SnappView, A Software Development Kit for Supporting End-user Mobile Interface Review

XAVIER DE RYCKEL, Insight Labs, Belgium
ARTHUR SLUŸTERS, Université catholique de Louvain, LouRIM, Belgium
JEAN VANDERDONCKT, Université catholique de Louvain, LouRIM, Belgium

This paper presents SnappView, an open-source software development kit that facilitates end-user review of graphical user interfaces for mobile applications and streamlines their input into a continuous design life cycle. SnappView structures this user interface review process into four cumulative stages: (1) a developer creates a mobile application project with user interface code instrumented by only a few instructions governing SnappView and deploys the resulting application on an application store; (2) any tester, such as an end-user, a designer, a reviewer, while interacting with the instrumented user interface, shakes the mobile device to freeze and capture its screen and to provide insightful multimodal feedback such as textual comments, critics, suggestions, drawings by stroke gestures, voice or video records, with a level of importance; (3) the screenshot is captured with the application, browser, and status data and sent with the feedback to SnappView server; and (4) a designer then reviews collected and aggregated feedback data and passes them to the developer to address raised usability problems. Another cycle then initiates an iterative design. This paper presents the motivations and process for performing mobile application review based on SnappView. Based on this process, we deployed on the AppStore "WeTwo", a real-world mobile application to find various personal activities over a one-month period with 420 active users. This application served for a user experience evaluation conducted with N_1=14 developers to reveal the advantages and shortcomings of the toolkit from a development point of view. The same application was also used in a usability evaluation conducted with N_2=22 participants to reveal the advantages and shortcomings from an end-user viewpoint.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; **User interface toolkits**; *Usability testing*; *Systems and tools for interaction design*; *Ubiquitous and mobile computing systems and tools*; • **Software and its engineering** → Graphical user interface languages; Software testing and debugging; Acceptance testing.

Additional Key Words and Phrases: Application review, Code instrumentation, GUI testing, Heuristic evaluation, Mobile computing, Remote evaluation, Software Development Kit, Usability problem, Usability evaluation, User interface evaluation.

ACM Reference Format:
Xavier de Ryckel, Arthur Sluÿters, and Jean Vanderdonckt. 2022. SnappView, A Software Development Kit for Supporting End-user Mobile Interface Review. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 173 (June 2022), 38 pages. https://doi.org/10.1145/3534527

Authors' addresses: Xavier de Ryckel, Insight Labs, Limal, 1300, Belgium, xavier@insight-labs.com; Arthur Sluÿters, Université catholique de Louvain, LouRIM, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium, arthur.sluyters@uclouvain.be; Jean Vanderdonckt, Université catholique de Louvain, LouRIM, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium, jean.vanderdonckt@uclouvain.be.

2573-0142/2022/6-ART173 $15.00
https://doi.org/10.1145/3534527

1 INTRODUCTION

During the last two decades, there has been increasing interest in Application Review (AR) [14, 16, 18, 23, 27, 50, 56], a process that allows any person to provide developers of an interactive application any form of feedback to foster iterative design, continuous evaluation and improvement of the application. For example, Begel *et al.* [4] found that one of the most important questions interactive application developers ask is "what parts of the interactive application are used/loved by end users and why?". Among all components of an interactive application, the Graphical User Interface (GUI) is the most exposed component, thus being subject to particular reviews. The application review can be examined from the angle of the various Quintilian questions:

- *By who?* Reviews can be written and submitted virtually by any interested person, from alpha and beta testers to end users. Developers themselves are sometimes interested in writing a review of someone else's application.
- *For who?* Designers and developers of an application generally receive reviews to keep track of the appreciation of their product [56], taken as a whole or for certain features [18], to continuously maintain and improve the usability of their application [19]. Reviews are also estimated to be useful to convince new potential users of this application.
- *For What?* An application review could concern any part of an interactive application, a standalone desktop application, or an application running on a mobile device such as a smartphone or a tablet: from bugs belonging to the functional core [22], API interface communication problems [14] to cosmetic problems of visual design [58, 59].
- *How?* A review could be expressed with different levels of granularity, ranging from simple "Like" [18] or a rating (* to *****) to a detailed critique [36] written in a special format (*e.g.*, using pairs of adjective-noun words [60], structured feedback [58]) or free form comments [16]. For example, UX-Tips [39] enables end-users to enter their review in terms of textual usability problems, a form of negative review. A review is said to be *individual* when it concerns only one application in itself or *comparative* when it concerns an A/B testing method [28] between two or more candidates, usually to position the application under review [27]. For example, AB4Web [54] supports comparing two GUI design alternatives. The review is also said to be *summative* when a quantitative measure is provided in the review (*e.g.*, a rating, a ranking) or *formative* when a qualitative comment is formulated instead. This last format is useful as it can be directly transposed into instructions to improve the application GUI, while the former is only indicative.
- *Why?* Application reviews enable developers to continuously improve their application development based on end users' feedback. When positive reviews are posted on the application profile (*e.g.*, in an application store or on its website), along with positive ratings, potential users become more akin to download it and to use it. These data influence the position of the application in the charts and search results, increasing visibility and improving ranking.
- *With which method?* Numerous Usability Evaluation Methods (UEMs) [55], whose experimental conditions and protocols vary widely, that could become candidates for the review of applications for interactive applications in general [25] or mobile applications in particular [20]. Among them are mockup-based evaluation [35], model-based evaluation [7], measure-based evaluation [62], GUI testing [9, 53], GUI ripping [1], and code instrumentation [12]. The least demanding UEMs are typically used in application review, such as heuristic evaluation (when the review is guided by a set of heuristics to be assessed or simply checked) and free-form evaluation (when the review is left open) [5].

Yet, for both parties – stakeholders involved in the development life cycle and people involved in the usage–, the review process still poses many challenges. Reviewers do not always know

how to write reviews, even if a structured format is imposed [58], they are constrained by the GUI review process, which often contradicts or conflicts with the GUI being reviewed (therefore, other modalities should be used [56]), they cannot enter their review in the very right context of use where a usability problem occurs (instead, they are diverted to another separate GUI). This problem is multiplied by the number of versions of the same user interface (UI), namely, in the case of multiple UIs [13] for the same mobile application [42] and for UIs for different devices [24]. Several differences exist between reviews and ratings on mobile applications when compared to web-based systems, in particular in their preference for more concise and aggregated reviews [56]. When a dedicated reviewing system is employed, developers should add several extra lines of code to instrument their code throughout the application for enabling reviews to be properly collected and aggregated. Developers who receive these reviews are often overwhelmed by a flood of reviews with different levels of detail on many aspects of the UI. They do not know how to handle this mass of data, especially when crowdsourcing [59, 61] is used for reviewing. Only around 35% of reviews are actually helpful to developers in improving their apps [8].

To bridge this gap, this paper explores both the feasibility and the usefulness of developing SnappView (Shake 'n' review your application), a new Software Development Kit (SDK) specifically tailored to facilitate the end-user review of graphical user interfaces for mobile applications and to streamline their input into a continuous development life cycle. This provides the following original individual contributions, which, taken together, make SnappView unique:

- On the *conceptual viewpoint*: a domain model capturing UI reviews for any interactive application, in particular, for mobile devices.
- On the *methodological viewpoint*: a four-stage process for supporting end-user review of mobile UIs based on this domain model.
- On the *implementation viewpoint*: a new software development kit to support this method and a website to manage reviews collected through a project dashboard.
- On the *experimental viewpoint*: a longitudinal study reporting the results of applying the four-stage process supported by SnappView to "WeBoth", a real-world mobile application deployed in an application store for personal usage.

The remainder of this paper is structured as follows: Section 2 reports on some selected work related to application review and discusses the background in which this work is conducted to draw some lessons for SnappView. Section 3 introduces, motivates, defines, and details the four stages of the SnappView process based on the underlying models involved in each stage, *i.e.*, a class diagram for the domain model, and activity diagrams for the stages. Section 4 describes the implementation of the SnappView SDK. Section 5 presents the results of two evaluation studies, one aimed at developers and one aimed at end users, based on "WeBoth", a mobile application for managing leisure activities for a couple of two people instrumented with SnappView. Section 6 concludes the article and discusses some future avenues of this work by reporting the feedback obtained with SnappView.

2 RELATED WORK AND BACKGROUND

This research touches on two research topics: mobile user interface evaluation, where UEMs [55] for mobile applications are analysed, and application review, a particular UEM inspired by heuristic or free-form evaluation for mobile devices. We hereby discuss how our work is related to each of these areas, respectively, in the next two subsections. We give a partial conclusion for each of these areas and we conclude by covering the intersection of these two areas with the focus on UIs.

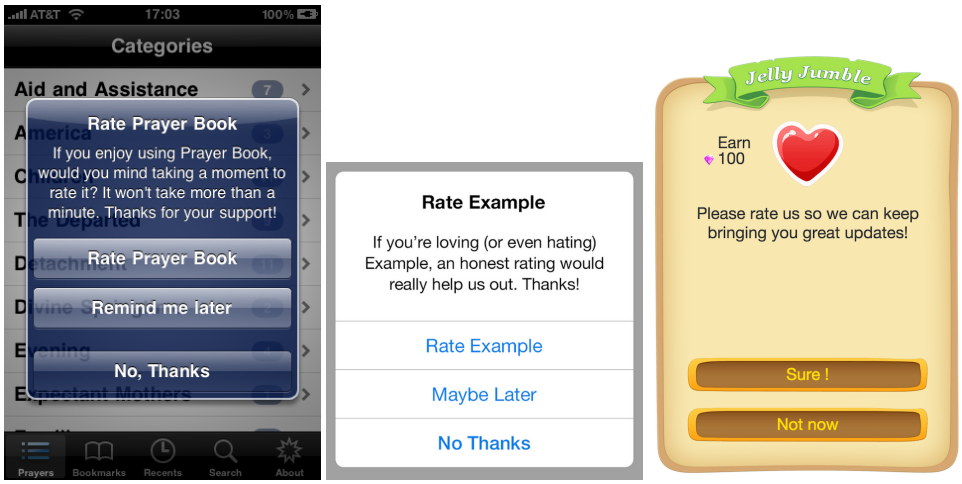

Fig. 1. Examples of rating software: (a) Appirater, (b) iRater, (c) pollJoy.

2.1 Mobile UI Evaluation

Whitefield *et al.* [57] provided a framework for classifying UEMs according to two axes: UI (real *vs.* represented) and end users (real *vs.* represented). An ideal UEM would be able to evaluate a real UI with real end-users, but this ideal situation is not always possible nor it is always cost effective, thereby suggesting to replace the real UI by a representation (*e.g.*, a prototype instead of the real UI) and the real end users by some representation (*e.g.*, a model or substitutes of end users). Based on this framework, application review is defined as a UEM performed by real users on the real application, particularly on its real UI. However, in principle any tester or reviewer, being a real end-user or not, can provide a UI review, whatever its representation is (*e.g.*, a real one, a screenshot, a wireframe, a sketch) with different levels of fidelity. Several UEMs have been considered to evaluate the user interface of an application, either manually [2] or semiautomatically [1], for example by ripping the GUI [1], evaluating the GUI as a service in GUICAT [9], guided testing [10], GUI bug finding [22]. Most of these UEMs do not transpose easily when evaluating the GUI of a mobile application that is used by thousands of people around the globe. It is not only difficult to instrument code [12], even with advanced methods such as bootstrapping [6], but the developer also faces the question of which method to handle the vast amount of data resulting from this process.

2.2 Application Review

Obtaining application reviews from experts, such as alpha or beta testers, is not only difficult (they are often involved in many other development tasks and are not always available), but also expensive (they need to be properly compensated for their participation according to their level of expertise). For these reasons, non-expert users have increasingly been transformed into application reviewers [27] to provide various forms of critique until the point of involving a very large number of such people in crowd-sourcing [61]. This is also partially supported by the observation that a crowd of users would cover multiple possible usages of the same application, especially when users have different experiences with the same device or not [42]. The feedback provided by the expert and nonexpert participants is usually encoded via an online form on a dedicated website, through requests on social networks, or even by the class peers writing a critique [58]. Novice feedback is assessed as helpful to developers, but is still perceived as not as valuable as expert feedback. Although a crowd of non-expert participants may lack relevant experience for reviewing

[56], by requiring a non-expert crowd to use a rubric to provide feedback, novice feedback was "rated nearly as valuable as expert feedback". This deviation is perceived even less significant when timeliness is the key factor for reviewing, especially when multiple versions of the same UI should quickly evolve over time.

Several software support the review process of UI [63], such as Applause, Apple's TestFlight, EvalComments [16], SpreadingWord [21], RevMiner [23], QuickReview [50] (a data-driven mobile interface to improve the quality and structure of application reviews), CrowdCrit [36], CrowdUI [44], AR-Miner [8], Voyant [59], and SplotLight [60], which demonstrates the effectiveness, efficiency, and usefulness of feedback provided by a non-expert crowd. Such systems usually gather enough diverse feedback from people, even if subjective, to reach a moderately objective conclusion [20] when some frequency is observed [21].

However, these systems still suffer from several common shortcomings: they impose some form of code instrumentation in multiple locations of the application, its UI code is not obvious to understand and to incorporate, the way reviewers can enter feedback is usually well and precisely managed, but the locus of control is almost always outside the context of use. Reviewers have to enter another system to write the review and cannot easily link their review to the contextual elements where the problem occurs, especially when it is a usability problem [20] or to provide a design critique [61]. This loss of context damages the quality of the review. When the instrumented application enables the end user to enter the review without changing the system, it is often conflicting in terms of interaction with the UI being reviewed. For example, Appirater (Fig. 1-a) is a software class to be incorporated into a mobile UI to remind users to rate an application after they used for a given period of time or number of times. However, reviewing an entire application on a 5-point rating scale, independently of its context of use and without linking it to any interaction history or screen, is a rather limited form of review without any formative feedback. Similarly to Appirater, iRater asks for a score on a 5-point rating scale depending on several parameters (Fig. 1-b), such as the number of days until the rating is asked, the required number of sessions, of events, of prompts, of minimal and maximal time period. PollJoy determine who should receive review requests and when (Fig. 1-c). For example, players who have completed the tutorial quest and who have reached level 5 and have played for at least 30 minutes. These rating applications do not provide any form of formative feedback.

On the evaluation of collected reviews, some approaches have already been proved successful, such as data mining [8], intelligent adaptation [50], crowd aggregation [36], to name a few. ReviewCollage [27] aims to compare two or more applications in the review by identifying the comparison criteria and estimating them. When extended to several applications, this process becomes a benchmarking that is probably not at the level of expertise of any end user, with the risk of producing fallacious or erroneous reviews. For example, HelpShift opens a chat between the end user and a bot to discuss an application, thus welcoming any type of textual comment. However, synthesizing textual comments remains a challenge. Text mining techniques, which are used to classify and summarize reviews of mobile applications, often produce complex outputs that are prone to overfitting. Instead, Jha *et al.* [26] suggested using frame semantics for application review to generalize textual reviews into more abstract scenarios).

To meet the demand for reliable reviews, researchers have explored technologies to connect designers with online feedback reviewers. While these systems support submitting some feedback, most of them do not support an end-to-end cycle. Foong *et al.* [15] proposed a conceptual framework to highlight critical processes that affect online feedback exchange. We contribute research questions for future feedback systems and argue that online feedback systems must be able to support designers through five activities that occur before, during, and after the feedback exchange, more from a psychosociological viewpoint.

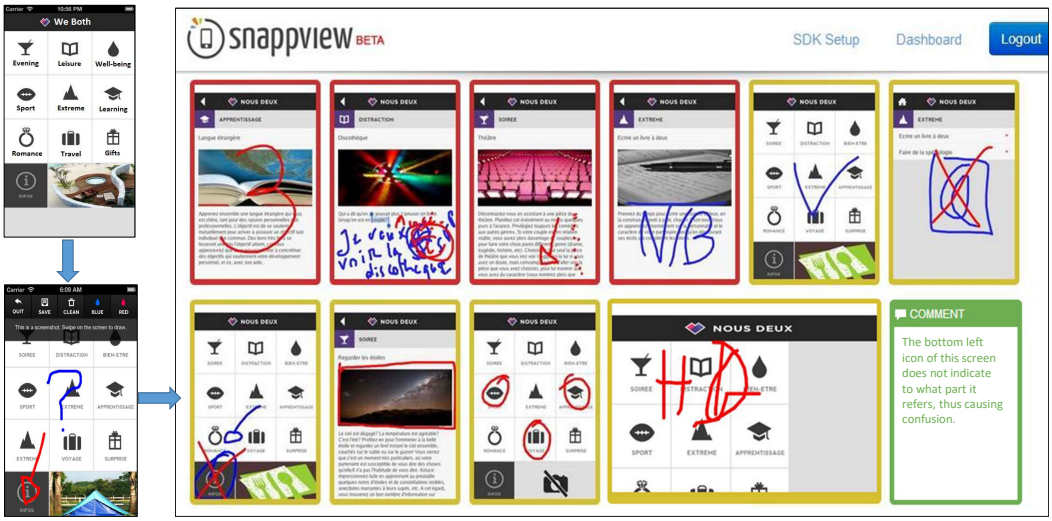

Fig. 2. Invoking SnappView on a mobile application, reviewing a particular screen, and a collection of multimodal feedback for application review.

The Interactive Thread [37] is a design method that collects detailed contextualised data from a large user population while sharing interaction design methods with professional designers of different disciplines. Crowdsourcing [45], which allows employing people worldwide to perform short and simple tasks via online platforms, can be a great tool for filling application reviews in a time and cost-effective way. On the other hand, the crowdsourcing environment does not allow for the degree of experimental control, which is necessary to guarantee reliable data. For example, CrowdUI [44] is a web-based software that supports website review by allowing users to visually express their positive or negative feedback with their suggestions to fix negative aspects.

In summary, we note that the invocation of the application review should be tightly integrated with the application under review (R1), with as little and understandable instrumentation as possible (R2), that the contextual conditions in which a screen is reviewed should be transmitted (R3). The tester should be able to provide feedback in a multimodal way to expand the expressiveness of reviews (R4): text entered, drawings performed by stroke gestures, vocal comments, and video records. Submitting the review should also remain a straightforward activity so that the results can be properly received, stored, organized, and structured (R5). With respect to R1, SnappView will be incorporated in the application code instead of calling an external application, like for most other review applications such as Appirater, iRater, and pollJoy. With respect to R2, SnappView should be below the workload imposed by similar review applications. In the case of TestFlight, the effort should remain comparable. Regarding R3, SnappView is expected to be superior in terms of context coverage to other review applications, but at the same level as TestFlight. SnappView will be the most distinctive with respect to R4 due to its multimodal ability, allowing the reviewer to prefer the review according to any preferred modalities. With respect to R5, SnappView offers a website where reviews are stored and structured, compared to a series of emails or a list of comments in many review applications. Again, TestFlight is the reference in terms of quality of reviews.

The next section presents how the four cumulative stages of SnappView fulfill these requirements. Fig. 2 shows a review invoking SnappView, taking a screenshot with textual comments and drawings using stroke gestures, and browsing the feedback in decreasing order of importance and frequency, with various configurations, such as smartphone *vs.* tablet, landscape *vs.* portrait mode.

3 THE FOUR STAGES OF THE SNAPPVIEW PROCESS

To leverage the capabilities of UI review while minimizing the code instrumentation from the developer's point of view and facilitating the UI review from the tester's point of view, SnappView provides an open source software development kit that facilitates end-user review of GUIs for mobile applications and streamlines their input into a continuous design life cycle. Fig. 3 overviews the SnappView process, which is decomposed into four cumulative stages, each step being described in the next subsections.

3.1 Stage 1. Account Management and Project/SDK Preparation

SnappView distinguishes stakeholders into four types of users (top of Fig. 4):

(1) The *Creator*, or *Conceptor*, is the stakeholder responsible for developing the application to be reviewed and willing to gather formative feedback while developing, such as a developer, a programmer, or a practitioner.
(2) The *Tester* is any stakeholder who is going to install and use the application developed by the creator and provide the creator with formative feedback while interacting with the application. A tester could be in principle any person: ideally any end user, any marketing person, but also a designer, a graphic artist, a reviewer, an experimenter or a researcher. Sometimes, a tester could even be the creator of the application or the creators of other applications, in the case of cross-application reviews.
(3) The *Writer* is a stakeholder writing an article about the application under review or providing any form of feedback that should be captured outside the SnappView scope. For example, the *Store Reviewer* is a writer checking that the application developed by the creator is compliant with design, development, and style guidelines promoted by the vendor of the application store. For instance, it could be an Apple reviewer for Apple's App Store, an Android reviewer for Google Play, or a person working at TestFlight, an external party serving as an intermediary between the creator and the store reviewer to collect preliminary feedback before releasing the application to the application store.
(4) The *Administrator* is the stakeholder responsible for managing the entire SnappView ecosystem and its projects introduced by creators.

After subscribing to the SnappView website with credentials and logging in, the creator who is willing to create a new project downloads and installs the SnappView SDK (if not already done). For creating a project, the creator fills in the corresponding form located on the dashboard (*e.g.*, an application name and various geographical data) and validates it, which makes the system create an effective project in the database, store its corresponding information, and produce a Universally Unique Identifier (UUID), a 128-bit label used for identifying any information in computer systems worldwide [29]. This UUID will serve as an identifier for the future application. To instrument any application with SnappView, some technical steps are required (Appendix A). Once this code instrumentation is completed, the source code of the whole project should be compiled and linked to build an archive file ready to be delivered to the testers.

3.2 Stage 2. Application Distribution and Preparation

The creator, after building a distributable archive, has two options: the archive is submitted to an application store to be reviewed by the corresponding store reviewer (*e.g.*, an Apple Reviewer will check whether the application is compliant with the Apple's code and usability guidelines) or to TestFlight for an intermediary check before submitting it to the application store. Both cases result into an article stored with the application under review.

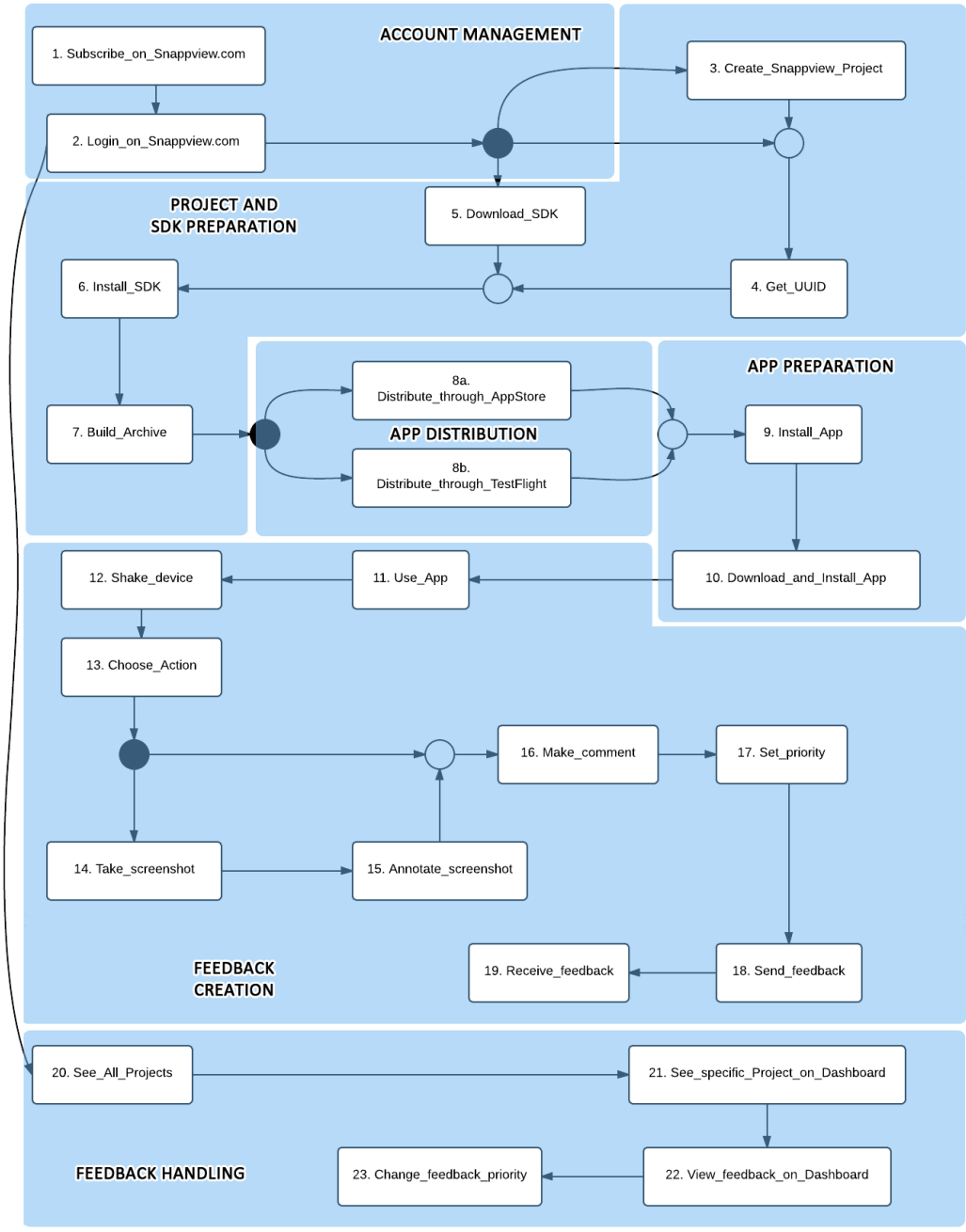

Fig. 3. SnappView process structured into four cumulative stages.

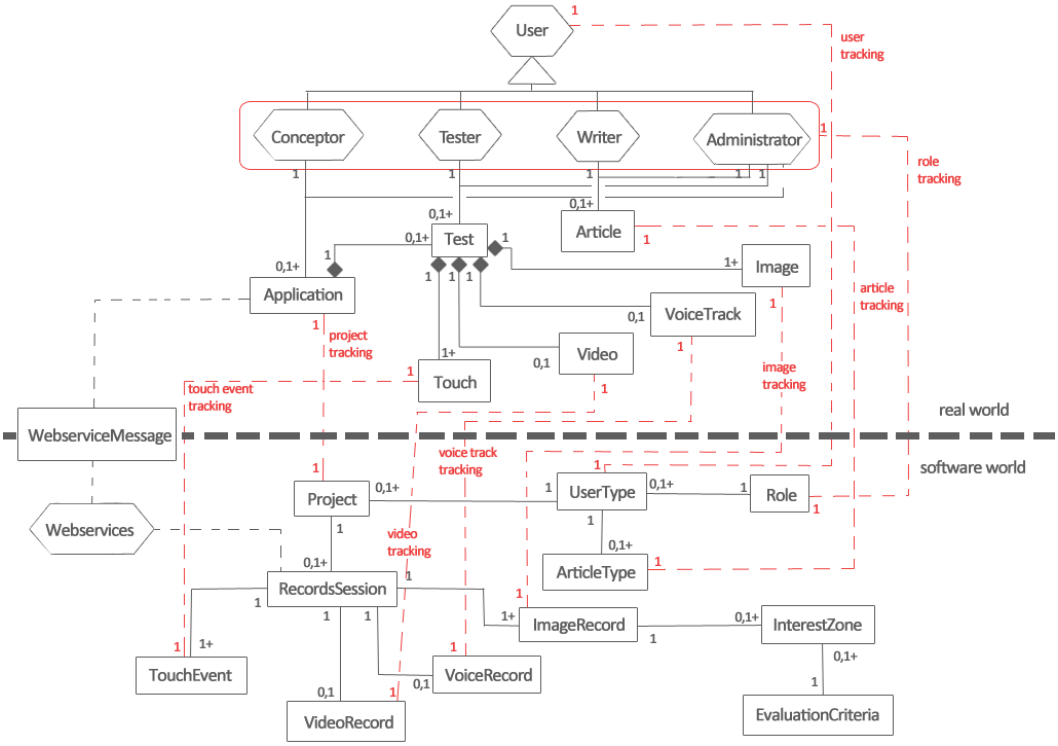

Fig. 4. Software and real worlds in SnappView.

3.3 Stage 3. Feedback Creation

After downloading, installing, and using the application, the tester can decide to follow a random plan (the GUI screens are browsed randomly) or a particular scenario (*e.g.*, according to a certain task defined) to perform the test. When the tester initiates the testing, *e.g.*, when something is wrong or when a comment should be left, the tester simply shakes the mobile device to invoke the pop-up SnappView (Fig. 2, left) to take a screenshot or leave a comment. In both cases, the tester is asked to specify the level of importance: *low* (depicted in green), *medium/neutral* (depicted in yellow) and *high* (depicted in red). For example, Fig. 2, right, reproduces a set of four severe screenshots with annotations, six medium screenshots with annotations, and one low-importance comment, all being captured on a smartphone in portrait mode, except one captured on a tablet in landscape mode. For any test of an application (see "real-world" upper part of Fig. 4), the tester is free to add a drawing performed by touch gestures (*e.g.*, digits, letters, symbols, text), a voice comment or a video if allowed by the device camera. The screenshot is stored as an image. SnappView creates the complete feedback by adding data related to the device (*e.g.*, which brand, make, model, operating system and version, which resolution), to the context if permitted (*e.g.*, which orientation) and sends it to the project website through the use of specific web services (left of Fig. 4).

3.4 Stage 4. Feedback Handling

To handle feedback received from testers for an application, the creator logs on the SnappView platform to display the dashboard of ongoing projects: Fig. 5 depicts a dashboard where projects have different status (*i.e.*, idea, under development, and deployed) and kinds of feedback (*e.g.*, text, screenshot with or without drawing, vocal comment, or video). Feedbacks are sorted in decreasing

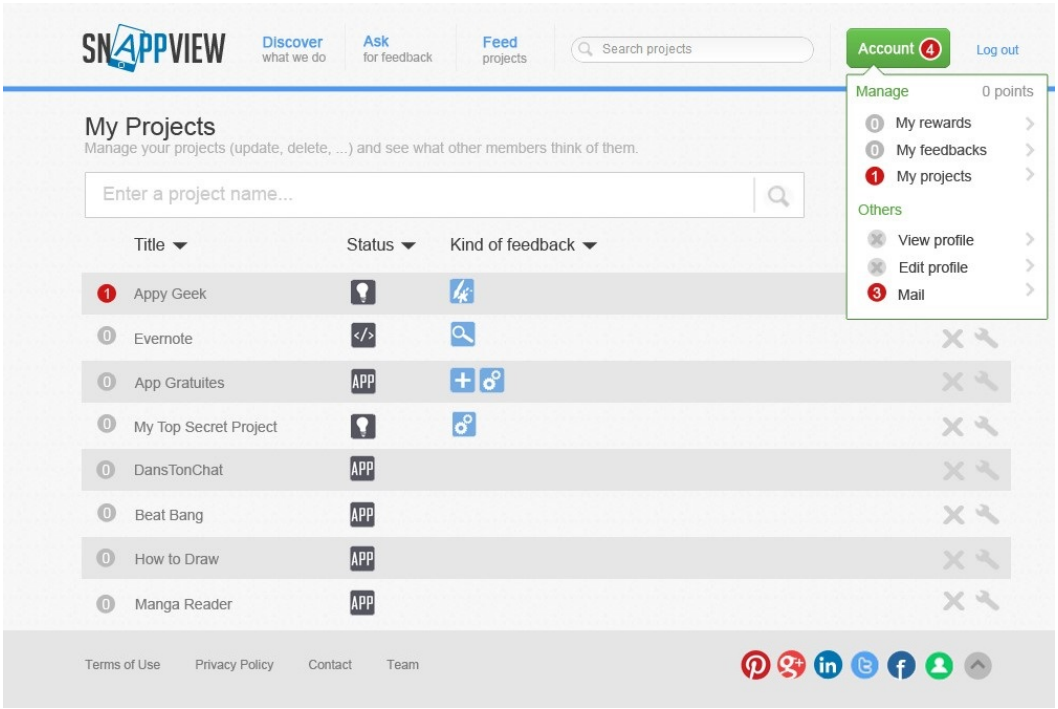

Fig. 5. Dashboard of SnappView projects.

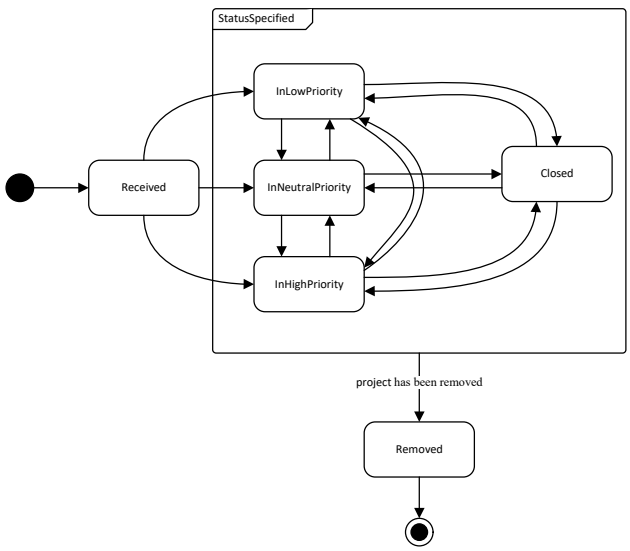

Fig. 6. State-transition diagram of a feedback.

order of their level of importance. Depending on the progress of the project, the creator can adapt the level of importance of any feedback received according to the state transition diagram represented in Fig. 6 until its closure.

4 SNAPPVIEW IMPLEMENTATION

SnappView has been implemented in Objective-C, which targets any Apple iOS compatible application that must be executed on a smartphone or tablet. SnappView is an Objective-C library that can be embedded in a mobile application. An APK beta version is also produced. SnappView relies on AFNetworking, which is a networking library developed on top of the Foundation URL Loading System to extend Cocoa's networking abstractions built into Cocoa, on Reachability to probe the current status of the network and its accessibility, and on UIDevice-with-UniqueIdentifier to identify any user playing the four roles defined. Fig. 4 represents a simplified version of the SnappView domain model as a UML V2.5 class diagram. This figure is decomposed into two parts as follows:

- The *real world*, where the mobile application under review is subject to a series of tests that involve the four types of user defined in Section 3.1. Each test can be made up of text, touch-based stroke gestures on the screen, voice records, video records, or an article by a writer. This information is transferred to the SnappView repository thanks to Web services and is attached to a project.
- The *software world*, which stores and manages information related to any project related to an application. A project involves users playing various roles and is attached to any recorded session. For example, a screenshot is stored as an image record, composed of different zones of interest, depending on how many gestures have been produced on the screenshot. Each zone can be assigned to an evaluation criterion, such as a heuristic, an ergonomic criteria [3], a usability guideline, or freeform comment.

In general, SnappView supports four scenarios represented as UML V2.5 activity diagrams:

- The *complete scenario* (Fig. 21) represents the most common scenario of an application review. The shake of the mobile device triggers SnappView thanks to its "application delegate", which allows one to listen to its accelerometer. If this movement is recognized by SnappView, the listener is deactivated to freeze the context of use and to automatically capture a screenshot of the application with related contextual data, such as screen orientation, resolution, device, screen size (in AppviewUtils). AFNetworking will be activated to probe and manage the connection between the mobile device and the SnappView repository, depending on the network status and accessibility. Depending on what the end user is providing for the review, a text, a series of stroke gestures, a vocal comment, or a video record activated via the device's camera, these data are sent to SnappView and stored in the repository. When the complete review is submitted, SnappView returns to its normal functioning.
- The *comment only scenario* (Fig. 22) represents the scenario of a review consisting of a textual review submitted by a writer. It is a simplified version of the previous scenario.
- The *normal scenario* (Fig. 23) represents the scenario that occurs when no disconnection from the Internet occurs, thus allowing SnappView to immediately store the review in the repository.
- Text *worse scenario* (Fig. 24) represents the same scenario when the mobile device is disconnected from the Internet for any reason, thus making its web services unavailable. In this case, the end user still performs the review by providing the multimodal content as desired, SnappView stores this information locally, periodically probes the Internet connection through AFNetworking, and sends this information as soon as the connection is restored.

Table 1 summarizes the requests received by the SnappView web services, which are responsible for managing and storing the various elements of the multimodal review. After its execution, the web service returns an HTTP code, such as 200, 404, 500, to notify AFNetworking if the data transfer was successful.

Request	Definition
K	Returns the project UUDI, which enables the mobile device to send any related information to the corresponding SnappView project
C	Returns the text related to a review (test), which could be a text associated to a screenshot explaining the comment or a textual comment in the case of a global review by a writer
L	Returns the level of importance, *i.e.*, low, medium, or large, of a review
I	Returns the universal date and time of a review
O	Returns the orientation of a screenshot submitted in a review
U	Returns the feedback type, "C" for a textual comment, "S" for a screenshot,...
DM	Returns the brand and model of the mobile device used
DN	Returns the device type, *e.g.*, a mobile phone, a smartphone, a tablet
IL	Returns the location where the review was submitted
IV	Returns the version of the operating system used on the mobile device
SV	Returns the SnappView version used on the mobile device for creating the review

Table 1. Definition of requests supported by SnappView web services.

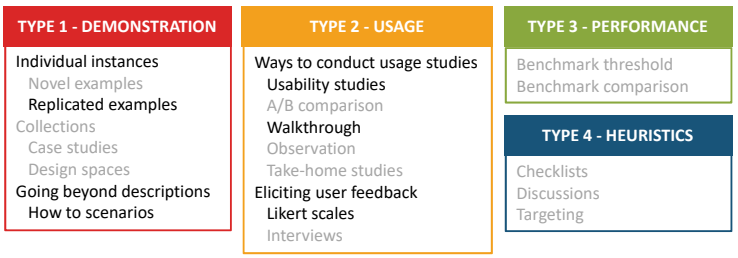

Fig. 7. Ledo *et al.* [30] evaluation strategies applied to SnappView are highlighted in black.

5 EVALUATION

Ledo *et al.* [30] surveyed 68 toolkit papers to classify their evaluation strategies into four types: demonstration, usage, performance, and heuristics. We evaluated SnappView based on two types and multiple strategies as follows (Fig. 7): strategies belonging to "Type 1-Demonstration" and "Type 2-Usage" evaluate what a toolkit can and who could use that toolkit. Individual instances refer to any example going throughout the process of the toolkit. Instead of creating a novel application, we create a *replicated example* of a mobile application intended for the general public (Section 5.1). This individual instance is then deployed and used in a *how to scenario* to illustrate the four stages of the SnappView process (Section 3) and to allow a system *walkthrough*, which consists of showing SnappView to a pool of potential developers to go beyond simple description and *eliciting their user feedback* and overall impressions using *Likert scales* [34]. A first study evaluates the user experience of SnappView process with N_1=14 developers (Section 5.2) while a second study evaluates the usability of the application resulting from SnappView as perceived by N_2=22 real end-users (Section 5.3). In the light of these evaluation strategies, our five requirements stated in Section 2 are then revisited in Section 5.4.

5.1 Individual Instance: the "WeTwo" Application

To evaluate the feasibility of SnappView, we conducted a one-month longitudinal study (from April 18 to May 18) on "WeBoth", a mobile application for managing personal activities (Fig. 8) between two people, such as members of a couple or a relationship. These activities include (Fig. 2, left part): evening activities (*e.g.*, restaurant, dinner, gala events), leisure activities (*e.g.*, exhibitions, museums),

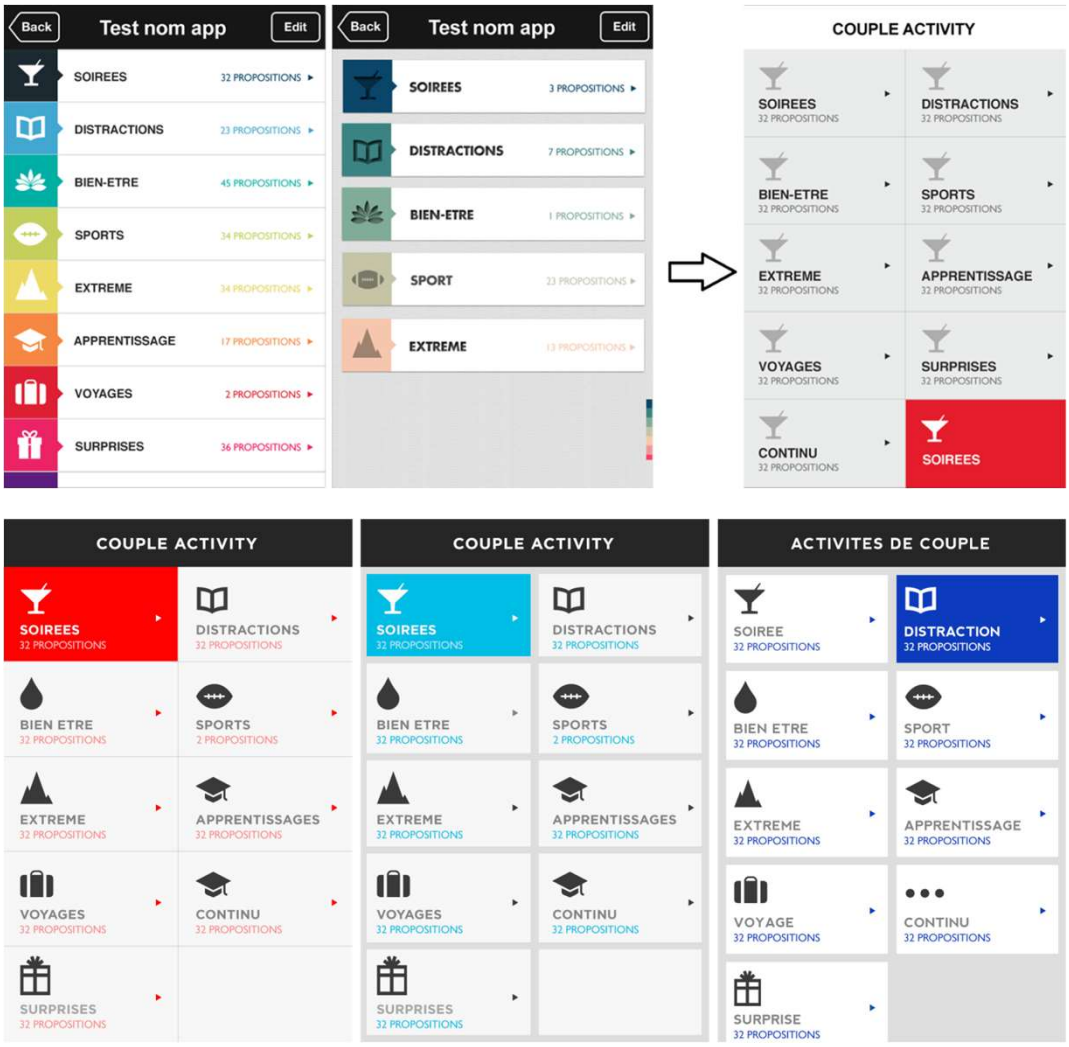

Fig. 8. Activities of "WeTwo" mobile application.

well-being activities (*e.g.*, sauna, massage), sport (*e.g.*, outdoor *vs.* indoor sports), extreme activities (*e.g.*, benji, skydiving), learning (*e.g.*, online courses), romance (*e.g.*, jewelry, bouquet), travel (*e.g.*, by train, by plane, all-in-one stays), and gifts. Once developed and instrumented according to the instructions reported during Stage 2 (Section 3.2), the application was published on the Application Store (Fig. 9) and was highly promoted on social networks (*e.g.*, FaceBook and Instagram) and by word of mouth. During this period, 420 new users registered with 8.04 average sessions per user, totalizing 1,278 sessions for an average time of 41 sec (Fig. 26). During this period, an average of 28.79 new users registered per day, all together producing 10,272 screen views. 295 reviews were collected. Fig. 25 shows the evolution of new users of "WeBoth" during the review period, as recorded by Google Analytics. Fig. 26 provides an overview of all users logging in/out of the "WeBoth" application during the reviewing period.

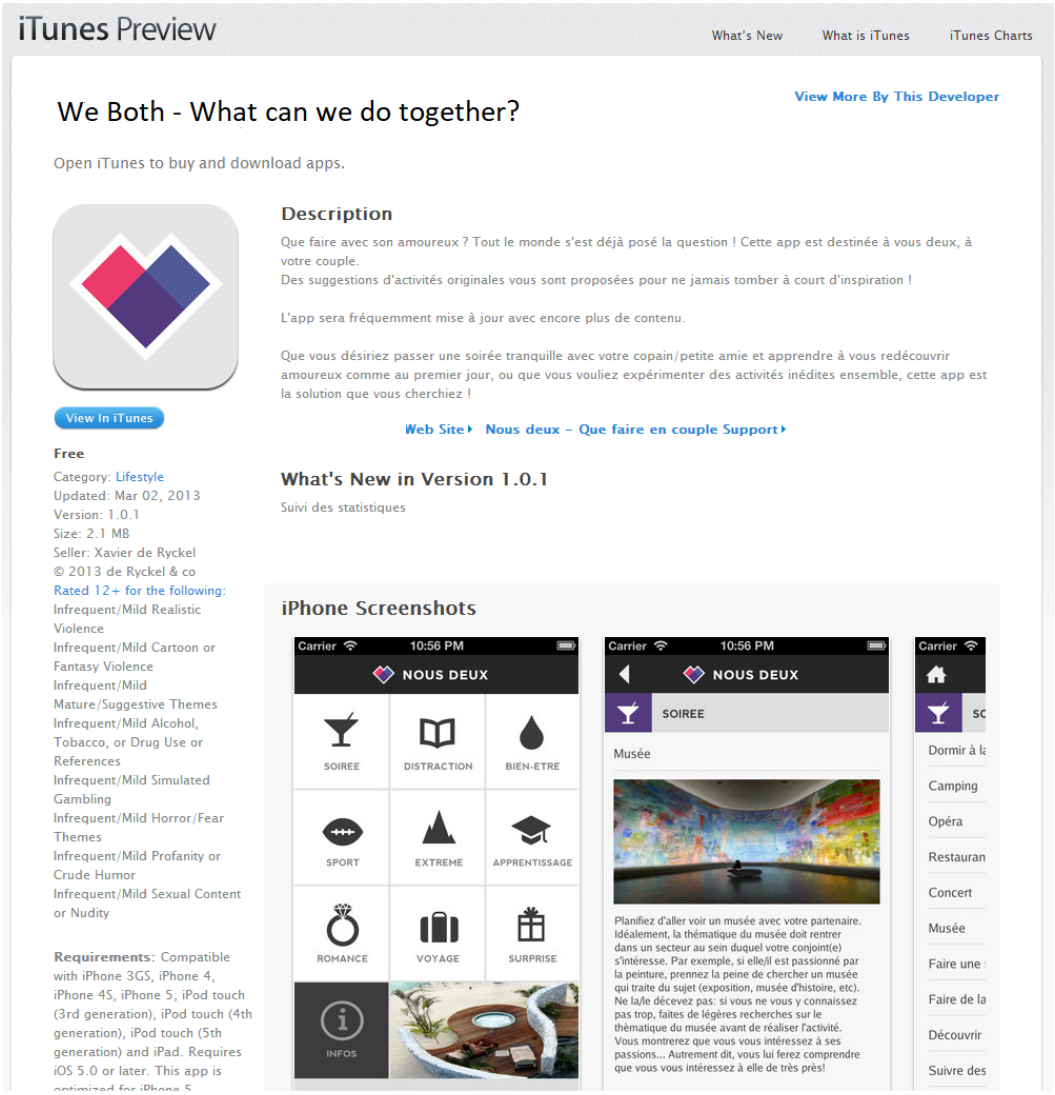

Fig. 9. How to install the test application from the application store?

5.2 User Experience Study with Developers

5.2.1 Recruitment. Participants were recruited from a mailing list of about 200 volunteers maintained at Anonymous. An email was sent to the mailing list asking for developers willing to assess SnappView. No compensation was offered to volunteers. To select volunteers with representative coverage and enough experience, each volunteer was asked to:

- Fill in a GDPR-compliant demographic questionnaire, which also includes statements about the frequency of use of a computer, a smartphone, a tablet, a game console, and a MS Kinect (expressed on a 7-point rating scale, 1=minimum, 7=maximum), as well as the number of years of experience in application development in general and in mobile application development in particular.

- Position herself/himself on Costabile *et al.*'s [11] continuum of development expertise, which consists of nine levels (Fig. 10): on the left side is Internet surfing, where no genuine development actually occurs; on the right side is professional application development. Each level defines both the activities a developer is able to perform and the corresponding development skills required to reach this level.
- Express her/his digital skills on Rose *et al.*'s [46] empirically validated questionnaire for assessing somebody's skills in exploiting digital technologies, such as the Internet, mobile applications, etc. This questionnaire was administered mainly to confirm the previous position.

The nine levels are defined as follows:

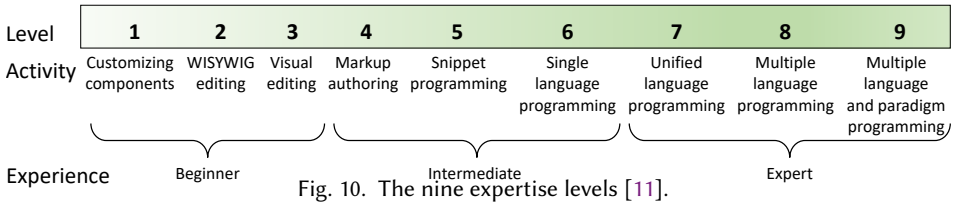

Fig. 10. The nine expertise levels [11].

(1) *Customizing components*: developers adjust existing GUIs by setting values to parameters, such as customizing service preferences, adaptation, and display.
(2) *WYSIWYG editing*: developers edit the GUI presentation in a WYSIWYG editor.
(3) *Visual editing*: developers create GUIs visually by adding, customizing, and connecting components, such as in mashup editors and component-based editors. Visual editing is possible without having learned programming, but requires a basic understanding of the programming paradigms.
(4) *Markup authoring*: developers produce GUIs using plain markup languages. This code editing uses a markup language, which a minimal understanding of declarative programming.
(5) *Snippet programming*: developers copy, paste, manually edit existing source code for templates called *snippets*. This is performed in a dedicated editor which automatically generates code.
(6) *Single language programming*: developers have a sufficient knowledge of a single programming language and they are able to develop a piece of functionality from scratch.
(7) *Unified language programming*: developers have sufficient knowledge of a single programming language, for creating all components of an interactive application.
(8) *Multiple language programming*: developers have a thorough command of several programming languages and combine them.
(9) *Multiple language and paradigm programming*: developers are professional programmers as they combine several different technologies to entirely build an interactive application. Programming paradigms can be declarative, imperative, or multi-paradigm.

A sample of N_1=14 participants (5 women, 9 men) was derived from this selection, aged 22 to 57 years (M=39.64, SD=10.24). All reported very frequent usage of computers (M=7.0, SD=0) and smartphones (M=6.43, SD=1.18), but to a lesser extent for tablets (M=3.86, SD=2.10), game consoles (M=2.57, SD=1.92), and Microsoft Kinect (M=1.36, SD=0.72). They occupy diverse responsibilities such as, but not limited to: consultant, project manager, assistant, employee in an Internet Service Provider, finance advisor, manager, and developer.

Their Costabile *et al.*'s level of development expertise (M=7.79, SD=1.57, MD=8) matches a profile of highly experienced developers, as well as their position on Rose *et al.*'s questionnaire [46]: 'I consider myself knowledgeable about the Internet and digital practices (M=6.43, SD=0.62), 'I am extremely skilled at using the Internet (M=6.14, SD=0.91), 'I am an expert in the internet and

digital technologies' (M=5.71, SD=1.28), and 'I know somewhat more than my friends about the internet and digital practices' (M=5.93, SD=1.03) all received very high average ratings. Participants reported a high average number of years in application development (M=13.50, SD=8.98, MD=11) and in mobile development (M=4.86, SD=4.10, MD=4).

5.2.2 Stimuli and Material. To provide the participants with a uniform methodological support that guarantees experimental consistency, a Microsoft PowerPoint was created to explain in general the four SnappView stages (Fig. 3), as described in Section 3, and to illustrate the actions corresponding to each stage that the participants will be required to perform on the "WeTwo" application:

(1) Stage 1. Account Management and Project Preparation: the participant, playing the role of a creator/conceptor, creates an account on the SnappView web site, logs into the system, creates a sample project entitled "WeTwo" to obtain a corresponding UUID. In the meanwhile, the SnappView SDK should be downloaded and installed. The development instructions (see Appendix A), the most important part, were illustrated one by one in the presentation.
(2) Stage 2. Application Distribution and Preparation. The participant is instructed to build an archive file containing the instrumented project and to post it in the SnappView project displayed on the dashboard (Fig. 5). The participant was not required to post the application to any application store in order to shorten the process. Indeed, such a submission would create multiple entries of the same project, which will be immediately rejected.
(3) Stage 3. Feedback Creation. In order to shortcut this stage, an imaginary project was made available with various reviews created by reviewers of the same application.
(4) Stage 4. Feedback Handling. The participant is instructed to browse the reviews of this project.

An archive file containing the entire source code of "WeTwo" was communicated, but without any instrumentation.

5.2.3 Task and Procedure. The aforementioned presentation with the application code was communicated to each participant who performed the experiment in their own development environment to preserve ecological validity. Each developer is instructed to follow the instructions contained in the presentation as if they were on their own in a real development context of use, but without imposing any time constraint to avoid any pressure. After going through the four stages, each developer is asked to perform a UEQ+ evaluation [49], a questionnaire to assess user experience with a product composed of several scales (*e.g.*, Attractiveness, Efficiency, Perspicuity, Dependability, Stimulation, Novelty), each scale being assigned a weight for its importance (on a 7-point rating scale) and decomposed into four subscales to be evaluated (*e.g.*, Attractiveness is decomposed into four subscales: annoying *vs.* pleasant, bad *vs.* good, unpleasant *vs.* pleasant and unfriendly *vs.* friendly).

UEQ+ was selected as an evaluation method because it is a modular and modern interface evaluation method where scales can be decided based on the interface to evaluate and covers the user experience (UX), not just usability, as assessed by questionnaires such as IBM PSSUQ [32] and SUS [47]. UEQ+ is also easy to administer to participants and remains valid even with a limited number of participants. For some scales, but not all unfortunately, a benchmarking of their values lead to an interpretation into five effect sizes [48]: bad, below average, above average, good, and excellent.

5.2.4 Results and Discussion. First of all, all scales benefit from a very high value of Cronbach's alpha coefficient, ranging from α=0.82 for Dependability to α=0.95 for Trust, suggesting that the scales are sufficiently consistent among participants. Developer answers were interpreted with the UEQ data analysis tool and are reported in Fig. 11, which depicts the means for the UEQ+ scales resulting from the evaluation. UEQ+ uses 7-point rating scales to measure each scale, *e.g.*,

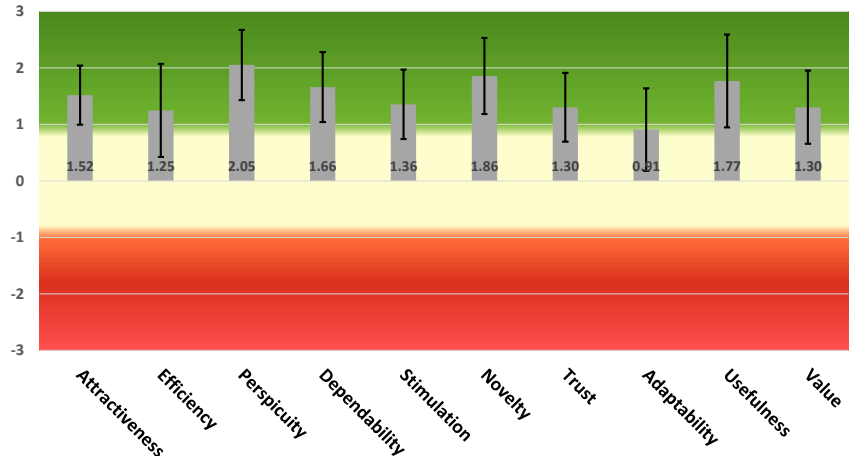

Fig. 11. Scale means for UEQ+ questionnaire (Horrible bad: -3.0 to -.8, Neutral: -0.8 to 0.8, Extremely good: 0.8 to 3.0). Error bars show a confidence interval of 95%.

Attractiveness (overall impression of the system), Perspicuity (how easy is to get familiar with the product), Efficiency (can users solve their tasks with the product without unnecessary effort), Dependability (does the user feel in control of the interaction), Stimulation (how exciting and motivating is to use the system), and Novelty (how innovating and creative is the system). According to Schrepp *et al.* [49], "it is extremely unlikely to observe values above +2 or below -2,..., the standard interpretation of the scale means is that values between -0.8 and 0.8 represent a neutral evaluation of the corresponding scale, values > 0.8 represent a positive evaluation and values < -.8 represent a negative evaluation".

Only Adaptability returned an average scale value below 1 (SM=0.91), but still above +0.8, which reveals a moderately sufficient consideration of this criteria. The mean scores of all other scales ranged from 1.25 for Efficiency to 2.05 for Perspicuity (M=2.05 is interpreted as 'excellent' in the benchmarking [48]), thereby demonstrating a sufficient consideration of all the other criteria and that, overall, the developers belonging to the sample were satisfied with SnappView. After

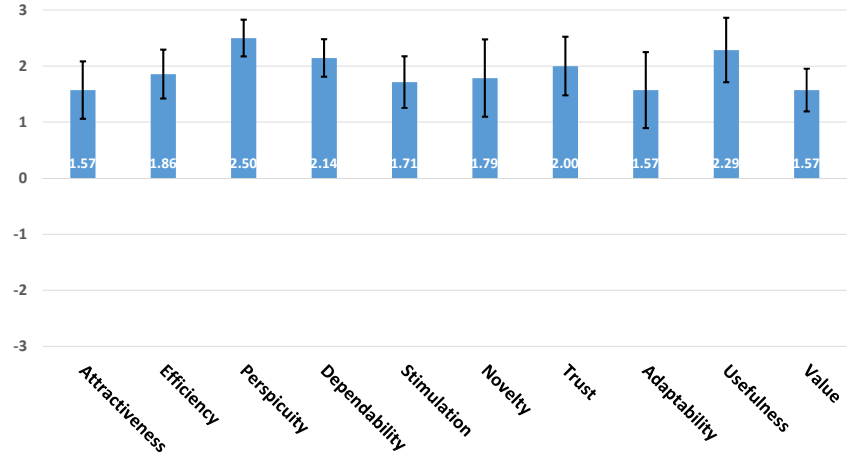

Fig. 12. Importance ratings for the UEQ+ scales. Error bars show a confidence interval of 95%

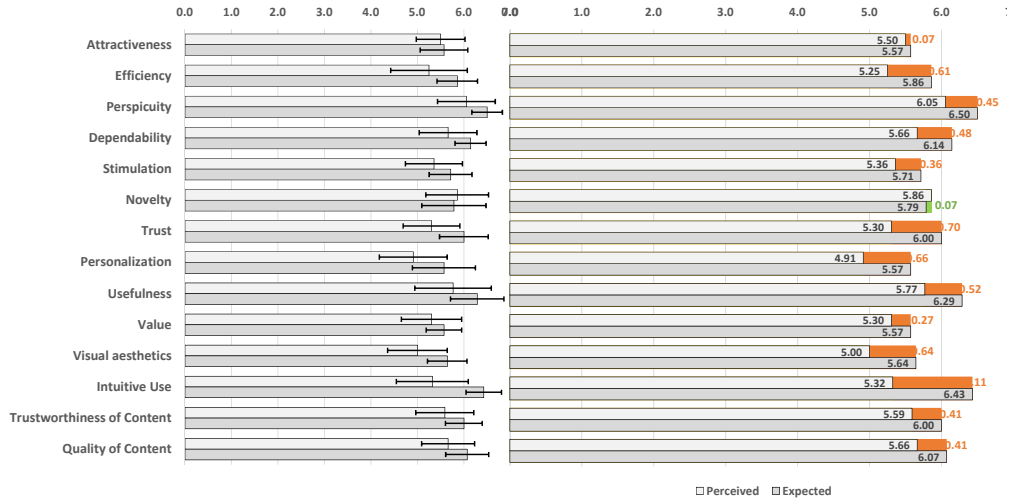

Fig. 13. Difference between expected and observed importances. Error bars show a confid. interval of 95%.

Perspicuity, the subsequent scales are: Novelty (SM=1.86, interpreted as 'excellent' in the benchmarking [48]), Usefulness (SM=1.77), Dependability (SM=1.66, interpreted as 'excellent' in the benchmarking [48]), Attractiveness (SM=1.52, interpreted as 'good' in the benchmarking [48]), Value (SM=1.30), Stimulation (SM=1.36, interpreted as 'good' in the benchmarking [48]), Trust (SM=1.30), Efficiency (SM=1.25, interpreted as 'above average' in the benchmarking [48]).

Fig. 12 shows the importance ratings assigned by participants to each UEQ+ scale. All importance ratings ranged from 1.57 (for Attractiveness, Adaptability, and Value) to 2.50 for Perspicuity (for all scales: M=1.90, SD=0.23). To evaluate to what extent these importance ratings were effectively addressed by SnappView, Fig. 13 highlights the deviation between the averaged importance rating for each scale (Expected) and the averaged rating obtained for all subscales of the corresponding scale (Observed). Fig. 13, left, presents the averages and the standard deviations between the Expected and the Observed, while Fig. 13, right, depicts the deviation between them. All scales were expected to be essential (M=5.94, SD=0.90), ranging from Personalization (M=5.57, SD=1.29) to Perspicuity (M=6.50, SD=0.63). Only Novelty surpassed its expectations, but with a small deviation though (Δ=0.07). All other scales saw their expected means exceeding their observed means. Although these dark orange negative deviations in Fig. 13 are relatively small in magnitude (M=0.51), they suggest that some improvements are requested to match the expected level for these scales. Attractiveness obtained the smallest negative deviation of all scales (Δ=0.77), thus suggesting that this scale was best addressed. However, Intuitiveness obtained the largest deviation (Δ=1.11), thus suggesting that this scale is the first to improve. From the optional comments, we observed that the participants repeated three main comments:

(1) Perspicuity is highly appreciated as it enable testers to freely express their reviews according to the format they prefer instead of imposing any of them or reducing the review to a few scores. The flip side of this medal is that it requires a comprehensive interpretation of the reviews by looking at them one by one. This is why Efficiency is classified so low.
(2) SnappView is written in Objective-C and distributed as a manually bundl-able library, which induces some worries about SnappView not being kept up-to-date, resorting to manually updating/rewriting the bundled libraries or drop the framework altogether in a not-so-distant future. Instead, SnappView should be distributed either through Cocoapods or SwiftPackageManager, which are estimated as the quickest and easiest way for developers to integrate new libraries into their application.

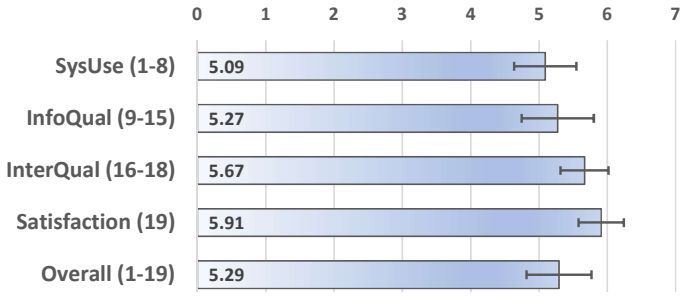

Fig. 14. Metrics computed for the IBM CSUQ. Error bars show a confidence interval of 95%.

(3) Integrating SnappView as a library into a mobile application should not increase its size too much and should not impact its startup time. Some developer recommend that a low startup time should be < 100 ms and that an acceptable application size increase should be around 2 MB or maximum 10% of the application size.

5.3 Usability Study with Real End-Users

This section reports the results of a qualitative evaluation conducted with real end users who used the "WeTwo" application and reviewed it with SnappView.

5.3.1 Participants. A random sample of 35 participants was created from an internal mailing list of volunteers maintained in our organisation. They were contacted first by email with a summary of the usability study and then by phone for further explanation, for collecting their agreement if positive, and for planning the experiment. This resulted into a subset of N_2=22 respondants (5 females, 17 males, with a response rate of 62%), having different backgrounds (*i.e.*, restaurant, administration, social security, economy, and law). The participants were then sent an email with the instructions to follow for the experiment, which consists basically of three steps: a link to download the "WeTwo" application from the application store and to install this application on their smartphone, an invitation to use this application for a period of one month, with a deadline to create reviews before the end of the trial period.

5.3.2 Task and Procedure. Participants were instructed to sign a GDPR-compliant consent form, complete a sociodemographic form, and, after the 1-month trial period, return by email an IBM Computer Satisfaction Usability Questionnaire (CSUQ) [31], a 19-item questionnaire that aims to evaluate the perceived usability of a system in terms of System Usefulness (SysUse), Information quality (InfoQual), Interface Quality (InterQual), and subjective satisfaction (Satisfaction) on a 7-point Likert scale [34]. This questionnaire is highly correlated with the real usability of the interface being tested ($\rho \geq 0.89$ for all metrics with α=0.03 according to [33]), its statements are generic enough to be applied to any user interface [32], it is used in a wide variety of contexts of use, even beyond simple user interfaces [17]. Two optional fields ended this questionnaire, one for writing up to three positive comments and one for up to three negative comments.

5.3.3 Results and Discussion. Participants used their own iPhone (41%), iPad (32%), iPod Touch (14%), or an unspecified smartphone (14%). 18% of participants completed their secondary school, 68% completed a university degree (*e.g.*, bachelor or master), 9% have more than a university degree (*e.g.*, a master after a master), and 5% came from a high school. Since the distribution of the answers to the 19 questions does not follow a normal distribution (all Kolmogorov-Smirnov normality tests did not pass with α=0.05), we computed for each question a Wilcoxon signed rank test for a simple sample with ties and continuity correction (with 10,000 iterations) to determine whether the

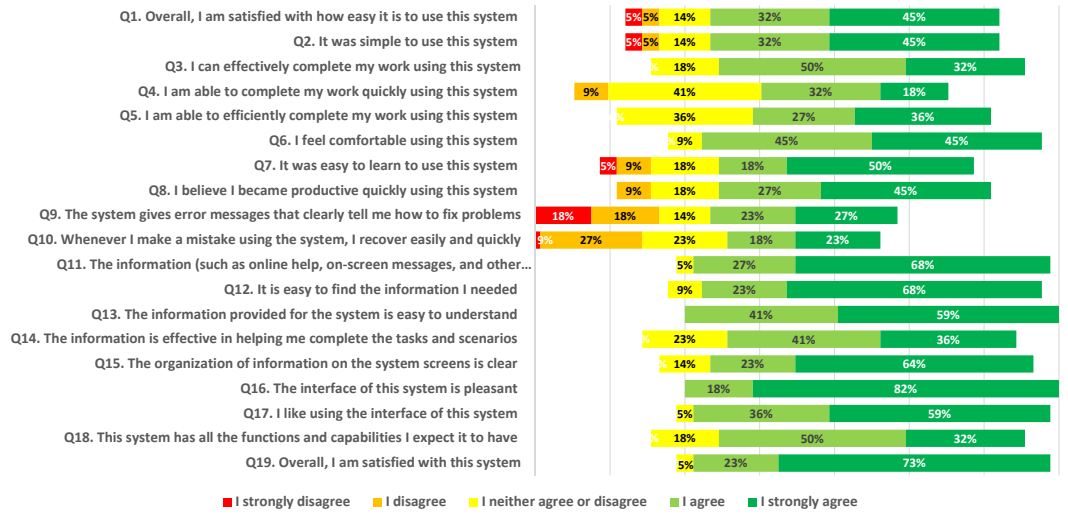

Fig. 15. Distribution of answers to the IBM CSUQ questionnaire.

answers are significantly departing from the median value MD=4, either above or below. Indeed, a question could receive an average answer above the median value, but not significantly. In the case of the IBM CSUQ, it is usually admitted that the average should be above the median MD=4, but better to be above 5.

Overall, the CSUQ results give the following metrics (Fig. 14): SysUse (items 1-8, M=5.09, SD=1.09), InfoQual (items 9-15, M=5.27, SD=1.27), InterQual (items 16-18, M=5.67, SD=0.84), Satisfaction (item 19, M=5.91, SD=0.79), and Overall (items 1-19, M=5.29, SD=1.14) all return an average value above 5, which denotes a very good appreciation of the various aspects. Apart for the interface quality and the satisfaction, answers are dispersed (standard deviations exceed 1). The internal consistency is moderately reliable (Cronbach's α=0.56, which is interpreted as 'poor', but Guttman's λ_4=0.90, which is interpreted as 'good'). The interrater reliability is estimated as moderate (Kendall' W=0.20, df=18, $p \leq 0.001^{***}$) with a small effect size (r=0.16).

Question Q1="Overall, I am satisfied with how easy it is to use this system" (M=5.47, SD=1.87) and Question Q2="It was simple to use this system" (M=5.47, SD=1.31) are significantly above the median value of 4 (p=0.0016**) with a large effect size (r=0.62). However, 5% of the participants expressed some confusion after shaking their smartphone: "*Once the screen is frozen, it looks like I can do everything on the screen, but I actually do not know how to express my review and how this review will be effectively transferred to a developer who should correctly interpret it and take it into account*". The rest of the participants did not express any confusion and directly went to their review by gesture.

Question Q3="I can effectively complete my work using this system" (M=5.00, SD=0.87) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.81). No participant expressed any concern about this question since all answers are above the median, thus suggesting that SnappView is effective enough but perhaps not efficient enough, as revealed by question Q4="I am able to complete my work quickly using this system" (M=5.21, SD=0.92), which is significantly above the median (p=0.0054**) with a large effect size (r=0.54). 9% of participants were negative and 41% were mitigated about this aspect, the largest percentage of all mitigated answers.

Question Q5="I am able to efficiently complete my work using this system" (M=5.79, SD=0.71) somewhat moderates this perceived lack of efficiency: while it is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.71), no negative responses were received but still

with 38% of participants being mitigated. A participant reported: "*I can quickly draw the region of concern, but it is very slow to write the comment, especially by gestures on screen and I do not know when to stop*".

Question Q6="I feel comfortable using this system" (M=4.95, SD=0.52) is significantly above the median ($p \leq 0.0001^{****}$) with a large effect size (r=0.62), as well as question Q7="It was easy to learn to use this system" (M=4.74, SD=0.87), which is also significantly above the median (p=0.00079***) with a large effect size (r=0.63). A participant clearly summarizes this: "*It is straightforward to use the reviewing system, but this freedom induces the other side of the coin: its lack examples on how to adequately review*".

Question Q8="I believe I became productive quickly using this system" (M=5.21, SD=1.03) is significantly above the median (p=0.00013***) with a large effect size (r=0.72). This question was considered rather irrelevant because reviewing does not usually require a maximum task completion time and there is no productivity goal.

Question Q9="The system gives error messages that clearly tell me how to fix problems" (M=6.05, SD=1.13) and question Q10="Whenever I make a mistake using the system, I recover easily and quickly" (M=5.47, SD=1.50) are the two most concerned questions: they are not significantly above the median (p=0.24, *n.s.*) and 36% of participants criticized that the system does not offer any form of guidance and does not tell when an error could occur since no error management is offered. A participants reported "*When entering my review, I see that the system records everything but does not tell me whether when I am right in what I did*".

Question Q11="The information (such as online help, screen messages, and other documentation) provided with this system is clear" (M=5.32, SD=1.16) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.86). However, the most frequent negative comment received in the free field is its lack of content and its lack of personalization. We recognize that the application was delivered with too few activities per category and that they were all generic, but this problem is mainly due to the contents of the application, not to SnappView.

Question Q12="It is easy to find the information I needed" (M=5.47, SD=0.84) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.86). Question Q13="The information provided for the system is easy to understand" (M=5.53, SD=0.84) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.88). Question Q14="The information is effective in helping me complete tasks and scenarios" (M=5.26, SD=1.10) is significantly above the median ($p \leq 0.0001^{****}$) with a large effect size (r=0.82).

Question Q15="The organization of information on the system screens is clear" (M=5.21, SD=1.08) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.82). Question Q16="The interface of this system is pleasant" (M=5.68, SD=0.89) is the question which received the most positive answers, being significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.89) but also no negative or mitigated answer. Only two questions fall into this category: Q13 and Q16. This suggests that shaking the smartphone and gesturing the review "*looks like a cool feature and does not give the feeling of entering a critique*". Question Q17="I like using the interface of this system" (M=5.21, SD=0.79) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.86). Although question Q18="This system has all the functions and capabilities I expect it to have" (M=5.11, SD=1.33) is significantly above the median ($p \leq 0.0001^{****}$) with a very large effect size (r=0.80), it also received some negative comments, such as the lack of geolocalization, the lack of automatic recognition of the review input.

Question Q19="Overall, I am satisfied with this system" (M=5.16, SD=1.21) is significantly above the median ($p \leq 0.0001^{****}$) with a large effect size (r=0.86). This question assesses the overall perceived usability of SnappView with only 5% of participants being mitigated, 23% being positive, and 72% of them being very positive, which concurs with the metrics reported in Fig. 14.

Gesture	Intention
Simple arrow, circle, ellipsis, rectangle or closed freeform	Designate a UI element or a zone
"Minus" symbol, cross symbol or equivalent (*e.g.*, α)	Delete a UI element
Double arrow, arrow with horizontal basis, arrow linked to another part	Move the UI element or zone towards the pointed position
Exclamation point, "Attention" sign	Attract attention to an important UI element or zone, often negatively
Question mark	Express a misunderstanding about a UI element or question the presence or absence of a UI element
Smiling emoticon	Express a positive statement
Sad emoticon	Express a negative statement about a UI element
Curve, line, often repeated	Underline the importance or the annoyance about a UI element

Table 2. Mapping between a touch stroke gesture and its intention.

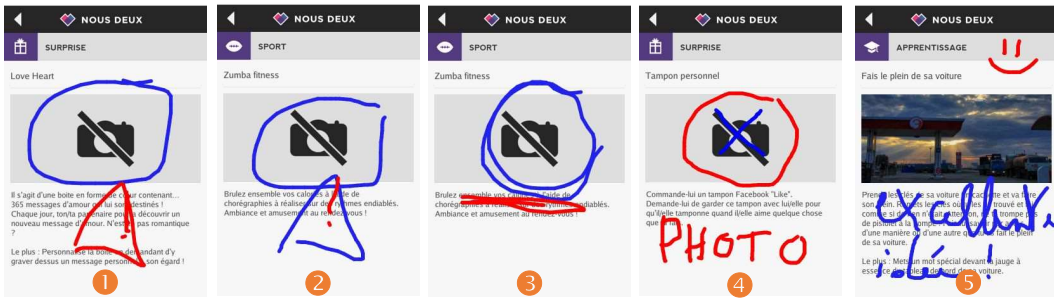

Fig. 16. Various types of annotations by stroke gestures.

5.3.4 Usage Analysis. Freeform stroke annotations are important and frequent in digital documents [52] and there are many different ways to capture and interpret them [51]. Table 2 summarizes the main annotation types collected in the 295 reviews received. While a deictic gesture, represented by a simple arrow, a circle, an ellipsis, or a rectangle, is the most common type for designing any UI element, we observe that two types or more are also often combined in a single review. Testers also extensively used various colors to convey a message associated with the review. For example, a blue arrow often, but not always, indicates a UI element that was satisfying, while a red arrow or a red symbol is used to designate a usability problem. For example, Fig. 16, 1-4, expresses the same review intent but are inconsistently represented by four different testers. The common ground is that the ellipsis, sometimes close to a rectangle, sometimes close to a circle, is used to designate a UI element of concern and that the red colour conveys dissatisfaction. Fig. 16-5 reveals an example in which the tester was a real end-user who used a finger to write a comment with a gesture instead of writing the comment with the virtual keyboard.

Although the gestures, the symbols, and the colors produced by the testers do not always share the same meaning, some trends could still be observed as summarized in Table 2, whose left column lists gestures on a screenshot and right column denotes the most frequent meaning.

5.4 Lessons Learnt and Limitations

Our five requirements resulting from Section 2 have been addressed as follows:

- *Integration of SnappView in the application* (R1- see Section 3.1): to guarantee a seamless transition from interaction mode to reviewing mode, the tester should only shake the mobile device to freeze the interaction and switch to the reviewing mode. Submitting a review frees the device. From the tester's viewpoint, this integration is transparent and appreciated (Section 5.3.3), but from the creator/conceptor's viewpoint (Section 5.3.3), the integration of SnappView into a released application requires several steps, which burdens the practitioner. Some developers also question the use of SnappView insofar as they are already using TestFlight, Apple's reviewing system, which risks causing duplication. Therefore, the requirement is considered to be 70% fulfilled.
- *Minimal instrumentation* (R2- see Section 3.1): precise instructions are given to perform code instrumentation of the target application, a stage that usually lasts less than ten minutes, which was expressed as a reasonable time (Section 5.3.3). Therefore, the requirement is considered to be 90% fulfilled
- *Automated transfer of contextual conditions* (R3- see Fig. 4 and Table 1): SnappView requests supported by the web services cover a wide range of contextual conditions which could be captured at review time and transferred to the repository, such as data related to the device, the environment, and the user if permitted. However, previous actions and interaction history are not captured at review time. Therefore, the requirement is considered to be 60% fulfilled
- *Multimodal feedback* (R4- see Fig. 4 and Section 3.3): the tester can submit feedback in various modalities, such as the text entered in an article, drawings performed by stroke gestures, vocal comments, and video records, all with a level of importance that can be controlled by the creator. Four scenarios are supported, as explained in Section 4. Therefore, the requirement is considered to be 100% fulfilled.
- *Structured management of feedback* (R5 - see Fig. 5). For each project, the creator is able to manage the feedback received depending on the level of importance, status, and the type of feedback (Fig. 4). New reviews are notified directly on the dashboard. But there are no data mining or machine learning techniques applied in reviews to better classify them. Therefore, the requirement is considered to be 70% fulfilled.

Although these requirements were mostly met, SnappView has some limitations revealed by the two aforementioned studies conducted with developers and end users:

- *No stroke gesture recognition*: numerous efficient recognizers exist [38] to accurately recognize stroke gestures on mobile devices. Recognizing these gestures and their color does not represent an elusive problem, even on mobile devices, but their interpretation is subject to too many variations depending on the testers. A solution would be to ask confirmation from the end user, but this would lengthen the stage. Another solution would be to offer a palette of symbols or shapes with different colors, thus eliminating ambiguity and allowing automated sorting of reviews by shape (*e.g.*, only triangles indicating a danger) or by color (*e.g.*, red reviews first). This palette should be displayable on demand. For example, frequently observed gestures, such as a cross, a 'plus' sign, a 'minus' sign, a deletion stroke, and a circle, could become predefined icons.
- *No predefined types of reviews*: on the one hand, developers and users appreciate freely annotating a frozen screen; on the other hand, both are requesting a kind of "review template", a predefined type of review which could be applied directly on a portion of the screen. Instead of annotating, the end user would drag a predefined "review template" from a palette and drop it into the interface region of concern. For example, a comment "I do not understand

this region" could be dropped in a region, a template "this region should be moved there" could be offered by designating the source and target regions. In this way, developers would be able to easily aggregate comments by template or by type, instead of having them all at once in an unstructured way.

- *No text recognition*: writing a textual comment with fingers on the screenshot is painful, writing down a comment in an edit field is preferable. In addition, performing gesture-to-text translation is challenging as many writers express the same text differently. For the textual comments written as an article in plain text, the job should be straightforward by automated sentiment analysis [43]. Martens *et al.* [40] analyzed more than seven million application reviews on the Apple AppStore with respect to their emotional feelings to gain insight into the nature of feelings in application reviews. In this way, positive comments could be distinguished from negative comments sorted by UI element and recognized. We did not apply text mining as in [8]. Automatic text recognition would also enable differentiating fake reviews from genuine reviews, a crucial problem [41].
- *No interface region designation*: while the end user could designate any region, any object on the interface by gesture on the screenshot, developers would appreciate incorporating a system that automatically detects the interface region and its corresponding code to speed up maintenance. For example, all reviews received for the "Hamburger" menu could be classified into this category by automatic detection of its icon.
- *No version management for software evolution*: while SnappView captures the version of the application under review, nothing shows the evolution of the application between versions. For example, a series of negative reviews for a given version should be linked to an area of a new version of the application that solves the negative reviews. In this way, testers should be able to follow the evolution of the software and appreciate whether their reviews have been effectively taken into account.

6 CONCLUSION AND FUTURE WORK

This paper presents SnappView, an open-source software development kit facilitating the end-user review of graphical user interfaces for mobile applications and streamlining their input into a continuous design life cycle. SnappView structures this user interface review process into four cumulative stages: (1) account management and project preparation, (2) application distribution and preparation, (3) feedback creation, and (4) feedback handling. The evaluation of this process on "WeBoth", a real application developed for personal use on mobile devices, suggests that end users are keen to adhere to this review process, internally and externally, as SnappView supports four roles. Any user can switch from one role to another, being the creator of one application and the reviewer for another. A potentially interesting avenue to this work would be to perform a scalable bootstrapping [6], a process consisting of developing complex software systems by using simpler software, namely, by automatic self-improvement or by manually using the software itself. In our case, SnappView could be used to evaluate itself and see how does it evolve over time: based on feedback collected on SnappView by using SnappView itself, we could come up with a series of recommendations issued by SnappView users themselves. Our next step is to make SnappView evolving towards supporting more dynamic aspects throughout the reviewing process. For now, a single screenshot is reviewed at a time; several screenshots could be also captured, but are treated as individual elements. Instead, a suite of screenshots representing a part of an interactive session could be captured, either as a series of screenshots taken one after another, or as an animated image or a video, and reviewed as a whole. In this way, testes could better review the dynamic aspects across screenshots in the same series instead of reviewing individual ones.

ACKNOWLEDGMENTS

The authors of this paper are very grateful to the anonymous EICS (meta)reviewers for their constructive comments on earlier versions of this manuscript. They also thank the participants involved in the studies reported in this paper. Arthur Sluÿters is funded by the "Fonds de la Recherche Scientifique - FNRS" under Grant no. 40001931.

REFERENCES

[1] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. 2012. Using GUI ripping for automated testing of Android applications. In *Proceedings of the 27th IEEE-ACM International Conference on Automated Software Engineering (ASE '12)*. 258–261. https://doi.org/10.1145/2351676.2351717

[2] Nathalie Aquino, Jean Vanderdonckt, Nelly Condori-Fernández, Óscar Dieste, and Óscar Pastor. 2010. Usability Evaluation of Multi-Device/Platform User Interfaces Generated by Model-Driven Engineering. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10)*. Association for Computing Machinery, New York, NY, USA, Article 30, 10 pages. https://doi.org/10.1145/1852786.1852826

[3] J. M. Christian Bastien and Dominique L. Scapin. 1995. Evaluating a user interface with ergonomic criteria. *International Journal of Human-Computer Interaction* 7, 2 (1995), 105–121. https://doi.org/10.1080/10447319509526114

[4] Andrew Begel and Thomas Zimmermann. 2014. Analyze This! 145 Questions for Data Scientists in Software Engineering. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 12–23. https://doi.org/10.1145/2568225.2568233

[5] Enrico Bertini, Tiziana Catarci, Alan Dix, Silvia Gabrielli, Stephen Kimani, and Giuseppe Santucci. 2009. Appropriating Heuristic Evaluation for Mobile Computing. *International Journal of Mobile Human Computer Interaction* 1, 1 (2009), 20–41. https://doi.org/110.4018/jmhci.2009010102

[6] Peter Birsinger, Richard Xia, and Armando Fox. 2013. Scalable bootstrapping for python. In *Proc. of 22nd ACM International Conference on Information and Knowledge Management, San Francisco, CA, USA, October 27 - November 1, 2013 (CIKM '13)*, Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi (Eds.). 2441–2446. https://doi.org/10.1145/2505515.2505630

[7] Elodie Bouzekri. 2018. Model-Based Approach to Design and Develop Usable and Dependable Recommender Systems. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2018, Paris, France, June 19-22, 2018*. ACM, 17:1–17:7. https://doi.org/10.1145/3220134.3220147

[8] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 767–778. https://doi.org/10.1145/2568225.2568263

[9] Lin Cheng, Jialiang Chang, Zijiang Yang, and Chao Wang. 2016. GUICat: GUI Testing As a Service. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE '16)*. Association for Computing Machinery, New York, NY, USA, 858–863. https://doi.org/10.1145/2970276.2970294

[10] Wontae Choi, George Necula, and Koushik Sen. 2013. Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning. In *Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA '13)*. Association for Computing Machinery, New York, NY, USA, 623–640. https://doi.org/10.1145/2509136.2509552

[11] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. 2008. End Users As Unwitting Software Developers. In *Proceedings of the 4th International Workshop on End-user Software Engineering (WEUSE '08)*. ACM, New York, NY, USA, 6–10. https://doi.org/10.1145/1370847.1370849

[12] Andrew Crossan, Roderick Murray-Smith, Stephen Brewster, and Bojan Musizza. 2011. *Instrumented Usability Analysis for Mobile Devices*. IGI Global, Hershey, PA, 1–19. https://doi.org/10.4018/978-1-60960-499-8.ch001

[13] Tao Dong, Elizabeth F. Churchill, and Jeffrey Nichols. 2016. Understanding the Challenges of Designing and Developing Multi-Device Experiences. In *Proceedings of the ACM Conference on Designing Interactive Systems (DIS '16)*. ACM, New York, NY, USA, 62–72. https://doi.org/10.1145/2901790.2901851

[14] Umer Farooq, Leon Welicki, and Dieter Zirkler. 2010. API Usability Peer Reviews: A Method for Evaluating the Usability of Application Programming Interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2327–2336. https://doi.org/10.1145/1753326.1753677

[15] Eureka Foong, Steven P. Dow, Brian P. Bailey, and Elizabeth M. Gerber. 2017. Online Feedback Exchange: A Framework for Understanding the Socio-Psychological Factors. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4454–4467. https://doi.org/10.1145/3025453.3025791

[16] Richard Gomer. 2012. Eliciting Evaluative Comments from Users in Web 2.0 Scenarios. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI '12)*. ACM, New York, NY, USA, 367–370. https:

//doi.org/10.1145/2166966.2167050

[17] Guillaume Gronier and Laurence Johannsen. 2022. Proposition d'une adaptation française et premières validations de l'échelle d'utilisabilitÃĺ Computer System Usability Questionnaire (F-CSUQ). In *Proceedings of the 33th International French-Speaking Conference on Human-Computer Interaction (IHM '22)*, Bruno Dumas, Gilles Bailly, and Jean Vanderdonckt (Eds.). 1–17. http://doi.acm.org/10.1145/200952.12371

[18] Emitza Guzman and Walid Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014*, Tony Gorschek and Robyn R. Lutz (Eds.). IEEE Computer Society, 153–162. https://doi.org/10.1109/RE.2014.6912257

[19] Emitza Guzman, Luís Oliveira, Yves Steiner, Laura C. Wagner, and Martin Glinz. 2018. User feedback in the app store: a cross-cultural study. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Society, ICSE (SEIS) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Valérie Issarny and Schahram Dustdar (Eds.). ACM, 13–22. https://doi.org/10.1145/3183428.3183436

[20] Rachel Harrison, Derek Flood, and David Duce. 2013. Usability of mobile applications: literature review and rationale for a new usability model. *Journal of Interaction Science* 1, 1 (07 May 2013), 1. https://doi.org/10.1186/2194-0827-1-1

[21] Leonard Hoon, Milica Stojmenović, Raj Vasa, and Graham Farrell. 2016. Spreading Word: Author Frequency of App User Reviews. In *Proceedings of the 28th Australian Conference on Computer-Human Interaction (OzCHI '16)*. ACM, New York, NY, USA, 643–645. https://doi.org/10.1145/3010915.3011850

[22] Cuixiong Hu and Iulian Neamtiu. 2011. A GUI Bug Finding Framework for Android Applications. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11)*. Association for Computing Machinery, New York, NY, USA, 1490âĂŞ1491. https://doi.org/10.1145/1982185.1982504

[23] Jeff Huang, Oren Etzioni, Luke Zettlemoyer, Kevin Clark, and Christian Lee. 2012. RevMiner: An Extractive Interface for Navigating Reviews on a Smartphone. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 3–12. https://doi.org/10.1145/2380116.2380120

[24] Maria Husmann, Nicola Marcacci Rossi, and Moira C. Norrie. 2016. Usage Analysis of Cross-device Web Applications. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays (PerDis '16)*. ACM, New York, NY, USA, 212–219. https://doi.org/10.1145/2914920.2915017

[25] Melody Y. Ivory and Marti A Hearst. 2001. The State of the Art in Automating Usability Evaluation of User Interfaces. *ACM Comput. Surv.* 33, 4 (Dec. 2001), 470–516. https://doi.org/10.1145/503112.503114

[26] Nishant Jha and Anas Mahmoud. 2018. Using Frame Semantics for Classifying and Summarizing Application Store Reviews. *Empirical Softw. Engg.* 23, 6 (Dec. 2018), 3734âĂŞ3767. https://doi.org/10.1007/s10664-018-9605-x

[27] Haojian Jin, Tetsuya Sakai, and Koji Yatani. 2014. ReviewCollage: A Mobile Interface for Direct Comparison Using Online Reviews. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services (MobileHCI '14)*. ACM, New York, NY, USA, 349–358. https://doi.org/10.1145/2628363.2628373

[28] Ron Kohavi and Roger Longbotham. 2017. *Online Controlled Experiments and A/B Testing*. Springer US, Boston, MA, 922–929. https://doi.org/10.1007/978-1-4899-7687-1_891

[29] P. Leach, M. Mealling, and R. Salz. 2005. RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace. Proposed standard. (July 2005). https://doi.org/10.17487/RFC4122 The Internet Engineering Task Force (IETF).

[30] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the ACM Int. Conf. on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1âĂŞ17. https://doi.org/10.1145/3173574.3173610

[31] James R. Lewis. 1995. IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of HumanâĂŞComputer Interaction* 7, 1 (1995), 57–78. https://doi.org/10.1080/10447319509526110 arXiv:https://doi.org/10.1080/10447319509526110

[32] James R. Lewis. 2002. Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies. *International Journal of HumanâĂŞComputer Interaction* 14, 3-4 (2002), 463–488. https://doi.org/10.1080/10447318.2002.9669130 arXiv:https://doi.org/10.1080/10447318.2002.9669130

[33] James R. Lewis. 2006. Sample Sizes for Usability Tests: Mostly Math, Not Magic. *interactions* 13, 6 (Nov. 2006), 29–33. https://doi.org/10.1145/1167948.1167973

[34] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology* 22, 140 (1932), 55–. http://psycnet.apa.org/record/1933-01885-001

[35] Adriana Lopes, Anna Beatriz Marques, Simone Diniz Junqueira Barbosa, and Tayana Conte. 2015. Evaluating HCI Design with Interaction Modeling and Mockups - A Case Study. In *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 3, Barcelona, Spain, 27-30 April, 2015*, Slimane Hammoudi, Leszek A. Maciaszek, and Ernest Teniente (Eds.). SciTePress, 79–87. https://doi.org/10.5220/0005374200790087

[36] Kurt Luther, Amy Pavel, Wei Wu, Jari-lee Tolentino, Maneesh Agrawala, Björn Hartmann, and Steven P. Dow. 2014. CrowdCrit: Crowdsourcing and Aggregating Visual Design Critique. In *Proceedings of the Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW Companion '14)*. ACM,

New York, NY, USA, 21–24. https://doi.org/10.1145/2556420.2556788

[37] Wendy E. Mackay. 2004. The Interactive Thread: Exploring Methods for Multi-disciplinary Design. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS '04)*. ACM, New York, NY, USA, 103–112. https://doi.org/10.1145/1013115.1013131

[38] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2021. Two-Dimensional Stroke Gesture Recognition: A Survey. *ACM Comput. Surv.* 54, 7, Article 155 (July 2021), 36 pages. https://doi.org/10.1145/3465400

[39] Leonardo Marques, Patrícia Matsubara, Walter Nakamura, Igor Wiese, Luciana Zaina, and Tayana Conte. 2019. UX-Tips: A UX Evaluation Technique to Support the Identification of Software Application Problems. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES 2019)*. Association for Computing Machinery, New York, NY, USA, 224âĂŞ233. https://doi.org/10.1145/3350768.3350783

[40] Daniel Martens and Timo Johann. 2017. On the Emotion of Users in App Reviews. In *Proc. of IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion '17)*. 8–14. https://doi.org/10.1109/SEmotion.2017.6

[41] Rami Mohawesh, Shuxiang Xu, Son N. Tran, Robert Ollington, Matthew Springer, Yaser Jararweh, and Sumbal Maqsood. 2021. Fake Reviews Detection: A Survey. *IEEE Access* 9 (2021), 65771–65802. https://doi.org/10.1109/ACCESS.2021.3075573

[42] Thanh-Diane Nguyen, Jean Vanderdonckt, and Ahmed Seffah. 2016. Generative Patterns for Designing Multiple User Interfaces. In *Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft '16)*. ACM, New York, NY, USA, 151–159. https://doi.org/10.1145/2897073.2897084

[43] Martin Obaidi and Jil Klünder. 2021. Development and Application of Sentiment Analysis Tools in Software Engineering: A Systematic Literature Review. In *Evaluation and Assessment in Software Engineering (EASE 2021)*. Association for Computing Machinery, New York, NY, USA, 80âĂŞ89. https://doi.org/10.1145/3463274.3463328

[44] Jonas Oppenlaender, Thanassis Tiropanis, and Simo Hosio. 2020. CrowdUI: Supporting Web Design with the Crowd. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 76 (June 2020), 28 pages. https://doi.org/10.1145/3394978

[45] Judith Alice Redi, Tobias Hoßfeld, Pavel Korshunov, Filippo Mazza, Isabel Povoa, and Christian Keimel. 2013. Crowdsourcing-based Multimedia Subjective Evaluations: A Case Study on Image Recognizability and Aesthetic Appeal. In *Proceedings of the 2Nd ACM International Workshop on Crowdsourcing for Multimedia (CrowdMM '13)*. ACM, New York, NY, USA, 29–34. https://doi.org/10.1145/2506364.2506368

[46] Susan Rose, Moira Clark, Phillip Samouel, and Neil Hair. 2012. Online Customer Experience in e-Retailing: An empirical model of Antecedents and Outcomes. *Journal of Retailing* 88, 2 (2012), 308–322. https://doi.org/10.1016/j.jretai.2012.03.001

[47] Jeff Sauro. 2010. *A practical guide to measuring usability: 72 answers to the most common questions about quantifying the usability of websites and software*. Measuring Usability LCC.

[48] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. 2017. Construction of a Benchmark for the User Experience Questionnaire (UEQ). *Int. J. Interact. Multim. Artif. Intell.* 4, 4 (2017), 40–44. https://doi.org/10.9781/ijimai.2017.445

[49] Martin Schrepp and Jörg Thomaschewski. 2019. Design and Validation of a Framework for the Creation of User Experience Questionnaires. *International Journal of Interactive Multimedia and Artificial Intelligence* 5, 7 (2019), 88–95. https://doi.org/10.9781/ijimai.2019.06.006

[50] Tavita Su'a, Sherlock A. Licorish, Bastin Tony Roy Savarimuthu, and Tobias Langlotz. 2017. QuickReview: A Novel Data-Driven Mobile User Interface for Reporting Problematic App Features. In *Proceedings of the 22Nd ACM International Conference on Intelligent User Interfaces (IUI '17)*. ACM, New York, NY, USA, 517–522. https://doi.org/10.1145/3025171.3025178

[51] Craig J. Sutherland, Andrew Luxton-Reilly, and Beryl Plimmer. 2016. Freeform digital ink annotations in electronic documents: A systematic mapping study. *Comput. Graph.* 55 (2016), 1–20. https://doi.org/10.1016/j.cag.2015.10.014

[52] Craig J. Sutherland, Andrew Luxton-Reilly, and Beryl Plimmer. 2016. Who changed my annotation? An investigation into refitting freeform ink annotations. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2016, Cambridge, United Kingdom, September 4-8, 2016*, Alan F. Blackwell, Beryl Plimmer, and Gem Stapleton (Eds.). IEEE Computer Society, 12–20. https://doi.org/10.1109/VLHCC.2016.7739658

[53] T. Takala, M. Katara, and J. Harty. 2011. Experiences of System-Level Model-Based GUI Testing of an Android Application. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. 377–386. https://doi.org/10.1109/ICST.2011.11

[54] Jean Vanderdonckt, Mathieu Zen, and Radu-Daniel Vatavu. 2019. AB4Web: An On-Line A/B Tester for Comparing User Interface Design Alternatives. *Proc. ACM Hum. Comput. Interact.* 3, EICS (2019), 18:1–18:28. https://doi.org/10.1145/3331160

[55] Arnold P. O. S. Vermeeren, Effie Lai-Chong Law, Virpi Roto, Marianna Obrist, Jettie Hoonhout, and Kaisa Väänänen-Vainio-Mattila. 2010. User Experience Evaluation Methods: Current State and Development Needs. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries (NordiCHI '10)*. Association for

Computing Machinery, New York, NY, USA, 521âĂŞ530. https://doi.org/10.1145/1868914.1868973

[56] Felix von Reischach, Erica Dubach, Florian Michahelles, and Albrecht Schmidt. 2010. An Evaluation of Product Review Modalities for Mobile Phones. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '10)*. ACM, New York, NY, USA, 199–208. https://doi.org/10.1145/1851600.1851635

[57] Andy Whitefield, FRANK Wilson, and John Dowell. 1991. A framework for human factors evaluation. *Behaviour & Information Technology* 10, 1 (1991), 65–79. https://doi.org/10.1080/01449299108924272 arXiv:https://doi.org/10.1080/01449299108924272

[58] Anbang Xu, Shih-Wen Huang, and Brian Bailey. 2014. Voyant: Generating Structured Feedback on Visual Designs Using a Crowd of Non-experts. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*. ACM, New York, NY, USA, 1433–1444. https://doi.org/10.1145/2531602.2531604

[59] Anbang Xu, Huaming Rao, Steven P. Dow, and Brian P. Bailey. 2015. A Classroom Study of Using Crowd Feedback in the Iterative Design Process. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '15)*. ACM, New York, NY, USA, 1637–1648. https://doi.org/10.1145/2675133.2675140

[60] Koji Yatani, Michael Novati, Andrew Trusty, and Khai N. Truong. 2011. Review Spotlight: A User Interface for Summarizing User-generated Reviews Using Adjective-noun Word Pairs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 1541–1550. https://doi.org/10.1145/1978942.1979167

[61] Alvin Yuan, Kurt Luther, Markus Krause, Sophie Isabel Vennix, Steven P Dow, and Bjorn Hartmann. 2016. Almost an Expert: The Effects of Rubrics and Expertise on Perceived Value of Crowdsourced Design Critiques. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW '16)*. ACM, New York, NY, USA, 1005–1017. https://doi.org/10.1145/2818048.2819953

[62] Mathieu Zen and Jean Vanderdonckt. 2016. Assessing User Interface Aesthetics Based on the Inter-Subjectivity of Judgment. In *HCI 2016 - Fusion! Proceedings of the 30th International BCS Human Computer Interaction Conference, BCS HCI 2016, Bournemouth University, Poole, UK, 11-15 July 2016 (Workshops in Computing)*, Shamal Faily, Nan Jiang, Huseyin Dogan, and Jacqui Taylor (Eds.). BCS. http://ewic.bcs.org/content/ConWebDoc/56903

[63] Li Lyna Zhang, Chieh-Jan Mike Liang, Wei Zhang, and Enhong Chen. 2017. Towards A Contextual and Scalable Automated-Testing Service for Mobile Apps. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications (HotMobile '17)*. Association for Computing Machinery, New York, NY, USA, 97âĂŞ102. https://doi.org/10.1145/3032970.3032972

A DEVELOPMENT INSTRUCTIONS

To incorporate SnappView into the Xcode of a mobile iOS application, the following steps are required, which can be partially automated via a script:

- Save the files libSnappview.a, snappview.h, and snappviewRe-sources.bundle in the Xcode project
- Add the libraries libSnappview.a in the Xcode build phases: "link binary with libraries" and snappviewResources.bundle in the Xcode build phases: "copy bundle resources"
- Add the required frameworks in the Xcode build phases: "link binary with libraries": Foundation.framework, CoreGraphics.framework, UIKit.framework, CoreData.framework, MobileCoreServices.framework, SystemConfiguration.framework, and libz.dylib
- Include in the main application delegate file the public API header file of snappview:

```
#import "snappview.h"   // Insert SnappView header
```

- In the application delegate file, find the function application didFin-ishLaunchingWithOptions and call SnappView to enable the SnappView agent to listen to shake (UIAcceleration) events:

```
(BOOL)  application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
        [[snappview sharedInstance] startWithAppKey:@"YOURKEY"
        trigger:@"shake"]; // Mandatory
        // Replace YOURKEY with the project's key generated in the dashboard
        // Example of key: u9p1.5120609547f790.46918612
        // If you need to separate the feedback of the different versions of your app,
        // use a different key for each version
        // If you need to disable the shake recognition, set trigger to @"none"
        // See below to find out how to manually trigger SnappView without using
        // shakes recognition
}
```

- Optional: to instantly display the SnappView menu wherever needed in the application or for testing it in Apple's simulator, call the facultative trigger function, which requires to import snappview.h in the corresponding file:

```
[[snappview sharedInstance] trigger]; // Facultative
// Will be working only if([[snappview sharedInstance]isTriggerable]==TRUE)
// (use setTriggerable function if you previously set it to FALSE)
// You should have previously called
// [[snappview sharedInstance]startWithAppKey: @"YOURKEY"
// trigger:@"shake"]; // or trigger:@"none"];
```

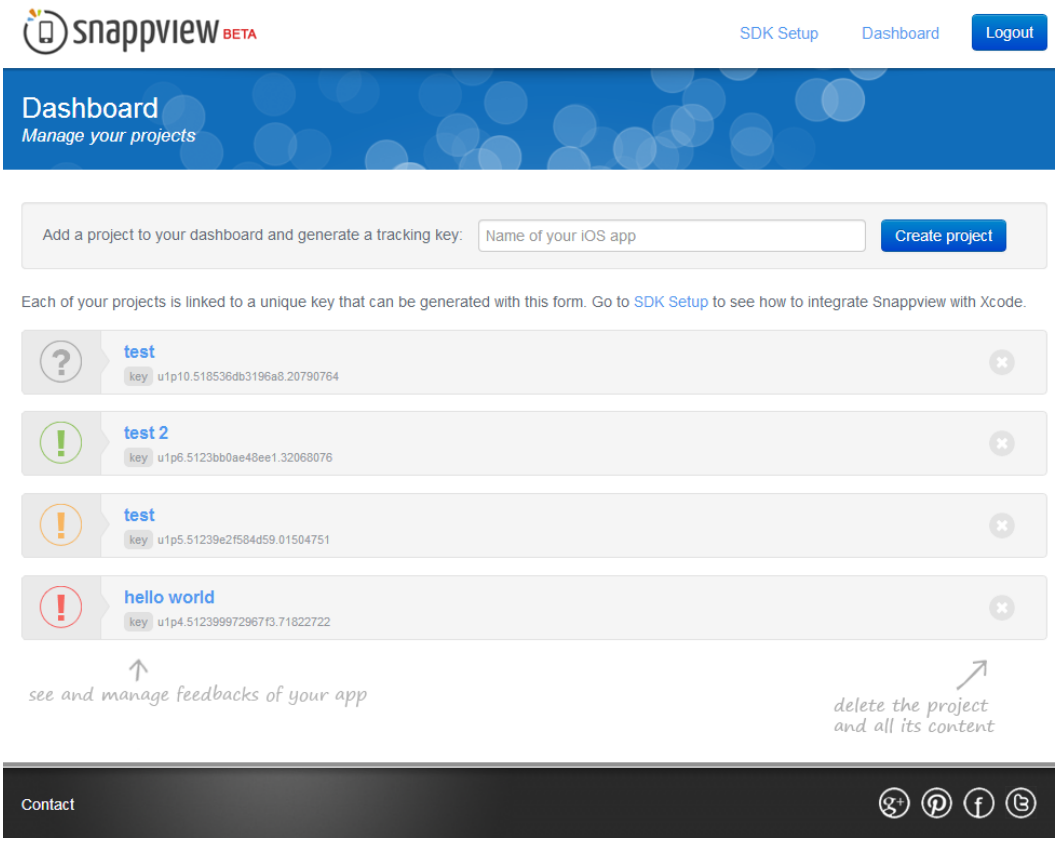

Fig. 17. SnappView dashboard of tested applications.

B SNAPPVIEW SOFTWARE ARCHITECTURE

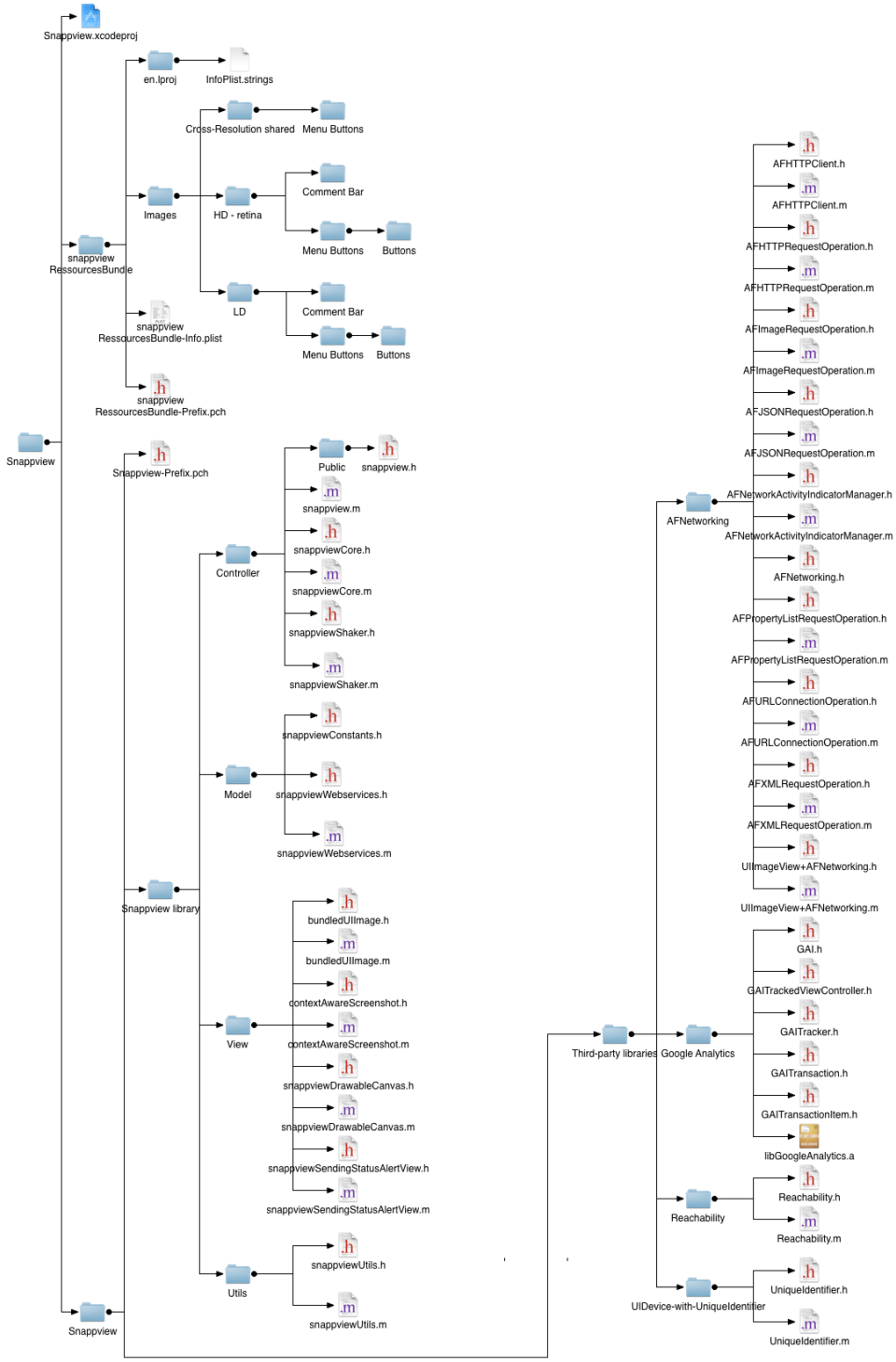

Fig. 18. Package architecture of SnappView.

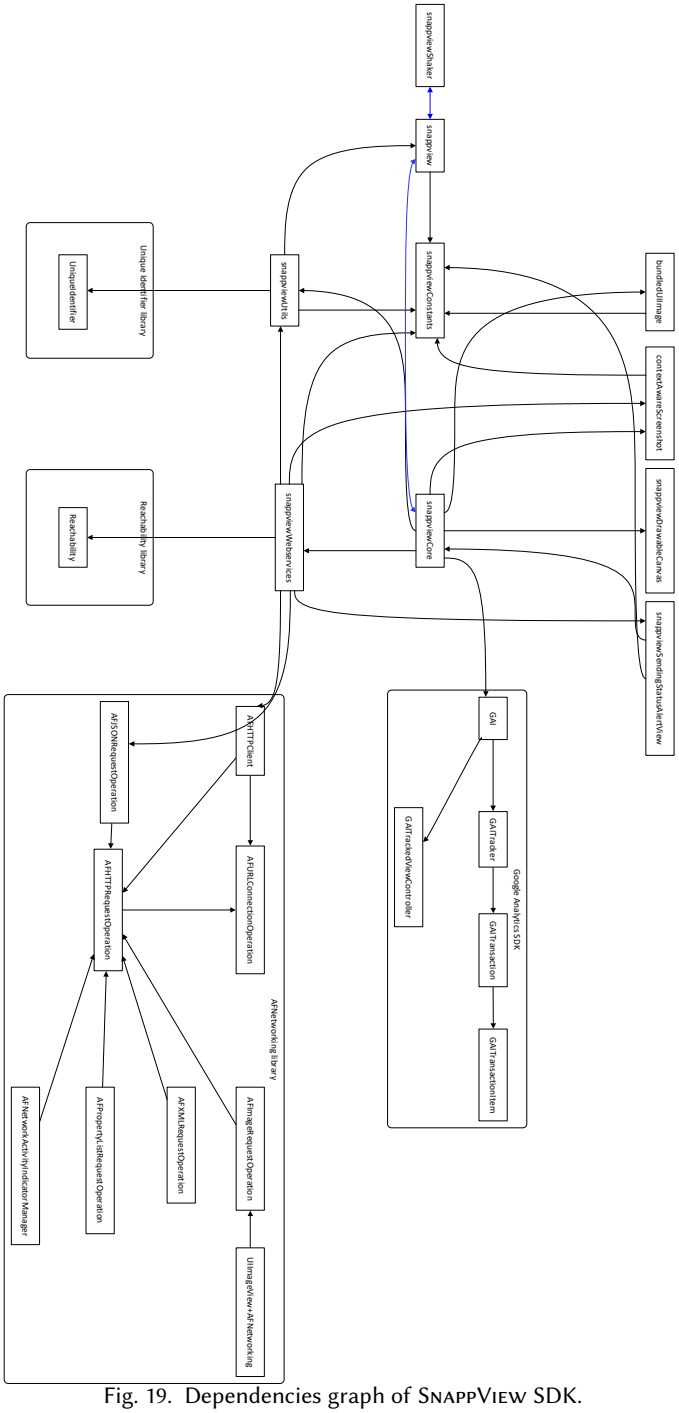

Fig. 19. Dependencies graph of SnappView SDK.

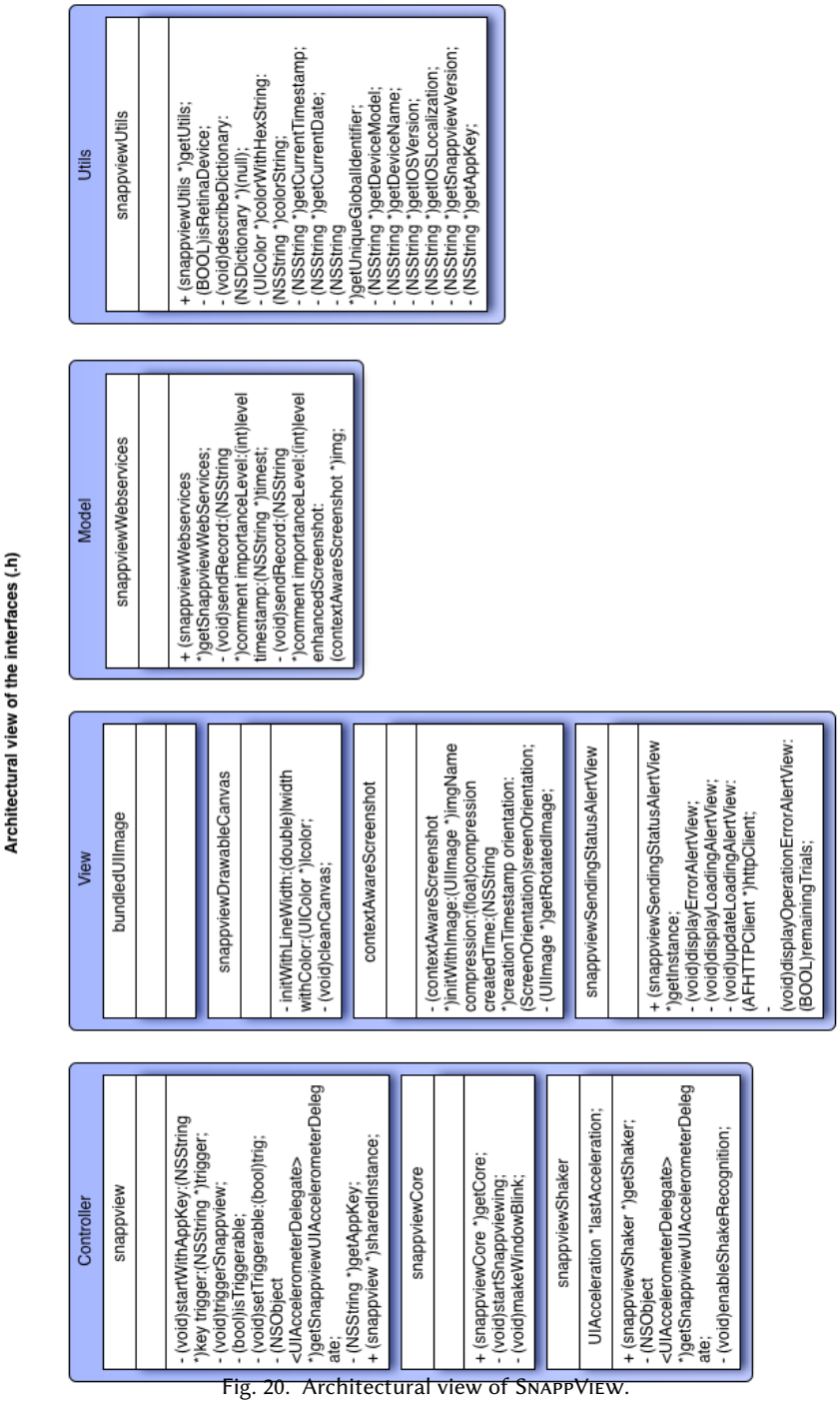

Fig. 20. Architectural view of SnappView.

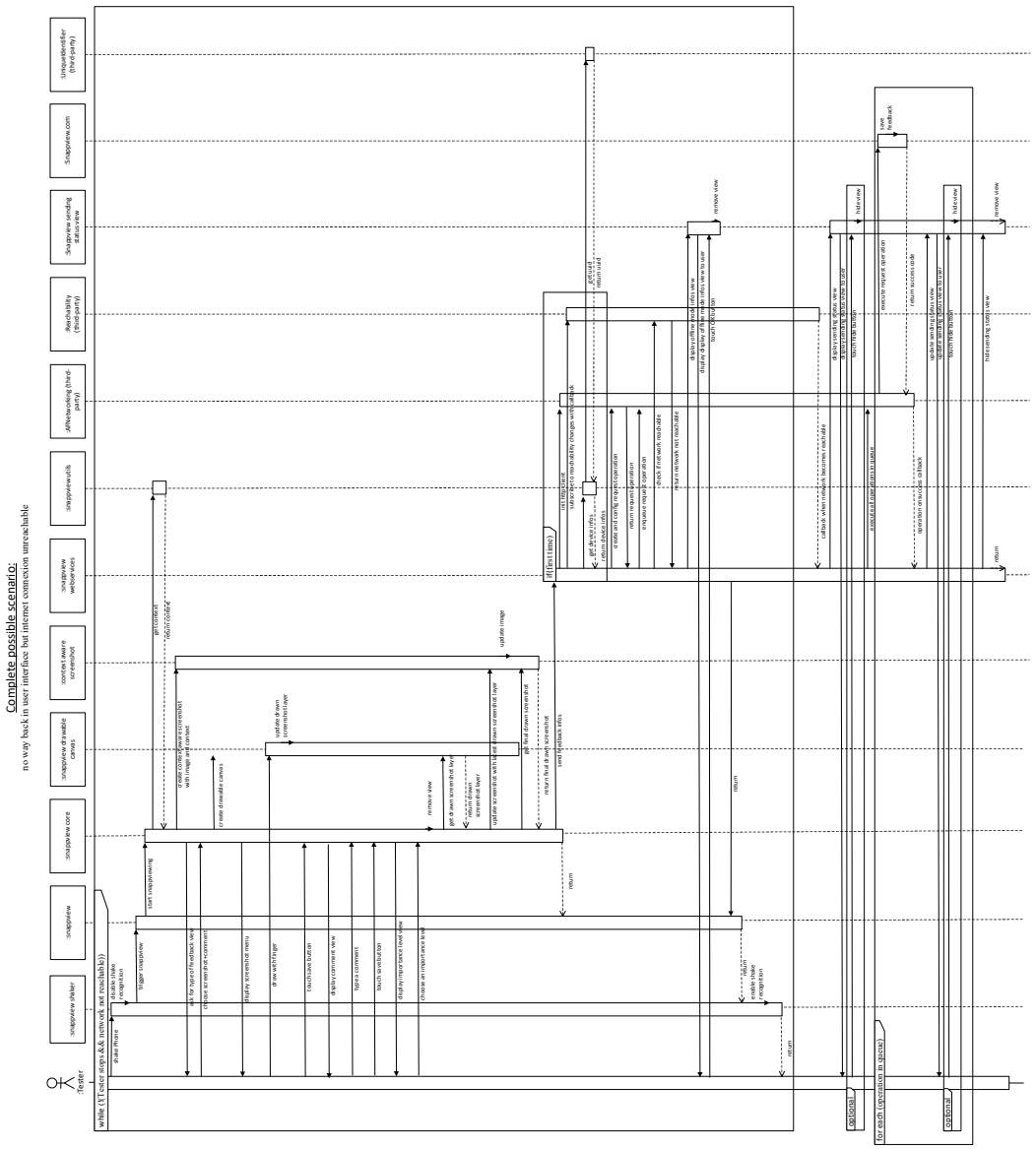

Fig. 21. "Complete scenario" activity diagram.

Received February 2022; accepted April 2022

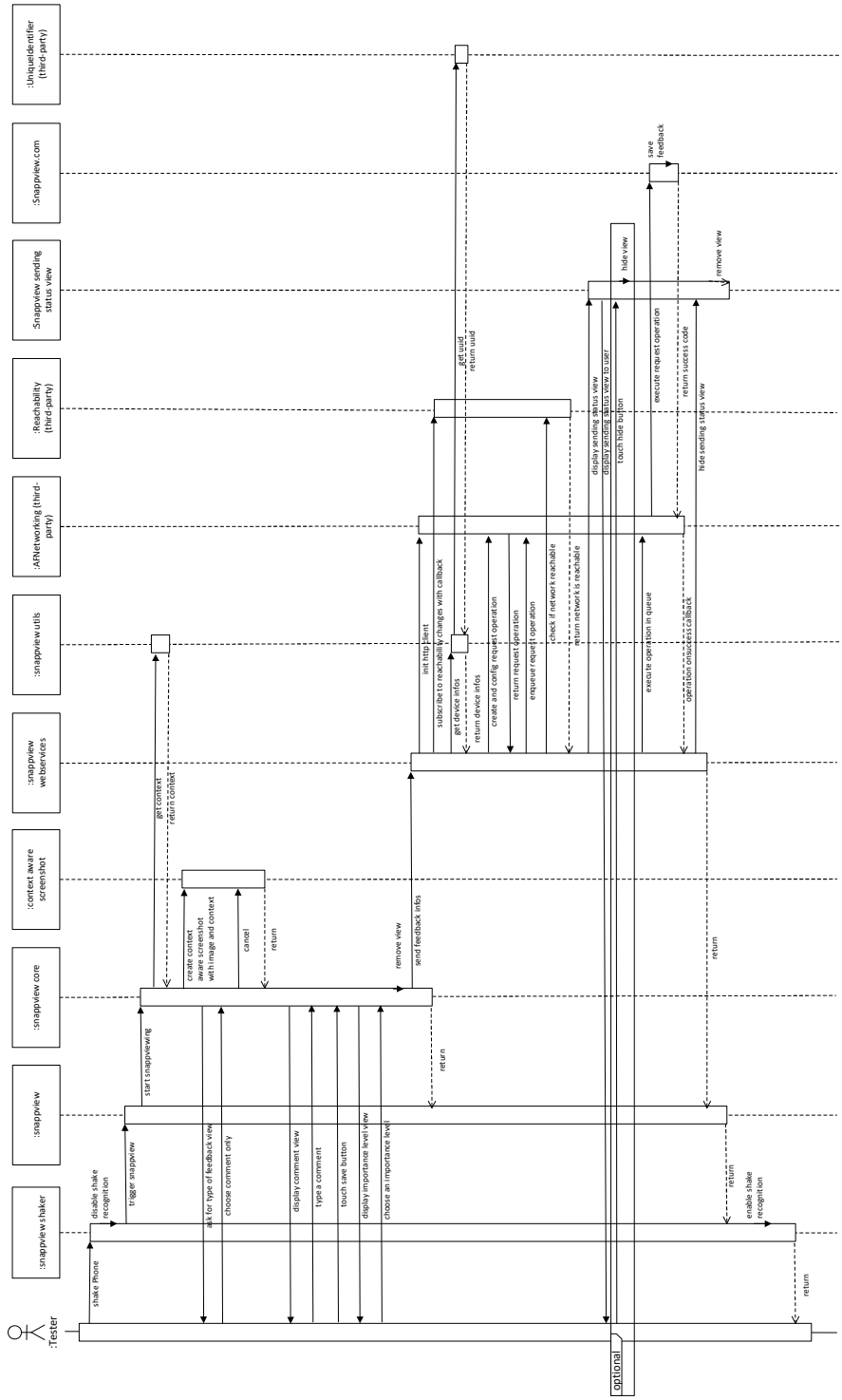

Fig. 22. "Comment only" activity diagram.

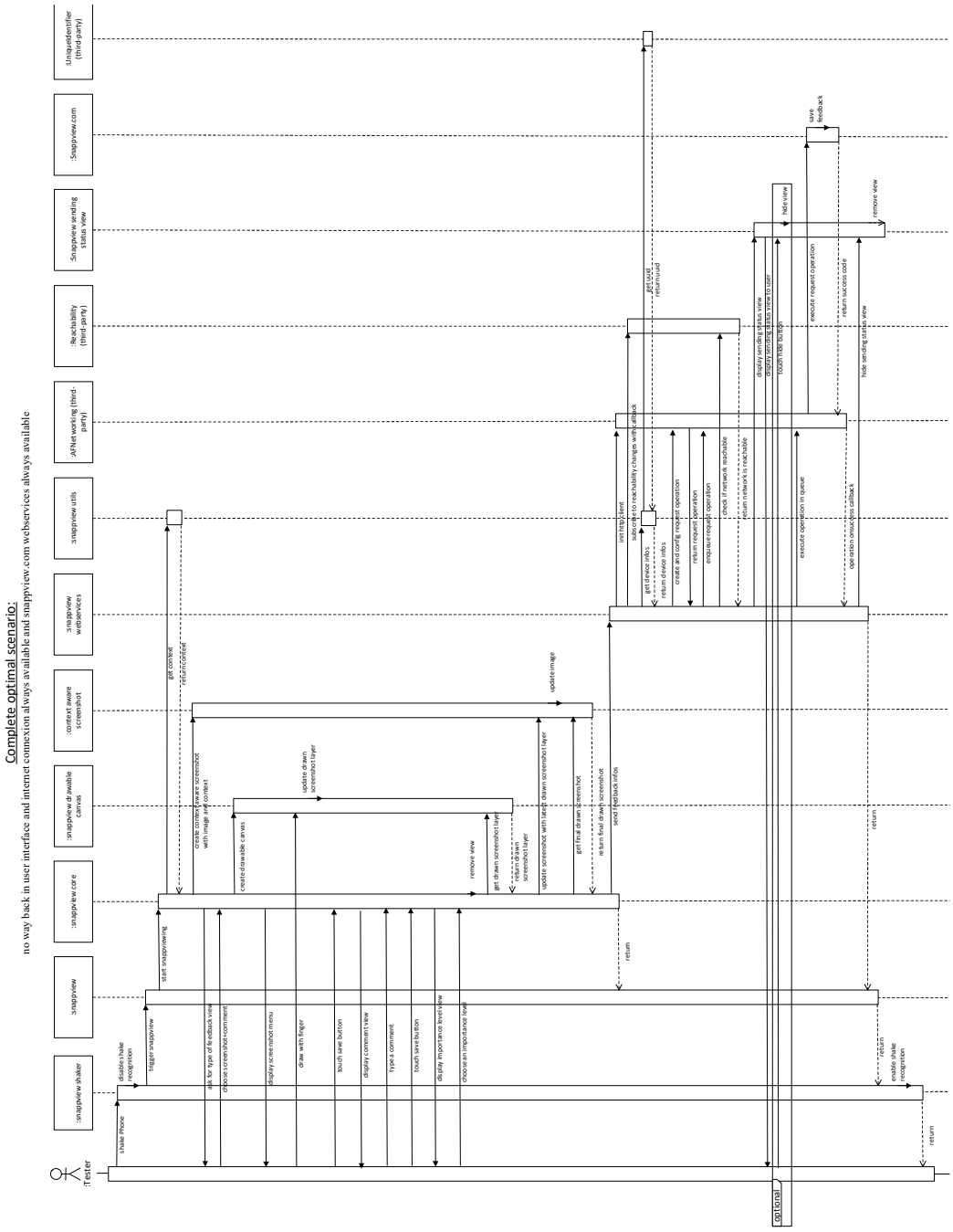

Fig. 23. "Normal scenario" activity diagram.

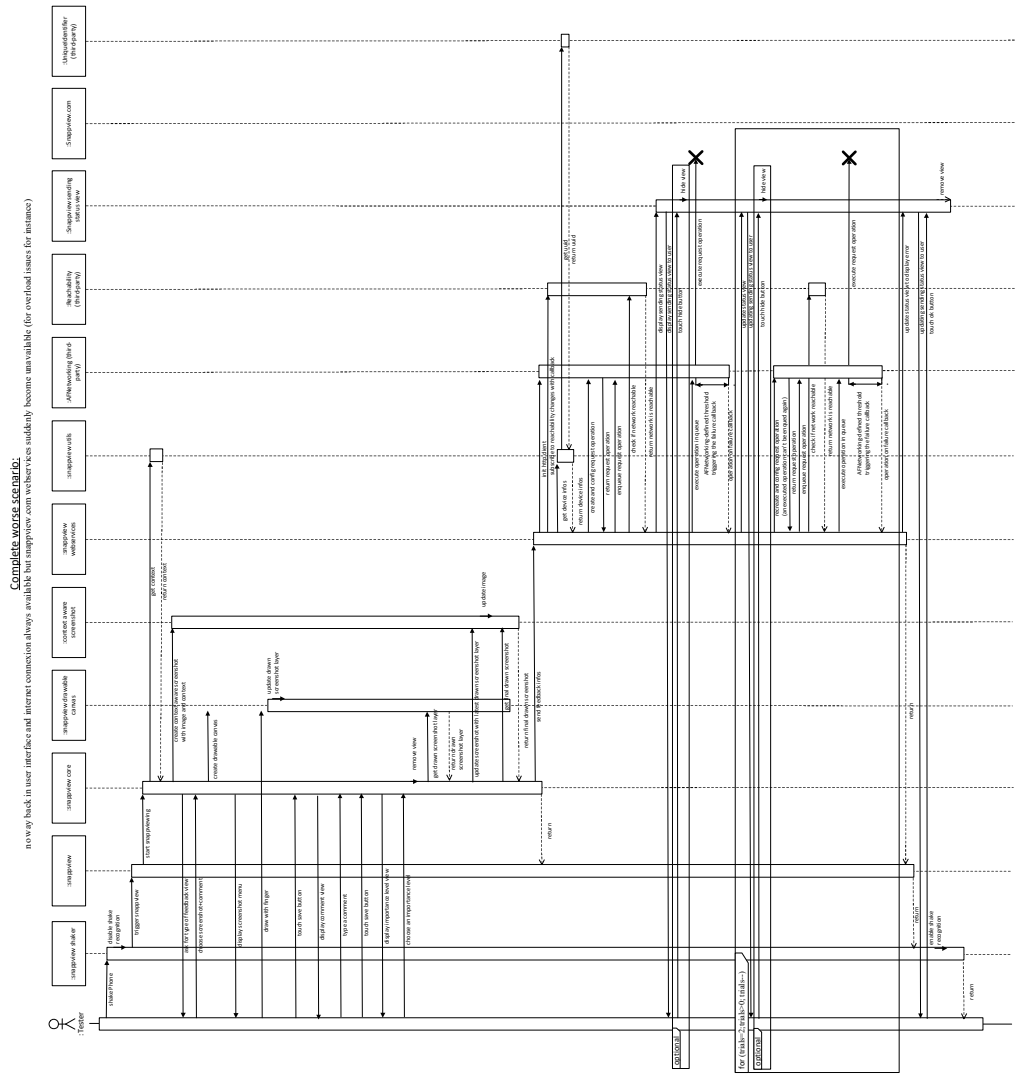

Fig. 24. "Worse scenario" activity diagram.

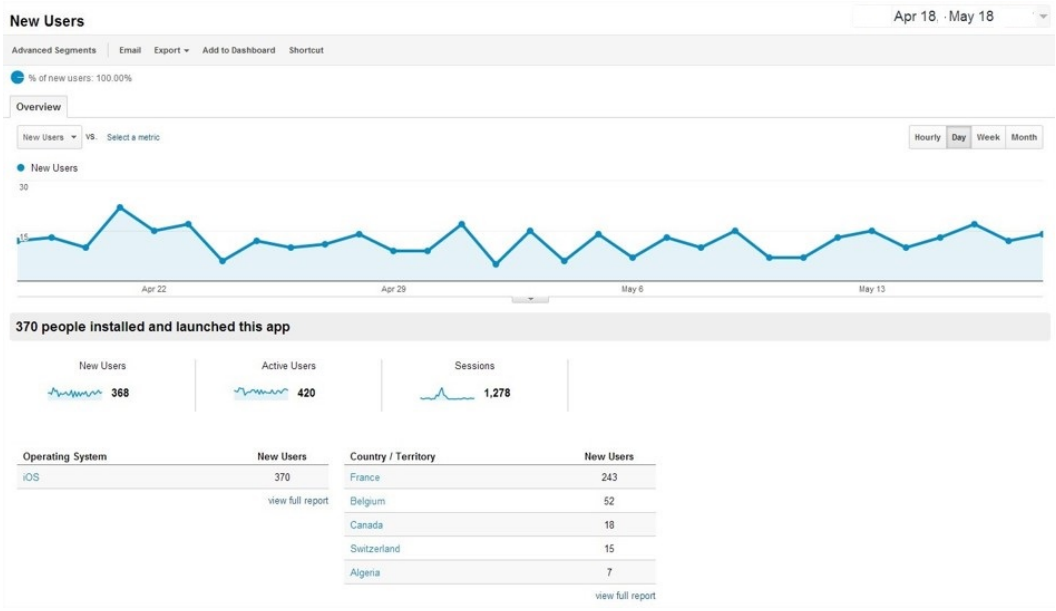

Fig. 25. New users of the tested application during reviewing period.

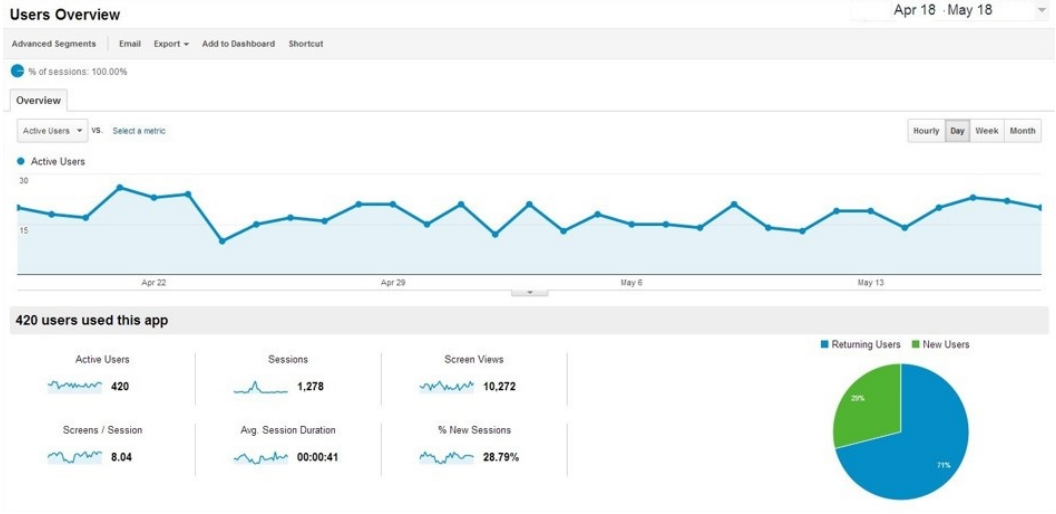

Fig. 26. Overview of users of the tested application during reviewing period.