User Interface Matters : Analysing the Complexity of Mobile Applications from a Visual Perspective

Corral, Luis

2021

Corral , L , Fronza , I & Mikkonen , T 2021 , ' User Interface Matters : Analysing the Complexity of Mobile Applications from a Visual Perspective ' , Procedia Computer Science , vol. 191 , pp. 9-16 . https://doi.org/10.1016/j.procs.2021.07.039

http://hdl.handle.net/10138/341045
https://doi.org/10.1016/j.procs.2021.07.039

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 191 (2021) 9–16

Procedia Computer Science

www.elsevier.com/locate/procedia

The 18th International Conference on Mobile Systems and Pervasive Computing (MobiSPC)
August 9-12, 2021, Leuven, Belgium

User Interface Matters: Analysing the Complexity of Mobile Applications from a Visual Perspective

Luis Corral[a], Ilenia Fronza[b,*], Tommi Mikkonen[c]

[a]*Monterrey Tech, Queretaro, Mexico*
[b]*Free University of Bolzano, Bolzano, Italy*
[c]*University of Helsinki, Helsinki, Finland*

Abstract

Product-centric techniques to analyze mobile applications leverage traditional source code analysis, size, market success, complexity, and others. Many of these techniques converge in the strategy of evaluating metrics taken from the source code that delivers the functionality of the software product. However, when following the Model-View-Controller (MVC) architecture, mobile applications are typically constructed by a compound of at least two programming languages, one to deliver the functionality and the other to describe the visual aspects. The latter is commonly left out from source code analysis, even though critical parts of the application are present in the graphic User Interface (UI). In this paper, we identify an opportunity to strengthen the product-centric mobile app analysis by incorporating UI metrics. This approach aims to enhance the expressiveness of source code metrics and deliver a more comprehensive analysis of the complexity, maintainability, and effort estimation of a mobile app. To introduce the concept, we present a case study realized using a block-based programming language to create mobile apps, in which we describe and calculate functional and UI metrics, discover commonalities and differences, discuss traits, and open tracks for further research.

Keywords: Mobile applications; User Interface; Complexity; Metrics.

1. Introduction

Source code metrics deliver value to understand both software products and processes. These metrics are used to understand the size and complexity of the product, anticipate the necessary development effort [13], outline market success, etc. Several source code metrics are used by researchers and practitioners, and extensive research has been done to identify, map and classify them [17]. Mobile software products are also studied using software metrics. Their source code metrics are typically calculated using the programming language that delivers the functionality of the

* Corresponding author. Tel.: +39 0471 016247
E-mail address: ilenia.fronza@unibz.it

1877-0509
10.1016/j.procs.2021.07.039

software product, for instance Java for Android or Objective-C/Swift for iOS. However, many mobile applications follow the Model-View-Controller (MVC) architecture [11], which commonnly implies the usage of more than one programming language. For example, in a traditional Android application, one can find mainly two languages, one to deliver the functionality (Java) and the other to describe the visual aspects (XML). In source code metrics, the language delivering the visual aspects is commonly left out from the analysis, even though critical parts of the application are present in the graphic User Interface (UI), including the construction of the screens, widgets, and images. In particular in Agile development contexts, determining technical debt is of prime importance; ignoring UI is a deficit in many of today's approaches.

In this paper, we identify an opportunity to strengthen source code metrics for mobile applications, implementing a strategy to balance the data collected using the *functional* source code, and complement them with *user interface metrics* to analyze the visual aspects of the application. We envision to make more robust the analysis of an application understanding its main components: visual and functional. Consequently, the expressiveness of mobile source code metrics will deliver a more comprehensive analysis to understand better the product in terms of complexity, maintainability, and effort estimation. To demonstrate our vision and illustrate our analysis, we present a case study, observed using a block-based programming language for mobile applications, in which we describe and calculate separately functional and UI metrics, observe commonalities and differences, and discuss product traits. The rest of this paper is structured as follows: Section 2 discusses related work and previous efforts that lay the foundation to build a novel proposal; Section 3 introduces our vision to collect source code metrics in diverse focus areas; Section 4 describes the case study in which we illustrate our approach to UI software metrics; Section 5 discusses the findings of the case study, risks and opportunities. Section 6 establishes expectations for future efforts, and Section 7 draws conclusions.

2. Background and Motivation

Software metrics have a very strong body of research. Source code metrics are heavily used both in industrial and academic settings to understand software processes, products, human aspects, and so on [19]. Analyzing the full body of research on this topic falls beyond the scope of this paper; however, we consider it relevant to understand how mobile Software Engineering has embraced the use of source code metrics for different purposes, including:

- **Structural analysis for effort estimation:** Kaur et al. compared static code metrics and process metrics for predicting defects in open source mobile applications, finding that process metrics are significantly better than source code metrics as predictors [9]. Catolino et al. presented a set of metrics to estimate the effort in early phases of development of mobile apps; however, not all these are focused on the source code itself [4].
- **Source code analysis for functional sizing:** Ferruci et al. proposed the COSMIC framework as code size measure; they analyzed 13 mobile apps to outline their functional size [6].
- **Source code analysis for security matters:** Sadeghi et al. created a set of vulnerability detection rules, and implemented it in a static analyzer that tries all the rules and finds all matches in the source code and metadata of a set of mobile apps [23]. Rahman et al. investigated how static code metrics, extracted from the source code of Android applications, can be used to predict security and privacy risks of Android apps. The authors found that, a proper selection of static code metrics can be used to predict security and privacy risks [20].
- **Source code analysis for app market prediction:** Corral and Fronza [5] analyzed the source code of 100 Android applications, concluding that the quality of the source code has a marginal impact into the indices that describe the market success. Martin et al. developed a systematic study and concluded that source code quality has no strong correlation to app success for open source apps [14]. Later, Catolino challenged these two previous assertions, affirming the existence of a relation between code quality and commercial success, in particular inheritance and information hiding metrics [3].
- **Usability of mobile applications:** The usability of mobile applications is critical for their adoption, in particular for a set of challenges related to the characteristics of mobile devices, such as the screen size, the display resolution and the capacity of memory [16]. The impressive number of users generates the need to evaluate the usability of applications. Research in this field has focused on understanding how to quantify the user experience [2] mainly by extracting usability-related information from UI events [8, 12] or by collecting user reviews [29]. Moreover, effort has been dedicated to exploring how traditional usability metrics (i.e., created

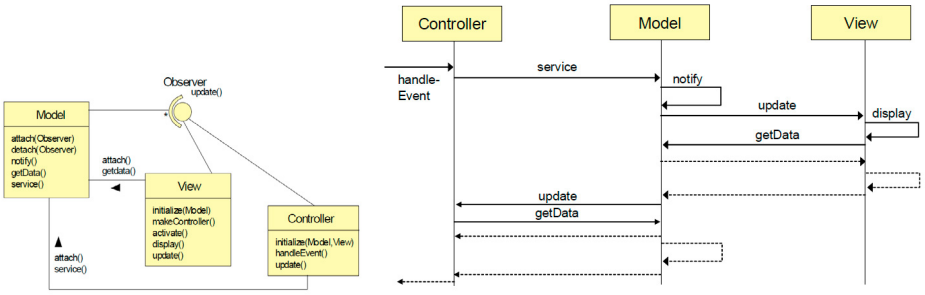

Fig. 1. MVC pattern class and sequence diagrams. Model holds the data, View displays it, and Controller enables manipulating data.

for desktop computer applications) can be used to propose interface design principles for mobile applications [1, 7]. Saleh et al. proposed a tool to evaluate the usability attributes and metrics of mobile device applications by using the unsupervised usability evaluation tool [24]. The tool presented by Riegler et al. calculates UI complexity metrics for mobile applications, by considering a variety of visual aspects, including the number of user interface elements, alignment, grouping, balance, density, the size of objects and consistency measures [21]; however, it does not consider the insights given directly from the language in which the UI is defined.

With respect to product analysis, we can find convergence in the strategy of evaluating source code metrics calculated mostly using as reference the programming language that delivers the functionality of the software product, only with the exception of [6] that analyzes the size of the UI-relevant XML files of Android applications, and [23] that analyzes other XML metadata. With such approach to collect software metrics, an important part of mobile application has been overlooked by previous studies. Determining technical debt is of prime importance, however, we can observe that ignoring UI is a deficit in many of today's approaches. Mobile applications can be computationally simple, while incorporating a heavy design, implementation and overall effort in constructing a complex UI that services a critical aspect of the product *communication* and *interaction* with the user. Previous research falls short at considering the UI definition of the product, and this gap opens the door to introduce a novel strategy to make more robust the analysis of a mobile app by posing a question: ***How complex can a mobile app be by analyzing its user interface?***. The answer may shed light on completing the understading of the complexity of a mobile application by considering UI as a complementary approach to the information delivered by traditional source code metrics.

3. Enhanced Metric Collection for Mobile Apps

Mobile apps are every day more and more complex. Depending on the target operating system (OS) and software development kit (SDK), mobile apps can combine foreground activities, services, notifications, resources and much more. Regardless of the OS and SDK, the organization of mobile apps concur in implementing the Model-View-Controller (MVC) architecture (Figure 1), that requires developers to separate the logic and presentation layers.

In practice, this requires the organization of the application logic in a tier (i.e., a particular physical file), written in a specific language, and the construction of the UI in a different physical file, coded in a different language. Examples of this organization are Java for the logic and XML for the UI using the Android SDK, or Typescript for the logic and HTML5/CSS for the UI using the Ionic SDK. All activities (i.e., screens) in an application are constructed by the compound and linkage of these two files, one to render the UI and the other to execute the logic within.

As illustrated in the body of research on source code metrics for mobile apps, one can leverage programming languages such as Java, Objective-C or JavaScript to calculate metrics that can shed light on quality aspects of the product. Many tools exist to perform automated analysis and deliver strong sets of metrics from programming languages implementing the application logic. In addition to classical results (e.g., [15, 22]), recent research has shown that interest remains in source code metrics to calculate size [6, 4], complexity, or object-oriented relevant metrics like depth of inheritance, coupling and cohesion [5, 3].

3.1. Introducing "User Interface Metrics"

Analyzing a UI is a process that can be done both visually and programatically. Markup languages like XML or HTML allow the definition of a structure that would be eventually rendered in graphic components. We propose an approach to survey the UI complexity starting with a basic yet comprehensive method. Let us take the example of markup languages like XML for Android, which allow the definition of components (e.g., layouts, widgets, graphics) via XML tags whose properties and content define the aspect of the element, as shown in Figure 2.

```
<Button
android:layout_width="wrap_content"
android:text="@string/button_text" />
```

Fig. 2. Definition of a button in an Android user interface.

For an initial notion of the size and complexity of user interfaces in a mobile app, we propose to start counting the following elements (acronyms are defined for eventual references):

- **Number of activities (NAC)**, the number of independent screens offered by an application.
- **Number of layouts (NOL)**, the number of structural elements to organize the views in a screen.
- **Number of views (NOV)**, the number of widgets (buttons, images, text fields) that an application includes.

A plain count would give an outline about the size and structure of an application. However, this count can be better understood if complemented by the analysis on the relationship between the UI components. For example, if in a List View of *n* elements each element has an image, such list view has an additional level of depth since it has one child. As result, we propose to count as well:

- **Depth of the UI tree (DOT)**, the maximum number of levels of children components a user interface has.

Let us take as example a trivial Android app that consists of a single activity (screen) that displays a basic text. Figure 3 how the UI is defined in an activity XML and using a block based programming language like Thunkable (http://www.thunkable.com) or AppInventor (http://appinvenor.mit.edu).

```
<RelativeLayout xmlns:android=""
xmlns:tools="http://schemas.android.com/tools"
... />
<TextView
android:layout_width="wrap_content"
... />
</RelativeLayout>
```

Fig. 3. Definition of a basic App: XML definition (on the left) and tree definition of the app activity in Thunkable (on the right).

The UI definition of this app consists of a single screen, with an organizing layout that hosts widgets. In this simple case, there is a simple text label as shown in Figure 4. Consequently, when analyzed for UI metrics, the application will deliver the numbers shown in Table 1. If we perform this analysis to more complex apps, we can extend our knowledge and understanding of the user interfaces, not only in terms of the mere size of the UI-relevant file, but also in the way that such UI is constructed, configured, and implemented.

4. Case study

To illustrate the potential of our UI-based analysis, we present a case study that analyzes 10 mobile application projects in which we calculate the UI metrics proposed Section 3.1, and we collect as well functional metrics that permit to add context to compare functional complexity and UI complexity. The 10 mobile apps were developed using

Fig. 4. Visual rendering of a basic Android activity.

Table 1. UI metrics for the simple app in Figure 3.

Metric	Value	Rationale
Number of Screens	1	The app has a single screen
Number of Layouts	1	The screen has a Relative Layout
Number of Widgets	1	The screen has a TextView
Depth of UI tree	2	The Layout (Row) has one child

a block-based programming language (BBPL), as an initial approach to a very well-differentiated MVC structure that permits to focus the scope of our analysis, whilst having a large outreach of developed applications. BBPL platforms count users and products in millions: as of 2021, Scratch repository hosted over 70 million projects created by over 66 million registered users[1], and App Inventor counted over 13 million users who created over 57 million mobile apps[2]. Also, there is a drastic increase of app building using BBPLs also among small businesses and entrepreneurs; for example, in 2018, over 16 million monthly active users around the world used apps built on Thunkable[3]. We decided to implement the initial case study analyzing Thunkable-built applications for two reasons:

1. Thunkable implements a block-based intermediate level of abstraction that leaves behind any need of coding in target-specific programming languages. The produced mobile applications can be built both for iOS and Android, removing as consequence any bias given by target-specific source code languages.
2. Thunkable organizes the source of the mobile application in the two layers previously discussed: a layer to design the UI and a separate layer to implement the functionality. This organization allows for the implementation of our proposed UI metrics.

4.1. Data collection

The 10 considered projects were developed in two separate training courses targeted to two different audiences. The teaching staff was the same for both courses. In both courses, the main requirement was to deliver a fully functional mobile application of a free topic chosen by the developer. To ensure interaction and internal processing, students were advised to create applications that receive an input from the user, elaborate internally, and produce an outcome.

The analyzed applications are (acronyms are assigned for an easier reference): 2 education applications to promote reading and mathematics skills (EA1 and EA2), 1 museum tour guide application (MT1), 2 walking tours for an historic old town (WT1 and WT2), 1 app to obtain information about commercial goods (CG1), 1 app to read news (NW1), 1 app to assist the decision on what degree to study (DS1), a simulation of an online store (OS1) and an

[1] https://scratch.mit.edu/statistics/
[2] http://ai2.appinventor.mit.edu/stats
[3] https://blog.thunkable.com/the-18-best-thunkable-apps-of-2018-c17856a5f7de

application to send a mayday message in case of emergency (EM1). To provide an integral assessment of the mobile projects, our data collection strategy is twofold. First, we collected the *User Interface Metrics* proposed in Section 3.1. Then, we collected metrics related to the functional code of the product (i.e., the instructions or "lines of code" defined by the BBPL). This way, we can provide a better view about the UI metrics that are relevant for our proposal, as well as represent context on how complex the functional execution of the product is. The functional code metrics that we use are:

- **Number of blocks of code (BOC)**, the number of functional blocks present in the application.
- **Number of events (NOE)**, the number of event listeners present in the application (e.g., onClick).
- **Number of conditions (NOC)**, the number of if/if else blocks present in the application.
- **Number of loops (NLO)**, the number of loop blocks (while, for, and so on) present in the application.
- **Number of variables (VAR)**, the number of variables defined by the developer to store values.

4.2. Results

Table 2 shows the overall results of the calculations. The analyzed projects are diverse and their goals and functionality are distinct. A common factor is that they all imply a heavy interaction with the user, which requires a considerable amount of effort implementing both the UI and the logic within. Some of the insights provided by our analysis are listed below.

Table 2. UI and functional metrics on analyzed Thunkable projects.

Proj.	**UI Metrics**				**Functional Metrics**				
	NAC	NOL	NOV	DOT	BOC	NOE	NOC	NLO	VAR
EA1	3	11	41	4	74	10	6	4	13
EA2	6	34	45	4	186	32	23	0	3
MT1	7	55	42	4	42	22	0	0	0
WT1	13	56	10	4	22	9	0	0	0
WT2	13	84	59	5	117	42	0	0	3
CG1	6	40	27	5	16	8	0	0	0
NW1	7	0	14	2	10	1	2	0	1
DS1	13	0	41	2	39	18	0	0	1
OS1	31	0	38	2	48	6	12	0	0
EM1	1	20	10	4	16	5	0	0	2

Simple UI, complex logic: The emergency app (EM) implements a simple UI (only 1 screen and 10 elements) but makes use of several blocks and code constructs that make them complex for a single screen application.

Complex UI, simple logic: Projects that implement the mere simulation of a process, like the online store (OS), or interactive systems like the guides (MT1, WT1, WT2), or the assistant to choose a degree (DS1), are very simple in terms of the code blocks used (i.e., only event listeners and blocks to navigate between screens); however, they are visually complex, as they integrate a large number of screens, images and widgets to promote user interaction. From the perspective of a traditional source code analysis, one could conclude that these apps are simple, yet our UI metrics model shows that they are not.

Complex UI, complex logic: Educational apps (EA1, EA2) are complex in the two sides. Indeed, they are visually attractive and complex (including canvas and draggable components) to attire the attention of children; the logic to manage all the elements is complex as well.

5. Discussion and Limitations

We acknowledge that this analysis is in its infancy and that a number of limitations needs to be overcome before drawing solid conclusions. First, this analysis was implemented in a code-agnostic platform: although we discussed

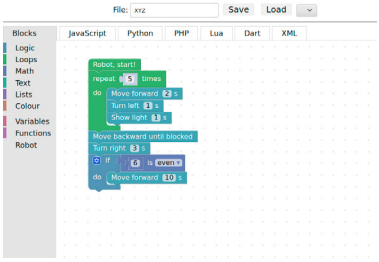

Fig. 5. Robot programming based on Blockly [10].

the advantages of this approach, the analysis needs to be extended to target-specific languages for different mobile operating systems, to determine bias or variations injected by languages themselves. Second, the number of studied applications can be considered small; the metric collection shall be performed in a larger set to make it statistically relevant. Finally, the set of metrics shall be reviewed, to identify other UI-relevant elements that could be insightful to calculate (e.g., resources like images or videos), or even derived metrics (e.g., average widgets per screen

We understand that this analysis could have as well limitations, but it represents an initial vision that is yet to be perfected. For instance, an important limitation that our strategy finds is when trying to analyze user interfaces built in Unity (`https://unity.com`) or similar graphic engines, which make use of complex constructions that are harder to examine, and hence are out of reach of this initial proposal. The proposed metrics directly reflect UI composition and widget hierarchy, which represent an initial approach for deeper understanding. Even UIs with many elements and deep hierarchy may appear simple and easy to use, depending on the development framework with which the UI composition is developed, it may be easy to maintain. Moreover, limiting analyzed apps to one framework and a specific programing model adds validity threats to the results. To extend the inference on the application of this analysis, subjects to validate the metrics may come from a boarder context, for example open source projects (`http://f-droid.org`).

6. Future Work

With the challenges identified, as next steps we propose to extend the analysis of UI metrics to a larger sample of Thunkable applications, to reproduce the experiment in a similar setting and validate the traits observed in our case study. After completing this analysis, we recommend to replicate the study incorporating other block-based languages such as AppInventor [27], in the context of mobile apps, or even taking our approach to other development environments of different scopes, like robot programming based on Blockly (Figure 5) [25]. Indeed, many mobile apps are today used to communicate with external devices, with the mobile device acting as a gateway, meaning that our metrics can also be applied in such setting as well.

A next stage of this exploration is the implementation of UI metrics in mobile development projects that make use of platform-specific SDKs, for instance Android SDK, iOS Apple Developer, Ionic or React Native, since they require the construction of user interfaces and the development of functionality in specific programming languages that may impact, pose challenges, or add considerations to the analysis performed purely in block-based programming languages. To execute this analysis, it is recommended as well the implementation of language-specific parsing tools that can help to calculate automatically the values of each UI metric. In general, a replication exercise in Android Apps and iOS apps is highly recommended to gage the effectiveness of the metrics set, the analysis, and suggest changes.

7. Conclusions

Mobile applications are software products that, by design and operative target, involve intensive interaction with the user. Cellular phones and tablets demand a lot of attention from users. UI design is a very important process in mobile application development, which needs specific design experience due to its physical limitation, limited small screen size [26]. The development of user interfaces has been studied and orchestrated via design patterns or design practices

[18, 28]; nonetheless, in addition to the procedural point of view, understanding the complexity of UI development from a product perspective may provide the means to quantitatively assess how complex an application is from a visual perspective. In this paper, we set an outline to leverage the analysis of user interfaces as an opportunity to strengthen source code metrics to understand the mobile software product. Much work is yet to be done to grow this idea, validate it, and evolve it until it reaches its practical and professional maturity.

References

[1] Adak, M.F., 2014. Studies on usability of mobile applications. Global Journal on Technology 6.

[2] Albert, W., Tullis, T., 2013. Measuring the user experience: collecting, analyzing, and presenting usability metrics. Newnes.

[3] Catolino, G., 2018. Does source code quality reflect the ratings of apps?, in: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, Association for Computing Machinery, New York, NY, USA. pp. 43–44.

[4] Catolino, G., Salza, P., Gravino, C., Ferrucci, F., 2017. A set of metrics for the effort estimation of mobile apps, in: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), IEEE. pp. 194–198.

[5] Corral, L., Fronza, I., 2015. Better code for better apps: A study on source code quality and market success of android applications, in: 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, IEEE. pp. 22–32.

[6] Ferrucci, F., Gravino, C., Salza, P., Sarro, F., 2015. Investigating functional and code size measures for mobile applications, in: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, IEEE. pp. 365–368.

[7] Harrison, R., Flood, D., Duce, D., 2013. Usability of mobile applications: literature review and rationale for a new usability model. Journal of Interaction Science 1, 1.

[8] Hilbert, D.M., Redmiles, D.F., 2000. Extracting usability information from user interface events. ACM Computing Surveys 32, 384–421.

[9] Kaur, A., Kaur, K., Kaur, H., 2015. An investigation of the accuracy of code and process metrics for defect prediction of mobile applications, in: 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions), pp. 1–6.

[10] Kinnunen, P., 2020. Creating a Visual Learning Environment to Introduce Programming and Robotics to Children. Master's thesis, University of Helsinki, Finland.

[11] Krasner, G.E., Pope, S.T., 1988. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of object oriented programming 1, 26–49.

[12] Ma, X., Yan, B., Chen, G., Zhang, C., Huang, K., Drury, J., Wang, L., 2013. Design and implementation of a toolkit for usability testing of mobile apps. Mobile Networks and Applications 18, 81–97.

[13] Mamun, M.A.A., Berger, C., Hansson, J., 2019. Effects of measurements on correlations of software code metrics. Empirical Software Engineering 24, 2764–2818.

[14] Martin, W., Sarro, F., Jia, Y., Zhang, Y., Harman, M., 2017. A survey of app store analysis for software engineering. IEEE Transactions on Software Engineering 43, 817–847.

[15] McCabe, T.J., 1976. A complexity measure. IEEE Transactions on software Engineering , 308–320.

[16] Moumane, K., Idri, A., Abran, A., 2016. Usability evaluation of mobile applications using iso 9241 and iso 25062 standards. SpringerPlus 5, 548.

[17] Mushtaq, Z., Rasool, G., Shehzad, B., 2017. Multilingual source code analysis: A systematic literature review. IEEE Access 5, 11307–11336.

[18] Nguyen, T.D., Vanderdonckt, J., 2012. User interface master detail pattern on android, in: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems, pp. 299–304.

[19] Nuez-Varela, A.S., Prez-Gonzalez, H.G., Martnez-Perez, F.E., Soubervielle-Montalvo, C., 2017. Source code metrics. J. Syst. Softw. 128, 164–197.

[20] Rahman, A., Pradhan, P., Partho, A., Williams, L., 2017. Predicting android application security and privacy risk with static code metrics, in: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 149–153.

[21] Riegler, A., Holzmann, C., 2015. Ui-cat: calculating user interface complexity metrics for mobile applications, in: Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia, pp. 390–394.

[22] Rosenberg, L.H., Hyatt, L.E., 1997. Software quality metrics for object-oriented environments. Crosstalk journal 10, 1–6.

[23] Sadeghi, A., Esfahani, N., Malek, S., 2017. Mining mobile app markets for prioritization of security assessment effort, in: Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics, Association for Computing Machinery, New York, NY, USA. pp. 1–7.

[24] Saleh, A., Ismail, R., Fabil, N., 2017. Evaluating usability for mobile application: A mauem approach, in: Proceedings of the 2017 International Conference on Software and E-Business, Association for Computing Machinery, New York, NY, USA. p. 71–77.

[25] Trower, J., Gray, J., 2015. Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control, in: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 5–5.

[26] Wetchakorn, T., Prompoon, N., 2015. Method for mobile user interface design patterns creation for ios platform, in: 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 150–155.

[27] Wolber, D., Abelson, H., Spertus, E., Looney, L., 2011. App Inventor. " O'Reilly Media, Inc.".

[28] Yáñez Gómez, R., Cascado Caballero, D., Sevillano, J.L., 2014. Heuristic evaluation on mobile interfaces: A new checklist. The Scientific World Journal 2014.

[29] Zha, R., 2019. Improving the usability of mobile application user review collection. Master's thesis. Tampere University. Faculty of Information Technology and Communication Sciences. M.Sc. thesis.