Extending Manual GUI Testing Beyond Defects by Building Mental Models of Software Behavior

Emily Kowalczyk
University of Maryland, College Park
Department of Computer Science
College Park, MD 20742
emilyk@cs.umd.edu

Atif Memon
University of Maryland, College Park
Department of Computer Science
College Park, MD 20742
atif@cs.umd.edu

ABSTRACT

Manual GUI testing involves providing inputs to the software via its GUI and determining the software's correctness using its outputs, one of them being the GUI itself. Because of its human-in-the-loop nature, GUI testing is known to be a time-consuming activity. In practice, it is done by junior, inexpensive testers to keep costs low at the very tail-end of the software development process. In this paper, we posit that the importance of GUI testing has suffered due to its traditional narrow role – to detect residual software defects. Because of its human-in-the-loop nature, GUI testing has the potential to provide outputs other than defects and to be used as inputs to several downstream activities, e.g., security analysis. One such output is the mental model that the GUI tester creates during testing; a model that implicitly informs the tester of the software designer's intent. To evaluate our claim, we consider an important question used for security assessment of Android apps: "What permission-sensitive behaviors does this app exhibit?" Our assessment is based on the comparison of 2 mental models of 12 Android apps – one derived from the app's usage and the other from its public description. We compare these two models with a third, automatically derived model – the permissions the app seeks from the Android OS. Our results show that the usage-based model provides unique insights into app behavior. This model may be an important outcome of GUI testing, and its consistency with other behavioral information about the app could later be used in software quality assurance activities such as security assessment.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Software Testing

Keywords

GUI Testing, Mobile Security, Human Factors, User Study

1. INTRODUCTION

System testing, i.e., testing a software system as a whole, of a GUI-based software necessarily involves supplying test inputs via the GUI and observing the software's correctness through the GUI [7, 11, 23, 27, 38]. In practice, a large portion of system testing is done manually, making it an expensive activity [24]. Most organizations employ junior testers, interns, and "trainees" to perform this type of testing at the tail end of the development process in order to keep their costs low [2, 35]. In doing so, they fail to fully harness the full potential of manual GUI testing.

In this paper, we posit that the human-in-the-loop nature of manual GUI testing may be exploited to broaden its scope beyond defect detection. A human tester who goes about for hours and days testing the software, clicking on buttons, opening windows, entering data, and examining output, creates an implicit mental model of the software under test. For example, while testing an automobile loan approval software application, the tester may create a mental model of the various parameters used by credit companies to determine eligibility for a loan. As another example, upon seeing a "camera" button in a mobile app, a tester may create a preliminary mental model that the app uses the device's camera, perhaps to take photos or video. The tester may then refine this preliminary model by additional interactions with the software. Such mental models may be useful for other software engineering activities, such as security assessment, especially if the tester's model of the software's behaviors is related to security-sensitive aspects of the software.

The very philosophy of GUI design lends itself to our new vision for manual GUI testing. GUIs are typically designed to help users form correct productive mental models [16, 19, 29]. These models are built upon prior knowledge, and GUI designers use this fact; relying on the familiarity of a user with an old, frequently used system gains user trust and helps accomplish tasks. Since recognition is better than recall, an effective GUI provides cues and visual elements to relieve the user from the memory load necessary to recall the functionality of the system [29, 34]. It is this design philosophy of GUIs that we now exploit to build mental models during GUI testing/usage and apply it to an aspect of software security analysis.

Our exemplar platform is the Android OS. Given an Android app, we have found that it is non-trivial to determine the type and nature of permission sensitive API calls that

it might make [8, 12, 18], and consequently, the type and set of device sensors (e.g., mic, camera) it might access. For example, it is not easy to answer this question: *Does the app invoke the phone's camera?* or in other words: *Does the app invoke a permission-sensitive API method that would lead to the app accessing the device camera?* These questions have clear implications for security – no one wants to install an app that might maliciously take pictures and send them to a network location. Current attempts to address these issues are largely based on static analysis of the app's code [6, 8, 20] or extraction of its requested permissions from its *AndroidManifest.xml* file [1, 10, 17, 28]. However, static analysis faces severe problems because Android apps typically import large code libraries without using them and are over-privileged [5, 12, 18, 30, 36].

We take a first step towards showing that the mental model from manual GUI testing may be used to develop a set of expected permissions an app may request. Specifically, we designed a survey to develop two different user mental models of an app's behavior based on differing perspectives: (1) on using an app (the use perspective) and (2) on reading an app's description (the description perspective). We then compared the outputs with a third perspective of an app's behavior, the app's permission requests in the AndroidManifest.xml file (the permission perspective). Our results indicate that while the models have many behaviors in common, each perspective still contains behaviors unsupported by the other perspectives. In fact, the model developed by app usage gives the most distinct perspective of the app's behavior with nearly 30% of its behaviors unsupported by the description or permission requests. We feel that our results may be used to encourage adding a new dimension to GUI testing, where GUI testers not only output a set of bugs, but also the mental model they developed during testing, one that describes what they expected the app to do.

In the next section we discuss related work and background. In Section 3, we present the design of our survey with results in Section 4. We conclude in Section 5 with a discussion of limitations and future work.

2. BACKGROUND & RELATED WORK

Mental models have received significant attention in fields such as psychology, cognitive science and human computer interaction. While there are many uses and purposes for mental models [29], in this paper we refer to them in the most general sense, an underlying understanding of a system's workings, i.e., the app and its functionality. Mental models have a long history in the field of human computer interaction where they have had use in such areas as user interface design and usability testing [25, 26, 32, 32]. In these areas they've typically been used to better understand how the user perceives the defined system and how to instruct the user or design to reduce user error.

Mental models and user expectation have also been incorporated into issues of app privacy. For example, Felt et al. [13] constructed a mental model of users' perception on different mobile privacy risks. They asked 3,115 smartphone users on 99 risks associated with 54 smartphone privileges. Each user was to rate how upset they would be if the risk was to occur and the results were used to rank privacy risks based on their importance to the surveyed users. Lin et al [22] used crowdsourcing to capture users' expectations of what sensitive resources mobile apps use. They then used this data to assess people's perceptions of whether a given action is legitimate for an app to do and how that action makes users feel with respect to privacy.

User comprehension of app permissions has also been studied. Felt et al. [14] evaluated whether Android users pay attention to understand and act on permission information during installation. They conducted two usability studies: an Internet survey of 308 Android users and a laboratory study where 25 Android users were interviewed and observed. Overall, they found users rarely paid attention to the permissions when installing and few comprehended what they meant. Similar conclusions were made by Kelley et al. [21] who conducted interviews with 25 users in two different cities and also found users did not pay attention to nor understand permissions.

Recently, significant attention has been given to expectation in app security. This research has primarily focused on defining expectation through *static* artifacts such as permissions and descriptions and has yet to incorporate a dimension based on use in their analysis. Qu et al. [28] examine the app's *Description* and *Permission* requests. Their analysis tries to determine whether text in the app description provides any indication for why the app needs a specific permission (they considered only 3 permissions: READ_CONTACTS, READ_CALENDAR, and RECORD_AUDIO).

In later work [33], they check for consistency between *Description* and *Permissions*, which they call description-to-permission fidelity. Their results show that the description-to-permissions fidelity is generally low on Google Play with only 9.1% of applications requesting permission sets that can be inferred from the description. They hypothesize that new or individual developers—those not from a software company—may fail to completely discuss the need for some permission in the app description.

Gorla et al. [17] make use of app *Description* and *API usage*. Their goal is to identify suspicious/outlier apps in respect to API usage. They first examine the description text to cluster descriptions into topics, and then use API calls to reclassify apps as normal or abnormal depending on whether their API usage falls in "normal" behavior, where normal is defined by other apps in the cluster.

Further, Huang et al. [20] compare text strings used in the user interface of an app with API calls to determine whether the API usage is consistent with user expectation; Fuchs et al. [15] extract security specifications from manifest files to check whether data flows through those applications are consistent with the specifications; and Peng et al. [31] use probabilistic models to assign each app a risk score based on its requested permissions.

Our research differs from the works mentioned above in that it focuses on the mental model developed by using the app, not the comprehension when reading the permissions requested or from static artifacts alone. Further, our research differs in its ultimate aim. We seek to construct models not

for usability testing, but understanding what the users expect the app to do and using this expectation to find apps behaving unexpectedly.

3. SURVEY DESIGN

We now describe in detail our study, which focused on determining whether mental models of an app's functionality constructed by using an app were distinct from those developed by other sources. Note the study's aim was not to determine the truth of such models but only that they were distinct from those generated using other sources and provided a unique perspective of the app. Specifically, we sought to answer the following research question:

RQ: Does the mental model built from *using* an app give unique information regarding an app's behavior that is not given in other commonly used specification sources?

To answer this question, we conducted a series of user studies where each tester was asked to complete a series of tasks and surveys. These responses were then compared against models constructed from other common sources of behavioral information such as an app's description and requested permissions. In the section that follows, we describe in detail the sample of apps evaluated and the users who participated. We then walk through a typical user study session and the evaluation methods used to develop and compare the constructed models.

3.1 Sample of Apps

Our testers's evaluated 12 randomly selected apps from the Google Play Store. These apps were selected from a database of 400,000 Play Store apps downloaded from Jan-Nov 2014. We sought a sample of diverse and commonly used apps. In order to better ensure this, two constraints were placed on the selection: 1) that the set of selected apps spanned a number of different Play Store categories (i.e., Games, Health, Productivity, etc) and 2) each had over 1 million downloads at the time of the study. Further details on the sample can be found in Table 1.

3.2 Testers

The testers evaluating the apps and descriptions consisted of 10 graduate and undergraduate students taking an upper-level Android mobile programming course or a graduate level software engineering course at the University of Maryland. Participants had varying degrees of familiarity with the Android framework, ranging from no knowledge to expert, and each had over 2 years of experience in programming. Since our participants were not professional testers and were instead using the app as opposed to testing, the terms tester and user are used synonymously through out the rest of the paper.

3.3 User Sessions & Surveys

A typical session lasted approximately 30 minutes and was broken into two sub-studies. The first study, which we will refer to as the "description study", asked users *to read a set of app descriptions* and then answer a one question survey regarding their expectation of the app's behavior based on the description. The second study, which will be referred to as the "use study", asked users to *use a set of apps* for 3 minutes each and complete a similar survey based on their interaction with the app. Note that the apps evaluated by a user in the description study were never the same as those evaluated in the use study (i.e., no user completed a survey for an app in the description study and then also completed a survey for the app in the use study).

In the description study, users were handed a copy of an app's description taken from the Google Play Store as well as a survey. Users were asked to read both documents and complete the one question survey which asked:

Based on the description you just read, which of the below behaviors would you NOT be surprised the app did?

The survey question was constructed to allow users to focus on what the app did (as opposed to what it did not) and make sure behaviors the user was uncertain of were not labelled unexpected. A checklist of possible behaviors was then listed and users were asked to check those behaviors which best applied. Each listed behavior correlated to one of Android's permission categories and a written description of the behavior was provided based on Android documentation. This description was provided to assist those users unfamiliar with the Android permission scheme and assist in behavior comprehension. For example, the contacts behavior correlated to apps who requested at least one of the following permissions: android.permission.READ_CONTACTS or andriod.permission.WRITE_CONTACTS. These permissions had the following documentation:

- READ_CONTACTS: Allows an application to read the user's contacts data.
- WRITE_CONTACTS: Allows an application to write (but not read) the user's contacts data.

As a result, the contacts behavior next to the checkbox on the survey was listed as follows:

☐ Access or write to your contacts.

In total, there were 10 behaviors listed on the survey–accounts, audio, bluetooth, camera, calendar, contacts, location, network/internet/wifi, phone, and SMS/MMS.

During the use study, the user was given an Android device and asked to use a given app until notified to stop. Each user was instructed to pay special attention to UI elements such as text, pictures or videos during their use as well as explore the app's UI and execution space as much as possible (i.e. click all possible options to see different windows and dialogs). In addition, users were given a new survey. This time asking, *"Based on your interaction with the app you just used, which behaviors would you NOT be surprised the app did?"* The same 10 behaviors were provided as choices.

In total, 10 user study sessions were conducted. For the description study, each user read the description of 2-3 apps

Table 1: The apps evaluated and their survey coverage. "D" is the number of completed description surveys and "U" is the number of completed user surveys. Apps in the left-hand table have multiple surveys for both studies, while those on the right-hand have multiple for one.

App	Category	Installs	D	U
Accuweather	Weather	>10M	3	3
Bubble Shooter Galaxy	Casual	>5M	2	3
Dictionary.com	Books & Ref	>10M	2	2
LED Flashlight	Tools	>100M	3	2
Runkeeper	Health & Fitness	>10M	2	3
Utorrent	Media & Video	>50M	3	2
YouTube	Media & Video	>1B	3	3

App	Category	Installs	D	U
Amazon	Shopping	>10M	1	2
Google+	Social	>1B	2	1
Instagram	Social	>500M	0	3
Pandora	Music & Audio	>100M	1	3
Skype	Communication	>500M	3	0

and for the use study each user used 2-3 apps. Whether a user evaluated 2 or 3 apps for either study was based only on the allotted time and the user's availability.

3.4 Evaluation Methods

In order to answer our research question, we constructed a series of venn diagrams to visualize and assess the relationship between the perspectives. Specifically, we generated the individual venn diagrams for each app (Fig. 1) by doing the following:

1. For both the description and use study, we averaged the available survey responses for each. A behavior was included in the average set if 50% or more of the users had checked the behavior. The resulting average set is referred to as the *mental model* for the given study. A mental model is said to be for a specific *perspective* based on the study the model represents. Therefore, each app has potentially 2 mental models developed from the studies. These models represent two different perspectives: 1) D, the mental model built from responses for the description study and representing the description perspective, and 2) U, the model built from the use study and representing the use perspective.
2. We then obtained M, a model based on the app's requested permissions and representing the permission perspective. We obtained these permissions by disassembling the apk file with apktool [4] and mapping the requested permissions to the behaviors based on PScout and Android documentation [3, 9]. Note only behaviors which mapped to the 10 behaviors listed on the survey were included and therefore any permissions mapping to different behaviors were excluded from the model.
3. We then plotted the available perspectives–U, D, or M–against each other using the R 'VennDiagram' package [37].

To obtain a summary venn diagram (Figure 2), we plotted the sum of each category (i.e., $U, M, D, U \cap M, U \cap D, D \cap M, U \cap M \cap D$) across the individual apps. To restrict errors, we restricted the sums to only include the 7 apps that contained multiple survey responses for each study.

All additional analysis was done using basic set operations on the resulting venn diagrams.

3.5 Threats to Validity

At this time, we outline several potential threats to validity. First, since the apps evaluated are well known apps, prior knowledge of the app may have been brought to the study. For example, users evaluating Skype may have used the application or been aware of its behavior prior to completing the survey. This previous knowledge may have influenced their response and allowed them to more accurately predict the app's behavior. That said, this would only cause the survey results to be more likely to match the app's true behavior and other perspectives, not less likely. Second, the behaviors evaluated in our survey represent only a subset of possible app behaviors. Therefore, the models we construct are not exhaustive models of each app. The survey excluded other functions such as accessing bookmarks, history, and social streams as well as reading and writing to external storage. Further studies could be done to include these and other additional behaviors.

4. SURVEY RESULTS & DISCUSSION

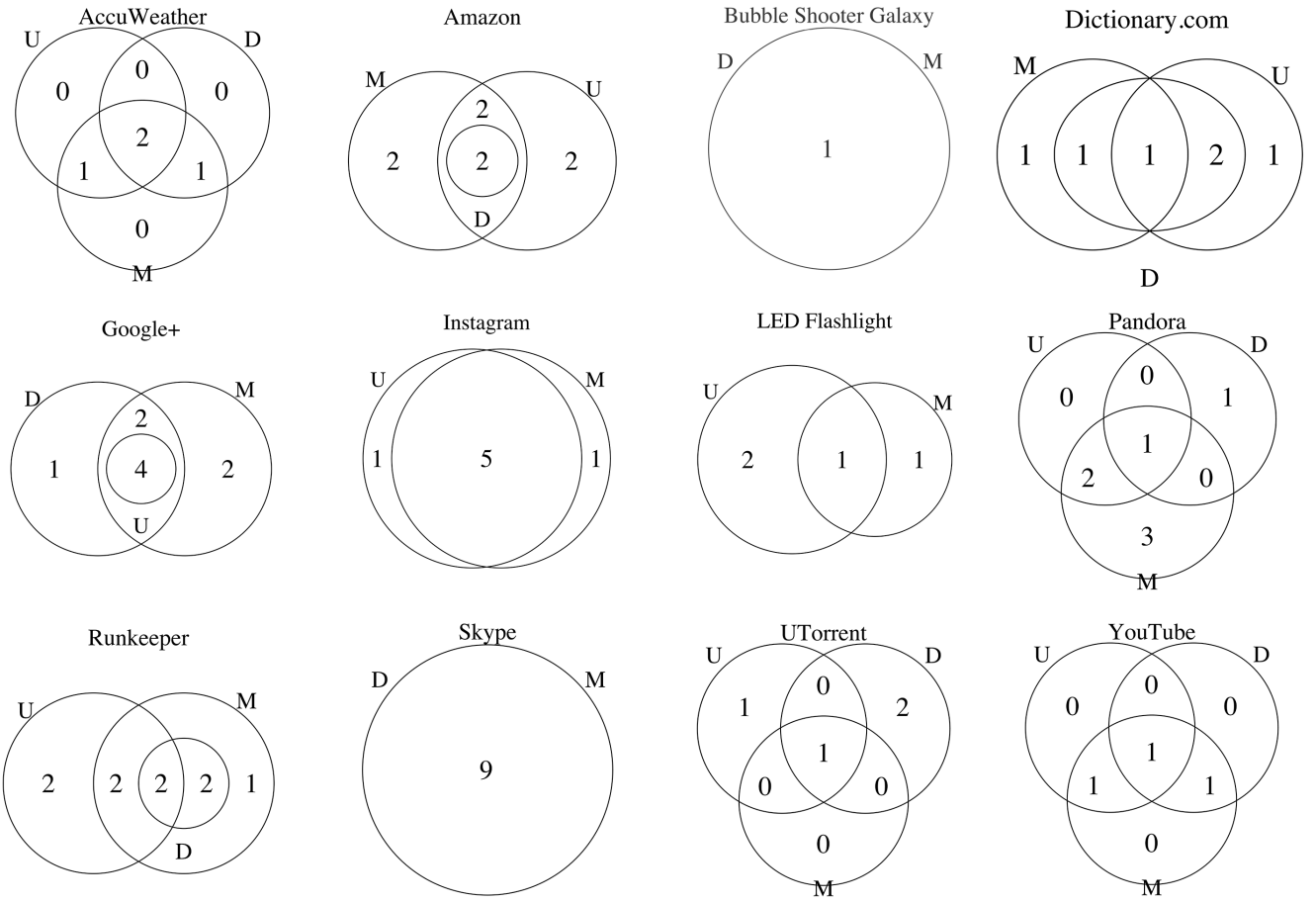
We now discuss the results of our research question, "Does the use of an app give information regarding an app's functionality that is not given in other commonly used sources?" We approach this question from two angles: 1) how the use perspective relates to other perspectives for individual apps in the study and 2) how the use perspective compared across the entire sample. Overall, 52 surveys were collected with 25 collected for the description study and 27 for the use. The survey coverage for each app can be seen in Table 1. Note the 7 apps listed on the left have multiple survey responses for both studies, while the ones on the right have multiple surveys for only one.

4.1 U vs. D vs. P: Per App

We begin by looking at the results for each app in the survey and their resulting venn diagrams (Fig. 1). As discussed in Section 3, we constructed these diagrams for each app by averaging the individual survey responses and collecting its requested permissions from the AndroidManifest.xml file. As a result, we obtained at most three sets of behaviors for each app which we call a model for a *perspective*. These models include:

- U(use perspective), the average set of behaviors users expected based on using the app
- D(description perspective), the average set of behaviors users expected based on the app's description

Figure 1: Comparing suggested behaviors in each app. "M" represents behaviors based on an app's permission requests. "U" and "D" are the average behaviors expected by users based on the app's description (D) or their use of the app (U). The numbers represent the number of behaviors in each set and intersection.



- M(permission perspective), the set of behaviors based on the requested permissions in the app's Manifest file

Each of these perspectives were then plotted against each other as venn diagrams. Below we highlight some observations made from these diagrams.

1: No app had all three perspectives equal. This means that none of the apps resulted in three equal sets where $U = M = D$. That said, Skype did obtain equality for all perspectives collected for it. Specifically, Skype had no survey response for U, but did have $D = M$. This result means the average user reading Skype's description found all requested permissions to be reasonable from the description. The app requested 9 of the 10 permission related behaviors: accounts, audio, bluetooth, camera, contacts, internet/wifi/network, location, phone, and sms. The average user response had checked all of these. Bubble Shooter Galaxy was also close to achieving such equality, and will be discussed in more detail in observation 4.

2: Two apps had all behaviors supported by at least two perspectives. AccuWeather and YouTube were close to achieving three equal sets with all of their behaviors being supported by at least two perspectives. This is represented in the diagrams by all of the behaviors being contained in the intersections $U \cap M$, $M \cap D$, and $U \cap M \cap D$. For AccuWeather, $U \cap M$ = {accounts}, $M \cap D$ ={SMS/MMS}, and $U \cap M \cap D$ = {internet, location}. For YouTube, $U \cap M$ = {camera}, $M \cap D$ ={accounts}, and $U \cap M \cap D$ = {internet}. Neither app contained behaviors in $U \cap D$, meaning no behaviors were in both the U and D perspective that were not also supported in M.

3: Several apps had perspectives contained in other perspectives. These apps included Dictionary.com, RunKeeper, Google+ and Amazon. Such cases suggest two possible scenarios: the use or description perspective may offer a limited view of the app's behavior or the containing perspectives is too large in scope. In 3 of the 4 apps, it was D that was the contained set. In one instance D was contained in M (RunKeeper) and in the others it was contained in $U \cup M$ (Dictionary.com) and $U \cap M$ (Amazon). Google+ had U contained in $D \cap M$. Other than the three apps mentioned above (Skype, Amazon and AccuWeather), M was never contained in any other perspective.

4: Several apps had empty perspectives. Bubble Shooter

Galaxy, a popular Android game, obtained surveys for both studies but on average users who used the app did not expect it do any of the listed behaviors based on their use. Since U is an empty set, the resulting venn diagram plots only 2 perspectives, D and M, despite having collected results for all 3. The plotted perspectives are completely aligned, meaning that the behavior that is requested by the permission was expected by the readers of the description. This behavior contained in D and M was internet usage.

LED Flashlight also contained an empty set and users who read its description did not find any listed behavior reasonable to expect. As a result, D was an empty set and only U and M are plotted in the venn diagram. Both U and M disagree on the majority of the behaviors, sharing only one in common which was again internet usage.

5: On average, each app had 2 behaviors unsupported by other perspectives. Of the 12 apps evaluated, 24 behaviors were unsupported with each app on average containing 2 such behaviors. When considering all 12 apps, these were most often seen in the permission perspective, followed by the use perspective and then description. When considering only the 7 apps with multiple surveys for each perspective, these unsupported behaviors occurred most often in the use perspective, followed by permissions and then the description perspective. This suggests permissions offer the most unique perspective of the app followed by the use and the description, while in the more robust sample the use perspective offered the most distinct perspective of the app's behavior.

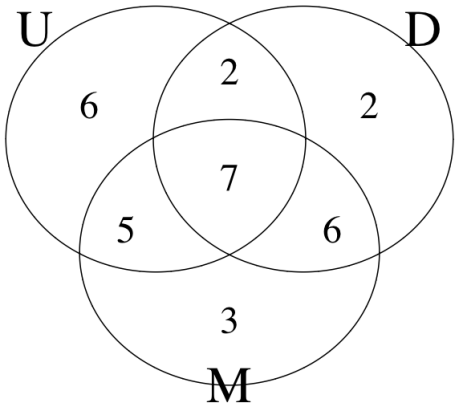
In conclusion, 3 apps had all their behaviors supported– Skype, AccuWeather, and YouTube (obs. 1 & 2). Of these 3, none had all three perspectives fully contained in $U \cap M \cap D$. On average each app had 2 behaviors that were unsupported by other perspectives (obs. 5). Further, of these perspectives, the use perspective was less likely than D to be contained and was occasionally nearly disjoint (obs. 3 & 4).

4.2 U vs. D vs. P: Overall

To generalize our results, we combined the individual venn diagrams into a cumulative venn diagram as described in Section 3 and presented in Figure 2. Overall, across the 7 apps considered, there were 34 unique behaviors suggested across the three perspectives. All three perspectives had a similar number of behaviors with $|M| = 21$, $|U| = 20$ and $|D| = 17$.

Of these behaviors, over half of each appeared in the intersection and were therefore supported by at least one of the other perspectives. Specifically, D had 15 of 17 intersect with $U \cup M$, U had 14 of 20 intersect with $M \cup D$, and M had 18 of 21 intersect with $M \cup U$. Of these intersections, $U \cap D$ had the least. $|U \cap M \cap D| = 7$ and therefore only 7 of 31 behaviors (22.6%) were predictable from all three perspectives: use, description and permissions. $U \cap M$ and $M \cap D$ were nearly equally split with $|U \cap M| = 12$ and $|M \cap D| = 13$. Therefore, if a behavior in M was not contained in both perspectives it slightly favored being suggested via description over use. $|U \cap D| = 2$ so very few behaviors were expected by users based on both the description and use that were not

Figure 2: Comparing behaviors across all apps. "M" represents behaviors based on app permission requests. "U" and "D" are behaviors expected by users based on its description (D) or their use of the app (U). The numbers represent the number of behaviors in each set and intersection. Only apps with multiple surveys for both sections are considered.



represented in the permissions. Similarly, only 2 behaviors were in D that were unsupported by the others. 18 of 21 behaviors in M were within $(U \cap M) \cup (D \cap M)$. This means of the behaviors found in the permissions roughly 86% were supported via use or reading the description.

While each set appeared to have the majority of its behaviors supported by at least one other perspective, when viewed collectively each set still gave a distinct perspective of the app's behavior with a large number of behaviors contained in one perspective and not in the others. In fact, $|U + M + D| = 11$. Therefore, 11 of 31 behaviors (nearly 35%) are suggested from only one perspective. Of these 11 behaviors, 6 were suggested by use only, 3 by permissions, and 2 by description. Therefore, users tended to consider more behaviors reasonable when using the app, but these behaviors were nearly a third of the time unsupported by the app's permissions or description. In contrast, behaviors suggested by permissions or description where unsupported by any other perspectives only 13% of the time.

In conclusion, each perspective's mental model had 70-88% overlap with other perspectives, but the remaining 12-30% provided a unique view of the app's behavior with mental models built while using the app being the most disjoint perspective of the three. Combined with the per app results discussed above, we conclude that the use perspective does indeed offer information about an app's behavior not provided in common sources such as the description or requested permissions (RQ).

5. CONCLUSION

Over the last 2 decades, many of us have advocated the automation of GUI system testing. In this paper, we have taken a step back, and instead have shown the benefits of performing GUI testing manually. We showed that the human tester/user implicitly builds a mental model of the software while testing/using the software. This mental model,

if output and used correctly, can be used for other software quality assurance activities. We demonstrated that a very simple mental model, captured as a set of expected permissions, has the potential to gain a unique understanding of a software's capabilities than would be obtained from its public description and the formal permissions it seeks combined.

Our survey based approach is just a starting point to understanding the true potential of manual GUI testing. Much future research is needed – we have identified 3 for the immediate future. First, more sophisticated mental models can be developed, perhaps reused from the human-computer interaction literature, and their benefits studied. Second, tied to these additional mental models, one needs to develop new perspectives, so that other downstream activities may be developed. Finally, new processes that support the new "more central" role of manual GUI testing need to be developed.

6. ACKNOWLEDGMENTS

This material is based on research sponsored by the National Science Foundation 1205501 and DARPA under agreement number FA8750-14-2-0039. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

7. REFERENCES

[1] K. Alharbi, S. Blackshear, E. Kowalczyk, A. M. Memon, B.-Y. E. Chang, and T. Yeh. Android apps consistency scrutinized. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pages 2347–2352. ACM, 2014.

[2] C. Andersson and P. Runeson. Verification and validation in industry - a qualitative survey on the state of practice. In *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*, pages 37–47, 2002.

[3] Official android documentation. http://developer.android.com.

[4] Apktool. http://ibotpeaches.github.io/apktool/.

[5] AppBrain. Appbrain stats. http://www.appbrain.com/stats/libraries.

[6] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[7] S. Artzi, J. Dolby, S. H. Jensen, A. Moller, and F. Tip. A framework for automated testing of javascript web applications. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 571–580. IEEE, 2011.

[8] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *SIGPLAN Not.*, 49(6):259–269, June 2014.

[9] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

[10] Z. Aung and W. Zaw. Permission-based android malware detection. *International Journal of Scientific and Technology Research*, 2(3):228–234, 2013.

[11] O. El Ariss, D. Xu, S. Dandey, B. Vender, P. McClean, and B. Slator. A systematic capture and replay strategy for testing complex gui based java applications. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 1038–1043. IEEE, 2010.

[12] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[13] A. P. Felt, S. Egelman, and D. Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 33–44. ACM, 2012.

[14] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM, 2012.

[15] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. Scandroid: Automated security certification of android applications. *Manuscript, Univ. of Maryland, http://www. cs. umd. edu/avik/projects/scandroidascaa*, 2(3), 2009.

[16] W. O. Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.

[17] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1025–1035. ACM, 2014.

[18] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 101–112, New York, NY, USA, 2012. ACM.

[19] M. G. Helander. *Handbook of human-computer interaction*. Elsevier, 2014.

[20] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang. Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1036–1046. ACM, 2014.

[21] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A conundrum of permissions: installing applications on an android smartphone. In *Financial Cryptography and Data Security*, pages 68–79. Springer, 2012.

[22] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 501–510, New York, NY, USA,

2012. ACM.

[23] A. M. Memon, M. E. Pollack, and M. L. Soffa. Hierarchical gui test case generation using automated planning. *Software Engineering, IEEE Transactions on*, 27(2):144–155, 2001.

[24] K. Naik and P. Tripathy. *Software testing and quality assurance: theory and practice*. John Wiley & Sons, 2011.

[25] J. Nielsen. *Usability engineering*. Elsevier, 1994.

[26] D. A. Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[27] A. C. Paiva, N. Tillmann, J. C. Faria, and R. F. Vidal. Modeling and testing hierarchical guis. In *Abstract State Machines*, pages 329–344. Citeseer, 2005.

[28] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security*, volume 13, 2013.

[29] S. J. Payne. Mental models in human-computer interaction. *The Human-Computer Interaction Handbook*, pages 63–75, 2007.

[30] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 71–72. ACM, 2012.

[31] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM, 2012.

[32] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human-computer interaction*. Addison-Wesley Longman Ltd., 1994.

[33] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1354–1365. ACM, 2014.

[34] B. Shneiderman. *Designing the user interface*. Pearson Education India, 2003.

[35] J. Spolsky. Top five (wrong) reasons you don‘t have testers. In *Joel on Software*, pages 171–177. Apress, 2004.

[36] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*. Citeseer, 2012.

[37] Venndiagram- r package. http://cran.r-project.org/web/packages/venndiagram.

[38] Q. Xie and A. M. Memon. Designing and comparing automated test oracles for gui-based software applications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(1):4, 2007.