for the Google Play Analytics SDK at the SafeDK Marketplace [22].
   In addition, gathering information on developer-user and developer-developer communication channels would be extremely helpful, e.g., by investigating the many feedback API's [16] and side-channel communities (e.g., high-quality code samples in Stack Overflow, Gists [9], and/or communications channels such as Slack). One challenge for such aggregation is to have a standard interchangeable format across markets and repositories. Moreover, the statistical datasets, after collection and analysis, still need to be validated with real users directly through ethnographic research methods to obtain empirical evidence, and the insights need to be checked with developers through action research methods.
   As we have said earlier, the *status quo* is only 1% of the users would actively leave feedback. To obtain their real opinion, measures need to be studied on how to usher them out of the silos and vote for the critical decisions to help their favourite win, using political election as an analogy. For effective preference elicitation and prioritization, it is also important to borrow methods from interdisciplinary studies e.g., well-known marketing analysis approaches such as apply Quality Function Deployment (QFD) and Kano methods [18].
2) **Program comprehension, refactoring, restructuring, and recommendation** are approaches to effectively improve mobile apps. Reverse engineering techniques, such as the horseshoe model [14], can help developers and architects *re-engineer* their software. Round-trip engineering, or re-engineering, allows developers to navigate between artefacts at different levels of abstraction, e.g., from architecture to requirements (i.e., reverse engineering), and from requirements to architecture (i.e., forward engineering). By applying reverse engineering tools on a range of candidate apps, the community could prepare a corpus of live updated traceability links between requirements and reusable artefacts, and combine with high-quality samples of third-party libraries, semi-automated retrieval of relevant reusable structures could be studied to enhance the quality feature requests, and automate the build with reusable structures continuously.
3) **Quality evaluation of apps and the engineering methods** are important to tell how effective the proposed quality injection process is able to enhance the quality of software products. This kind of study could aim to obtain quality criteria for mobile app development, the measurement of quality attributes, and monitor the quality metrics for the entire life-cycle. The community could combine qualitative and quantitative analysis approach to obtain their correlations in the mobile analytics dataset; carry out time-series analysis on the historical change of quality metrics recorded over the life-cycle apps; and evaluate these qualities by adding automated monitors to the communication channel.

The remainder of the paper is organised as follows: Section II

presents related work in this area. Comparing the proposed approach and research methodology for Mobile Twin Peaks to related work, Section III enumerates the main contributions in terms of novelty. Section III summarises the paper and calls for action to fulfil the proposed research agenda with fellow researchers in the community.

II. Related Work

Existing mobile apps research focuses primarily on software artefacts. For example, at the developers end, taint analysis [7] and instrumentation on binary or source form of apps are used to analyse whether they contain malicious code for security [15] and privacy [24] or whether certain features such as ads cause excessive energy and network consumptions for performance/usability quality [10]. At user's end, app store metadata such as app descriptions, popularity metrics such as number of downloads/installs, ratings, comments [6] are used for studying whether certain features are desired by certain cluster of users [23], or which features are offered by apps and which features are requested by users [11]. However, these artefacts-based studies have not captured the dynamics between developers and users and the rich contexts of use, which are equally important to analyse how effective they are communicating to each other. Technical books for developers teach them hands-on skills and best practices to make a mobile app using specific language/platform quickly. Guide books, blog's and magazine articles often prepare users to the lists of popular and useful apps they should know, and so on so forth. However, none of them provide readers with in-depth guidance on achieving effective and adaptive communications between developers and users. For example: What makes an app great? How to make it greater? How to let developers listen to the users? Why cant I do that [20]?

III. Mobile Twin Peaks Approach

In this section, we will characterise the key methods that are uniquely chosen for conducting research for the Mobile Twin Peaks mission. The original Twin Peaks model [21] is shown in Figure 3, where the requirements peak and the architecture peak are refined iteratively and concurrently. One of the key tenets is Rapid Change, which is especially necessary for live mobile apps where Platform updates are commonplace, security challenges are often in full view of potential attackers, and many users expect apps to be fresh. An iterative approach, such as Twin Peaks can help address these challenges faced by the developers in particular.

The research agenda will innovate on the specialised representations of requirements traceability links, focusing on iterative and evolving satisfaction arguments between the Twin Peaks of requirements and architectures. To do so, the research community could use semantic hypertext representation, flexible modelling, and reactive programming techniques from HCI/CSCW. Essentially, we aim to develop a handbook mobile app, which facilitates the communications between mobile app developers and their users.

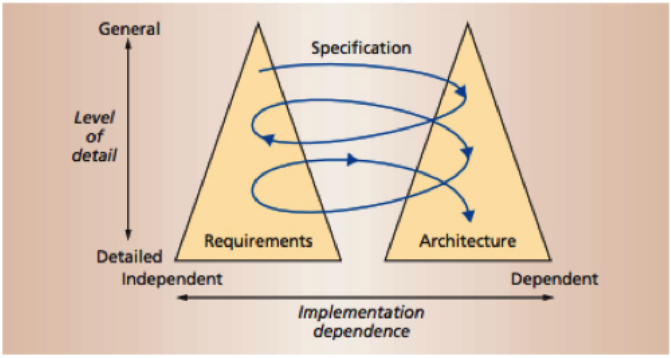

Fig. 3. Weaving together requirements and architecture

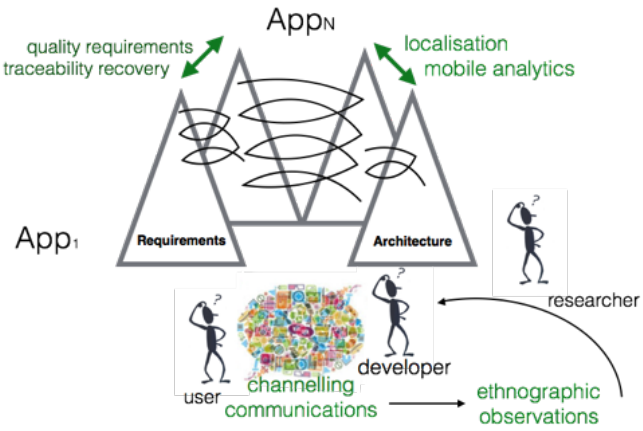

Fig. 4. Mobile Twin Peaks for App Development

Figure 4 illustrates the architecture of a proposed Mobile Twin Peaks app. It extends the Twin Peaks of a single software product to accommodate the large number of apps in the App Stores. Hence there are at least N peaks on either end of requirements or architecture, where N is the refinement step of requirements or architecture. Secondly, the interactions between each app may be useful for other apps. By leveraging on the architectural or requirements knowledge of other or similar existing apps, using the techniques such as traceability recovery, localisation, and mobile analytics, it is possible to establish indirect relationships between the co-evolving apps in the eco-system. Furthermore, the channelling communications between users and developers are now observable through ethnographic studies so that researchers could learn common patterns and heuristics to advise or recommend app developers on the architecture-significant requirements. This new architecture would facilitate two kinds of research methods, described as follows.

Ethnography in user case studies: To address unsolved research questions such as how software developers communicate with users about their quality requirements satisfaction in user studies, one option for the research community is ethnography [16], a qualitative research method to study people, cultures and their associated social and work practice.. Typically, ethnography study requires live observations that

do not obstruct the flow of development. In this case, the proposed handbook app facilitates the communication channel between developers and users; as a positive side effect, the logged events through user case studies would provide a rich source of information analytics for high-quality feedback.

Action research to 'eat own dog food': Developing a mobile app as part of the research agenda can be a complementary approach to ethnography as it offers an opportunity to learn-by-doing. Researchers can apply what they discover when new effective approaches are discovered in other research. However, we are conscious of the fact that our actions as researchers may invalidate the scientific findings when we perform activities meant to be observed remotely as a scientist (also known as the Uncertainty Principle in quantum physics). In the design of the action research methodology; therefore, after some internal clean up such as anonymization of personal identifiable information, the community could open up the resources and tools created from this research for other software engineers to replicate, and only after the observatory data at the ethnography studies stage have been archived.

Furthermore, new ideas compared with the current state-of-the-art, are described as follows.

Elicitation of NFR satisfaction arguments: Assessing Non Functional Requirements [4], [19] of mobile apps requires innovative approaches by mixing together direct and indirect elicitation approaches. Directly elicited NFR's from good practices in software engineering could be analysed systematically using goal-decompositions and operationalisations. Consequently, the goal-oriented evaluation consists of identifying the goals under evaluation, and defining the associated key performance indicators as measurable quality criteria for satisfaction arguments. Whilst direct elicitation approach is infeasible due to e.g. the lack of access to developers or users, natural language processing analysis of their communication logs provided in app stores could be used in classifying software defects and feedback to quality requirements according to latest ISO 25010 standard on Software product Quality Requirements and Evaluation (SQuaRE). Key performance indicators on the App Store such as daily ratings, retention rates, downloads, installs, and uninstalls, etc. offers a fine-grain evaluation of the cause and effect of changes at a given time. Such approaches could elicit evidence as an objective basis to provide satisfaction argumentation.

Maintaining live satisfaction arguments using bidirectional transformations: Any change to evidence could rebut or mitigate the claims of the satisfaction arguments of NFR's. To keep these satisfaction arguments alive, the argument structures need to be maintainable and updateable, and the results of argumentation to be highlighted in a dashboard, and triggering automated notifications through chat bots. To begin with an initial exploration, e.g., Oxford Universitys Haskell research group and NII Japan research group on bidirectional programming, are potential collaborators to this community by enhancing `pandoc` to document the quality requirements satisfaction arguments [26]. As the initiator of this community, we plan to open up the collaborations with active researchers in the community to introduce multiple forms of live argumentation.

Documenting architectural decisions in existing mobile apps: It is easy to say that documentation is necessary for mobile software development. However, in practice it is hard to do. The main reason often lies in tacit architectural decisions whose consequence to NFR's is felt too late. Here we would deploy requirements monitors, as reflective services built from structured information retrieval, to proactively collect feedback of poor architectural decisions. Architectural design decisions will be structured hierarchically so that the accuracy of requirements diagnosis can be achieved on demand at the right level of abstraction through, e.g., incremental model transformations [4].

Facilitating in-app bidirectional communications: Including effective communications in an app can significantly improve the volume and quantity of conversations, reviews, feedback, etc. Apptentive are a commercial company who provide "in-app communication tools". They provide an SDK that can be embedded in mobile apps to enable bi-directional feedback. They report their tools provide: a 330% in survey completion, a 15-fold increase in ratings, a 17% survey completion rate [2]; and "Product Decisions supported through feedback" [1]. These figures are significantly better than the industry expects and show the potential of effectively designed in-app communications. An important consideration is being available to respond to communications initiated by users, particularly for smaller teams with a widespread user base. Unless the team follows-the-sun [5] they need to find ways to provide acceptable, timely responses even if they're not available. Adaptive chatbots may be one approach provided they don't degenerate into a similar pit that beset Microsoft's Tay bot [17].

IV. Conclusions

This paper posits a new research agenda to bring mobile app developers and users together in eliciting, refining, and sharing their requirements and architectural design decisions in an integral, iterative, and inspiring process, using an in-app bidirectional communications channel. This channel may enable and encourage richer conversations and engage more users, further increasing the positive feedback cycles where developers then improve the app and users are 'rewarded' with improved apps where the improvements are based on their feedback so they are encouraged to participate again.

The conceptualisation of Mobile Twin Peaks develops the existing Twin Peaks methodology, however, other variants of the implementation can also be envisioned, for instance, to use mobile apps to support the existing software development such as the ReviewReviews app [12].

Acknowledgements

We thank Anthony Finkelstein and Bashar Nuseibeh for inspiring this work. In part it is supported by ERC Advanced Grant 291652 - ASAP.

REFERENCES

[1] Apptentive. Actionable customer feedback guides roadmap and reduces churn. http://cdn2.hubspot.net/hubfs/232559/Case_Studies/ActionableCustomerFeedbackReducesChurn.pdf, fetched on 2 March 2017.

[2] Apptentive. Why apptentive. https://www.apptentive.com/why-apptentive, fetched on 2 March 2017.

[3] J. Atwood. Listen to your community but don't let them tell you what to do. https://blog.codinghorror.com/listen-to-your-community-but-dont-let-them-tell-you-what-to-do, fetched on 2 March 2017.

[4] G. Avellis. Assessment of non functional requirements in mobile learning. In *the 10th World Conference on Mobile and Contextual Learning (Bejing, China, October 18-21, 2001)*, pages 1–8, 2011.

[5] M. Betts. 24/7 global application development? sounds good, doesn't work. https://en.wikipedia.org/wiki/Follow-the-sun, fetched on 2 March 2017.

[6] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 767–778, New York, NY, USA, 2014. ACM.

[7] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. D. McDaniel, and A. Sheth. Taintdroid: an information flow tracking system for real-time privacy monitoring on smartphones. *Commun. ACM*, 57(3):99–106, 2014.

[8] A. Finkelstein. Turn on, tune in. http://blog.prof.so/2011/06/turn-on-tune-in.html, fetched on 2 March 2017.

[9] GitHub Help. About gists. https://help.github.com/articles/about-gists, fetched on 2 March 2017.

[10] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 100–110, Piscataway, NJ, USA, 2015. IEEE Press.

[11] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: Msr for app stores. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 108–111, June 2012.

[12] J. Harty. The reviewreviews app. https://github.com/julianharty/app-store-reviews-app, fetched on 2 March 2017.

[13] J. Harty and A. Aymer. *The Mobile Analytics Playbook: A Practical Guide to Better Testing*. Hewlett Packard Enterprise, 2015.

[14] R. Kazman, S. G. Woods, and S. J. Carrière. Requirements for integrating software architecture and reengineering models: Corum ii. In *Proceedings of the Working Conference on Reverse Engineering (WCRE'98)*, WCRE '98, pages 154–, Washington, DC, USA, 1998. IEEE Computer Society.

[15] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan. Semantic modelling of android malware for effective malware comprehension, detection, and classification. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 306–317, New York, NY, USA, 2016. ACM.

[16] L. Merrick. 10 most popular user feedback tools for mobile apps. http://www.buzinga.com.au/buzz/user-feedback-tools, fetched on 2 March 2017.

[17] Microsoft. Tay bot. https://en.wikipedia.org/wiki/Tay_(bot), fetched on 2 March 2017.

[18] J. Mikuli and D. Prebeac. A critical review of techniques for classifying quality attributes in the kano model. *Managing Service Quality: An International Journal*, 21(1):46–66, 01 2011.

[19] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.*, 18(6):483–497, June 1992.

[20] A. Nhlabatsi, T. Tun, N. Khan, Y. Yu, A. K. Bandara, K. M. Khan, and B. Nuseibeh. why cant i do that?: Tracing adaptive security decisions. *EAI Endorsed Transactions on Self-Adaptive Systems*, 15(1), 1 2015.

[21] B. Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, 2001.

[22] SafeDK Marketplace. Measure everything about your mobile app. https://www.safedk.com/sdks/google-google-play-analytics, fetched on 2 March 2017.

[23] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain, and die in app stores. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 76–85, Aug 2015.

[24] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 25–36, New York, NY, USA, 2016. ACM.

[25] Y. Yu, V. N. Franqueira, T. T. Tun, R. J. Wieringa, and B. Nuseibeh. Automated analysis of security requirements through risk-based argumentation. *Journal of Systems and Software*, 106:102 – 116, 2015.

[26] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux. Maintaining invariant traceability through bidirectional transformations. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 540–550, Piscataway, NJ, USA, 2012. IEEE Press.