

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/343206805>

On the Detection of Structural Aesthetic Defects of Android Mobile User Interfaces with a Metrics-based Tool

Article in *The ACM Transactions on Interactive Intelligent Systems* · July 2020

DOI: 10.1145/3410468

CITATIONS

3

READS

374

4 authors:



Narjes Bessghaier

École de Technologie Supérieure

7 PUBLICATIONS 38 CITATIONS

[SEE PROFILE](#)



Makram Soui

Saudi Electronic University

57 PUBLICATIONS 386 CITATIONS

[SEE PROFILE](#)



Christophe Kolski

Université Polytechnique Hauts-de-France

458 PUBLICATIONS 3,691 CITATIONS

[SEE PROFILE](#)



Mabrouka Chouchane

Ecole Nationale des Sciences de l'Informatique

8 PUBLICATIONS 63 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Project

Project-based learning [View project](#)



Project

HCI engineering integrated with capability maturity models [View project](#)

On the Detection of Structural Aesthetic Defects of Android Mobile User Interfaces with a Metrics-based Tool

NARJES BESSGHAIER*, ENSI Manouba, Tunisia

MAKRAM SOUI, College of Computing and Informatics Saudi Electronic University, Saudi Arabia

CHRISTOPHE KOLSKI, Université Polytechnique Hauts-de-France, France

MABROUKA CHOUCANE, ENSI Manouba, Tunisia

Smartphone users are striving for easy-to-learn and use mobile apps user interfaces. Accomplishing these qualities demands an iterative evaluation of the Mobile User Interface (MUI). Several studies stress the value of providing a MUI with a pleasing look and feel to engaging end-users. The MUI, therefore, needs to be free from all kinds of structural aesthetic defects. Such defects are indicators of poor design decisions interfering with the consistency of a MUI and making it more difficult to use. To this end, we are proposing a tool (ADDET) to determine the structural aesthetic dimension of MUIs. Automating this process is useful to designers in evaluating the quality of their designs. Our approach is composed of two modules. 1) *Metrics assessment*: is based on the static analysis of a tree-structured layout of the MUI. We used 15 geometric metrics (also known as structural or aesthetic metrics) to check various structural properties before a defect is triggered; 2) *Defects detection*: the manual combination of metrics and defects are time-consuming and user-dependent when determining a detection rule. Thus, we perceive the process of identification of defects as an optimization problem. We aim to automatically combine the metrics related to a particular defect and optimize the accuracy of the rules created by assigning a weight, representing the metric importance in detecting a defect. We conducted a quantitative and qualitative analysis to evaluate the accuracy of the proposed tool in computing metrics and detecting defects. The findings affirm the tool's reliability when assessing a MUI's structural design problems with 71% accuracy.

CCS Concepts: • **Human-centered computing** → **Human computer interaction (HCI); HCI design and evaluation methods; Graphical user interfaces; HCI theory, concepts and models.**

Additional Key Words and Phrases: Structural aesthetic defects, Automated evaluation, Android MUI, Optimization algorithm, NSGA-II

ACM Reference Format:

Narjes Bessghaier, Makram Soui, Christophe Kolski, and Mabrouka Chouchane. 2020. On the Detection of Structural Aesthetic Defects of Android Mobile User Interfaces with a Metrics-based Tool. *ACM Trans. Interact. Intell. Syst.* 1, 1, Article 1 (January 2020), 28 pages. <https://doi.org/10.1145/3410468>

1 INTRODUCTION

Nowadays, smartphones are an important part of human life. Such smart devices give their users considerable advantages in terms of usability, ubiquity and ease of use [32]. This trend has soared

Authors' addresses: Narjes Bessghaier, bessghaier.narjess@gmail.com, ENSI Manouba, Campus Universitaire de la Manouba 2010, Manouba, Tunis, Tunisia, 1120; Makram Soui, College of Computing and Informatics Saudi Electronic University, Abu Bakr Street, 11673, Riad, Saudi Arabia, m.soui@seu.edu.sa; Christophe Kolski, Université Polytechnique Hauts-de-France, Valenciennes, France, Christophe.Kolski@uphf.fr; Mabrouka Chouchane, ENSI Manouba, Campus Universitaire de la Manouba 2010, Manouba, Tunisia, chouchane.mabrouka@gmail.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.

2160-6455/2020/1-ART1 \$15.00
<https://doi.org/10.1145/3410468>

rapidly, and apps have become more sophisticated, providing consumers with many feature sets (high display resolution, 3G/4G network access, simulation of 3D games, and multimedia). As mobile apps become more accessible, versatile, and user-friendly, users tend to use mobile phones over software applications [39]. Users of smartphones communicate with an application through its user interface (UI), which is a touch-sensitive display that recognizes the events of the user and delivers the results via graphical components. Improving the usability of the MUI, therefore, requires the identification of the structural design defects. Mahajan et al. exposed that inconsistent GUIs (Graphical User Interface) could increase the completion of the task time between 10% and 25% [26]. Similarly, 65% of GUI defects resulted in a loss of some features, while 60% of code errors were at the GUI level [42]. Within this way, several studies have shown that usability is an essential measure of the quality and sustainability of mobile apps [6].

Aesthetics refers to the perceived aesthetic level of the MUI, and how it draws the interest of users. Aesthetics includes the structural (a layout-related property dealing with graphical elements geometric positioning) and the colorful aspect (the choice of harmonically appropriate colors). In computer science, we find aesthetics to be the level of the visual appeal of a design. It is a Usability sub-characteristic with a slight line of differentiation. On the one side, the perceived usability appears to generate an usable and informative MUI (i.e, given functionalities are informative and not misleading; data must be transparent and descriptive, etc). Through focusing on the structural placement and colorfulness of the MUI elements, aesthetics, on the other hand, focuses on user attraction. Aesthetics evaluation approaches are divided into two categories: 1) *Qualitative/Subjective evaluation*: the emphasis is on evaluating whether the concept can follow a set of visual guidelines [26, 30, 31]. Such assessment is achieved by getting experts or end-users present. 2) *Quantitative/Objective assessment*: it is based on the use of metrics to evaluate the characteristics of a design feature, such as cohesion, density and balance. Strict metric assessment involves how good or bad a specific structural property is.

Scholars have proved that aesthetics quality has an impact on the engagement level [4, 37, 38, 50, 53], the perceived usability [23, 46, 50, 51], and users loyalty [27, 46, 50]. Trkyilmaz et al. [52], highlighted that end-users rank the design aesthetics as necessary as the operational side of an application. Despite the importance the aesthetics plays in attracting end consumers, most of the current studies have proposed subjective [5, 13, 19, 49] and objective [33, 47, 53, 57] evaluation approaches and the identification of defects were fortunate.

We suggest an automated tool for evaluating the structural dimension of MUIs to cope with the limitations as described above. Our approach uses an optimization algorithm (NSGA-II) to generate defects detection rules to ensure a non-extensive knowledge of the rules specification. In addition, we automatically adjust the thresholds for metrics, as they are the essential and delicate factors in detecting a structural defect. A quantitative and qualitative analysis has proven the efficacy of the method in identifying defects in the visual quality.

The remainder of this paper is structured as follows. Section 2 provides the related work. Section 3 introduces and defines the set of selected metrics and defects. Section 4 presents the modules of our proposed tool ADDET, while section 5 validates the findings. In section 6, we discuss the results, while we present the threats to validity in section 7 and we conclude with some future research directions in section 8.

2 RELATED WORK

We present some of the studies targeting the analysis of the MUI quality. We focus on automated assessment and the application of predictive metrics. The techniques are classified into two categories: qualitative methods and quantitative methods.

2.1 Qualitative evaluation methods

There is a wide variety of attempts in the literature to determine the consistency of MUIs. Providing high-quality software is a significant move. It allows end-users with less effort to communicate with the app on various platforms. [7, 24, 25]. Scholars suggested the iterative approach to ensure that the end-users are involved in the program's life cycle. It is a cyclic prototyping process based on feedback from the user [11, 48]. A broad variety of usability evaluation tools and techniques such as laboratory testing, questionnaires, and inspection methods have been proposed [5, 19, 43]. Mainly, the inspection methods and questionnaires are addressed to desktop GUIs, which have different requirements for User eXperience (UX) than smartphones. Usability experts have sought to adapt those standards to the mobile world for this reason. Yanez re-adopted a desktop-centric checklist for the mobile interface to identify unique MUI problems and established independent OS guidelines [55]. Kuparinen et al. presented a comparison of generic heuristics and domain-specific heuristics for applications with maps [21]. The findings highlighted that the proposed domain-specific checklist aided users to find more usability issues. These studies developed heuristics manually to assess the MUI. The heuristics evaluation is one of the most widely used methods of evaluating mobile apps' usability in an early design phase. It tests a MUI by a set of rules [35]. This method is productive, but at the same time, error-prone, and subjective.

Other scholars have relied on predicting the user's aesthetics judgment through the crowd. Reinecke et al. have explored the impact of the first impression of visual complexity and colorfulness on the aesthetics of widescreens [40]. The experiment was performed across 450 websites. After 500ms, participants were asked to determine the complexity and appearance of the webpages. The results demonstrated how a strong negative association is found between the visual complexity and the aesthetic based on the judgment of users. As the complexity increases, the appeal diminishes. The colorfulness, on the contrary, did not show any significant impact on the first users perception of appeal. These results show that visual complexity is, at first sight, the dominant feature. A similar study was proposed by the [28] on 51 mobile user interfaces. Early work consisted of determining how complexity and aesthetics could be influenced by a particular exposure period by the users. For both of the quality dimensions, a small difference in means was noticed. This result may call for an empirical study aiming at how the familiarity of the users might change their judgment. In the second analysis, six metrics of consistency (symmetry, depth of color, visual noise, contrast, edge distortion, dominant colors) showed a good association with both aesthetics and complexity.

The above subjective methods appear to be time-consuming and error-prone, since they depend on participants' experience. In addition, these procedures are conducted at a late design stage which may cause a substantial delay in the release of the application.

2.2 Quantitative evaluation methods

In order to resolve the manual interpretation of rules, Ines et al. used the genetic programming to create evaluation rules for the automatic detection of usability defects in MUI [16]. To provide user-adapted detection rules, the authors considered the user profile parameters, such as motivation, level of experience, and age. These rules are a combination of user criteria, metrics, and defects. Their methodology was 70% successful. In the same line, Riegler and Holzmann suggested a set of 8 complexity metrics (number of UI elements, the smallness of element, misalignment, density, imbalance, the complexity of color, complexity of typography, and inconsistency) to determine the visual appearance of mobile apps user interfaces [41]. The method is based on a computer vision technique, which does not require the parsing of any code. The critical weakness of this analysis is the lack of tuning of metrics thresholds. As well, the metrics were not given any weights to know which parameter contributes more to the complexity. Alemerien and Magel introduced

GUIEvaluator, a metric tool that assesses the complexity of a GUI automatically. The method measures five quantitative measurements: orientation, grouping, size, density, and balance [1]. The authors have expanded GUIEvaluator to GUIExaminer, an SLC (Screen Layout Cohesion) metric measurement tool. The SLC is a hybrid metric gathering aesthetics (structural and color), and semantic dimensions of interface layout [2]. Soui et al. have developed a plugin called PLAIN that evaluates the usability quality of Android mobile apps user interfaces [47]. Eight metrics are determined by the tool: regularity, composition, sorting, complexity, integrity, density, repartition, and symmetry. We found that the parser used in PLAIN considers the *ViewGroups* and the *layouts* as visual components to evaluate metrics computation. Such a method can have an inaccurate number of components of the visual graph. Zen and Vanderdonckt have proposed QUESTIM, a region based method to measure a MUI as a screenshot based on a collection of metrics such as density, balance, balance, symmetry, borderBalance, borderDensity. User will load the website, upload a screenshot and draw the regions using a rectangle form [56, 57]. As selected arbitrarily, this analysis lacks the automated adjustment of metric thresholds. Alternatively, new metric values will be determined based on the shapes of the regions. In the same vein, a tool named tLight has been proposed with eight complexity and color metrics to assess the graphical quality of webpages and iPhone apps. The metrics only showed a high significance on webpages due to the different layouts and design guidelines used by Web and iPhone apps. [29].

None of the above tools implement a module for identification of defects, as we present in this study. However, we use QUESTIM and PLAIN to assess the accuracy of our proposed tool's metrics calculation module.

2.3 Genetic algorithms for the detection of design violation

Defining the defects detection rules manually requires extensive UI/UX domain knowledge. However, these manually constructed rules are not scalable, because designs are rapidly changing, and can not match the new properties of MUIs. Except for a cyclic manual regulatory definition that requires considerable effort and time. Researchers adopted the genetic algorithms (GA) to produce detection rules to cope with the problem of rules interpretation. GA is a family of algorithms for optimization that works to find solutions to complex issues. It is considered to be a highly ideal search method to pursue the best resolution of problems with a large number of possible solutions [20]. The genetic algorithms with multi-objectives are focused on satisfying two competing objective functions. The GA has been used extensively in identifying code smells in software quality domains, which are internal poor design issues that may impede the quality of the application [14, 22, 54]. Shoenberger et al. exploited the genetic programming to generate detection rules for bad code practices committed in JavaScript projects [45]. Along the same line, the multi-objective genetic algorithm (MOGP) was used by Kessentini and Ouni to find the best set of detection rules covering the higher number of bad code practices. Their method hit an average of more than 82% precision and a 77% recall [18]. In our research, we use the multi-objective genetic algorithm NSGA-II to produce the detection rules.

3 BACKGROUND

3.1 Metrics definition

Metrics based assessment methods are designed to quantify the quality of user interface designs. Our ADDET tool implements 15 metrics including 14 metrics that ¹ were selected from various studies [17, 33, 34]. Our method has the most significant number of calculated metrics in the graphical user interfaces assessment area, to the best of our knowledge. The effort to compute

¹All the 14 selected metrics formulae can be found in the appendix

several metrics is to satisfy most of MUI’s structural properties before a defect is triggered. We describe 12 state-of-the-art metrics in Table 1 and Table 2 that could be demonstrated with a MUI prototype. All left prototypes show non-violation of the corresponding metric property. As for the three metrics remaining, we define their purpose as follows:

- **Nb-Elements:** It computes the number of components in a single MUI [41]. To normalize this metric, we evaluated a set of 80 MUIs and selected the minimum and the maximum number of components. Then, we computed the normalization equation (1). with: margin=[0,1], m1=1, m2=0.

$$normalize(metricValue) = (m1 - m2) * \left(\frac{metricValue - min}{max - min} \right) + m2 \tag{1}$$

- **Complexity:** This metric is the sum of all the computed metrics in this analysis, as described by [33]. It provides a relative score about the composition of the MUI, taking into account the variability of all structural aspects. Complexity metric is a descriptive metric and is therefore not used in any of our study’s statistical measures.
- **Clarity:** The MUI’s clarity relates to how comprehensible the interactive elements are. In reality, Hannah Alvarez [3] has shown that only 60% of users have successfully predicted the functionality of an unlabeled icon, while 88% of the time, users have been able to correctly anticipate what will happen when they tap a labeled icon. Thus, we provide the *Clarity* metric which counts the number of *android:text*, *android:title*, and *android:hint* label attributes by the number of buttons, menu items, and editTexts elements. Equation (2) provides the metric formula.

$$CL = \frac{\sum_{i=1}^{i=n} (label + title + hint)}{\sum_{i=1}^{i=n} (button + item + EditText)} \tag{2}$$

In Figure 1, we present an illustrative example portraying the issue of a comprehensible MUI against a non-comprehensible MUI.

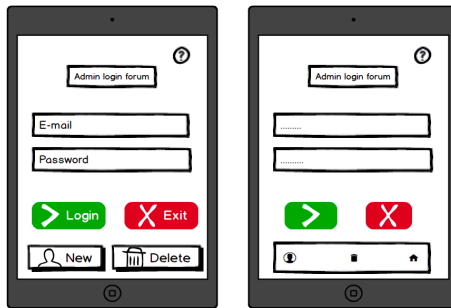


Fig. 1. A comparison of clarity level: (left) clear navigation ; (right) difficult navigation.

Table 1. Metrics definition with an illustrated prototypes

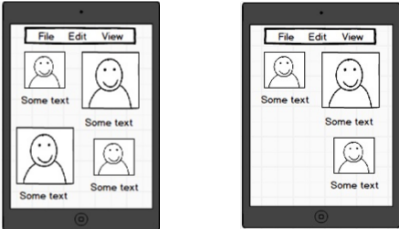
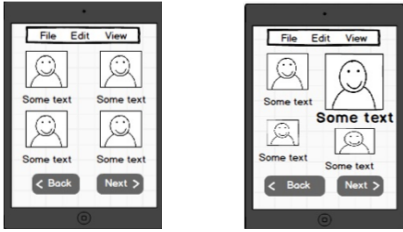
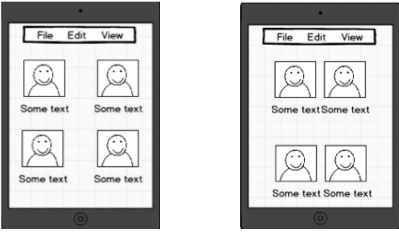

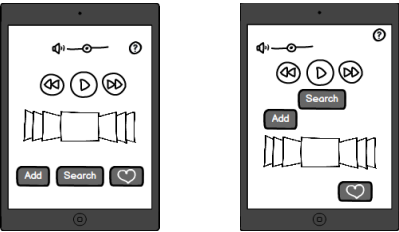
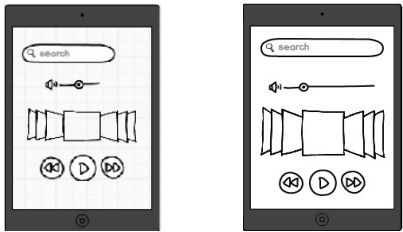
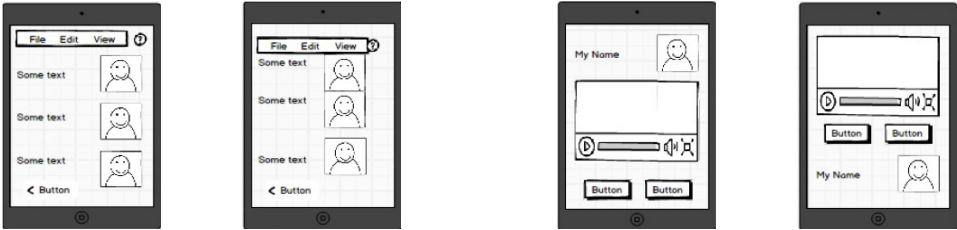
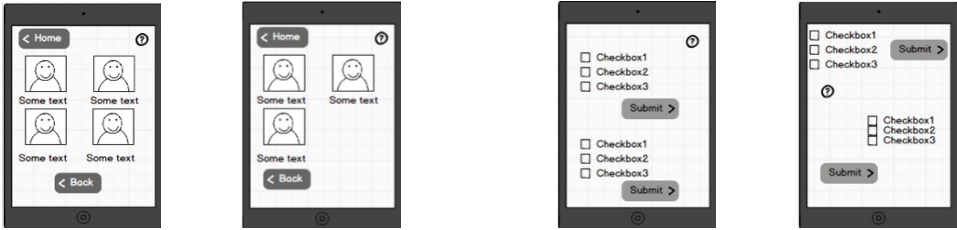
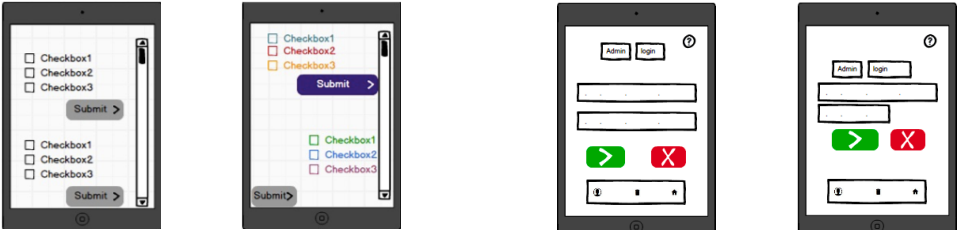
Metrics	
<p>1-Balance</p> <p>It provides an equal distribution of components weights between the different parts of a MUI. This metric is suggested by [33].</p> 	<p>2-Economy</p> <p>It consists of minimizing the different components sizes to display the information in a simple and clear way. This metric is proposed by [33].</p> 
<p>3-Cohesion</p> <p>It is a measure of the degree of conceptual inter-relatedness of components parts. It is based on promoting similar aspect ratios, which refers to the relationship between heights and widths. Cohesion is achieved by maintaining a consistent aspect ratio of screen elements. This metric is suggested by [10].</p> 	<p>4-Layout Uniformity (Uniformity)</p> <p>It measures the degree of consistency and orderliness of a MUI layout. It calculates the layout of different components "height, width, alignment points from 4 edges". This metric is suggested by [36].</p> 
<p>5-Simplicity</p> <p>It measures how many elements are situated on the same X and Y axis. It is seen as a trade-off between optimizing the number of elements and screen overload. This metric is suggested by [33].</p> 	<p>6-Density</p> <p>It measures the screen area occupied by objects. The aim is to minimize the density level of occupied area displayed to the user. This metric is suggested by [33].</p> 

Table 2. Continuity for Table 1:Metrics definition with an illustrated prototypes

Metrics	
7-Regularity	8-Sequence
<p>It consists of providing consistent spacing between MUI elements based on horizontal and vertical alignments to organize the layout structure. This metric is suggested by [33].</p>	<p>It aims to organize MUI elements from top-left to bottom-right corner. This metric is suggested by [33].</p>
	
9-Homogeneity	10-Grouping
<p>It measures the degree of evenness of objects distribution among the four parts of the screen. The quadrants should contain a more or less equal number of elements. This metric is suggested by [33].</p>	<p>It aims to strengthen the attraction between widgets by grouping them at the center of the MUI. It is the degree of making widget groups be as one unit. This metric is suggested by [10].</p>
	
11-Unity	12-Integrity
<p>It is the extent to which the screen seems homogeneous. Unity tends to unify the MUI elements into the same shapes, same colors, and the same size. For ease of development, we exclude the color from the formula. This metric is suggested by [10].</p>	<p>It aims to minimize the use of different widgets sizes on the MUI, and that of the space between the area occupied by components and the frame area. We matched the lengths of the TextFields to get same margin distance. This metric is suggested by [33].</p>
	

3.2 Structural aesthetic defects

To select the set of our structural defects, we looked for the ones that have been widely discussed in previous studies [8, 9, 16, 17, 44].

- **Imbalance of MUI (IM):** It represents the unequal distribution of the components on the four parts of a MUI layout. Figure 2 (left) provides an example of an imbalanced MUI.
- **InCohesion of MUI (ICM):** It is the lack of the inter-relatedness of MUI components. Figure 2 (middle) provides an illustrative example of a triggered InCohesion defect.
- **Overloaded MUI (OM):** From an end-user point of view, the overloaded MUI defect is detected when the user sees a high number of components on the MUI. Figure 2 (right) provides an illustrative prototype.

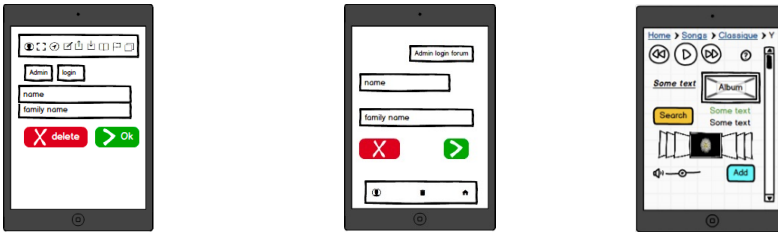


Fig. 2. An illustration of an Imbalanced MUI (left), an In-cohesive MUI (middle) and an overloaded MUI (right)

- **Incorrect Layout of Widgets (ILW):** From an end-user perspective, when the widgets have a high number of different layouts, the ILW defect is detected. That is, the widgets are placed in different rows and columns. Figure 3, offers an illustrative illustration of defective widget layout with the right layout version to differentiate the difference between the ILW and the ICM defects.



Fig. 3. An illustration of Incorrect layout defect: (left) Incorrect layout ; (right) correct layout.

- **Difficult Navigation (DN):** It entails the lack of descriptive labels associated with components to describe its functionality. Figure 1 provides an illustrative prototype of a DN defect.

4 TOOL IMPLEMENTATION

The tool ² is a part of a framework currently being developed to provide an automatic evaluation, recommendation, and restructuring support of Android MUIs graphical components. ADDET (Aesthetic Defects DEtection Tool) is a Java-based evaluation tool developed to assess Android

²All the assets used in this study: tool, experiment results, questionnaires, MUIs, etc, are available at https://github.com/NarjessBessghaier/Defects_Detection_Tool

mobile user interfaces. The tool evaluates native apps as well as hybrid apps wrapped into apk format³. The only downside with hybrid apps is that the simulator reads all elements of the MUI as views. That is, we can not tell the different types of active widgets. Therefore, the metrics of grouping and unity does not determine an accurate value. Our tool includes three modules, as shown in Figure 4: 1) parser, 2) metrics calculator, and 3) detector defects.

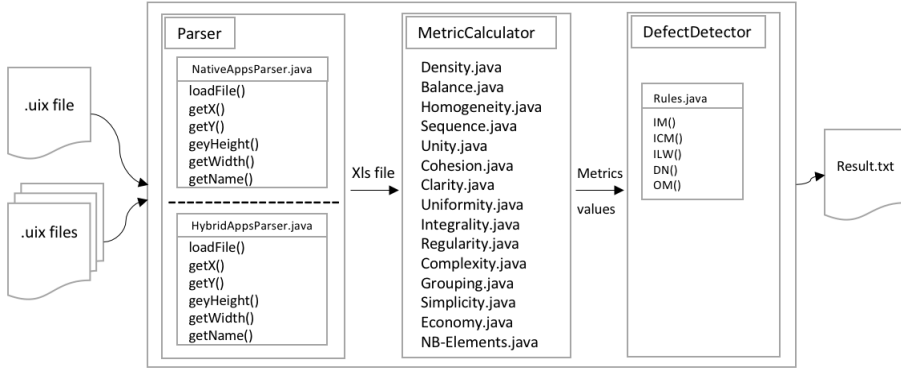


Fig. 4. Modules of the proposed tool ADDET

4.1 STEP1: Extraction of MUI graphical data

To extract data from MUI components, scholars have used several computer vision techniques that include methods for acquiring, processing and analyzing image dimensional data [41, 57]. In this study, we use a simple yet effective way to extract geometrical data accurately from a MUI. Our extraction process consists of running the application via the Android studio packaged tool *UIAutomatorViewer*. The tool leverages the Android Debug Bridge (ADB)⁴ server to generate the UI layout-hierarchy, allowing to inspect and retrieve the properties of the on-screen UI components. The *UIAutomatorViewer* allows us to save a dump file⁵ that encompass all components' geometric data.

4.2 STEP2: Parsing the dump file

The dump file provides the properties of all visible and invisible components such as checkable, clickable, enabled, focusable, scrollable, selected, etc. All components names are declared in a "`<class="android.widget.NAME">`" tag and the components properties are declared in a "`<bounds="[x,y][w,h]">`" tag. The views elements of wrapped hybrid applications are declared in a "`<class="android.view.View">`" tag. Thus, our hybrid apps parser extracts only the View class corresponding bounds. The native apps parser matches the regex patterns and retrieves all content in between the widget's tags. To use the parser, the user needs to specify the path to his/her dump file. As presented in Figure 5, an Xls file will be generated, including all types of components with their geometrical values. Then, we remove all tags related to *RelativeLayout*, *LinearLayout*, and *FrameLayout* components as they are invisible containers.

³apk: It is a file format for Android operating system applications.

⁴The ADB server is a plugin used in Android application development (Android studio), where the developer can run his/her application on in either an emulator instance or in the plugged-in device.

⁵A dump file with the extension (uix) is used to save the UI layout-tree from the debuggable application. After inspecting the MUI elements the file could be saved via the *UIAutomatorViewer* emulator. It includes the MUI tree with all the geometric data of each element, with additional attributes such as 'text,' 'disabled,' 'clickable,' etc.

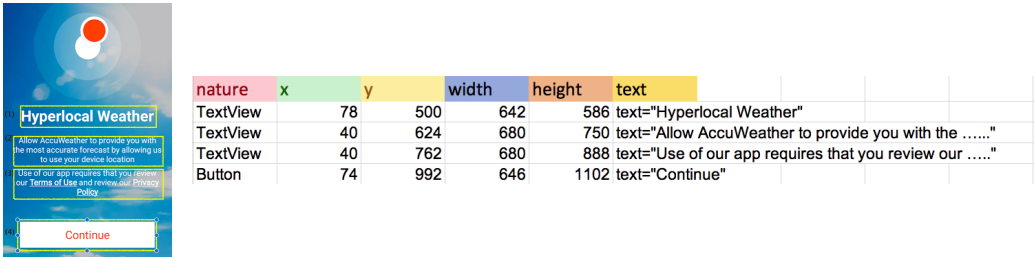


Fig. 5. Extraction of a MUI components geometrical data of the application AccuWeather V6.0.8-free

4.3 STEP3: Metrics measurement

ADDET assesses the structural quality of the MUI based on a set of 14 metrics (we excluded the complexity metric). Such metric values are measures of other structural defects. A metric value can mean one or more defects are present. Thus, we need to equate these values with an acceptable threshold in the inspection process. The boxplot technique allows us to generate a threshold after calculating the min, max, and median of the values set. We evaluated a list of 11 applications of different categories having 80 MUIs. Figure 6 shows the generated thresholds of our proposed metrics.

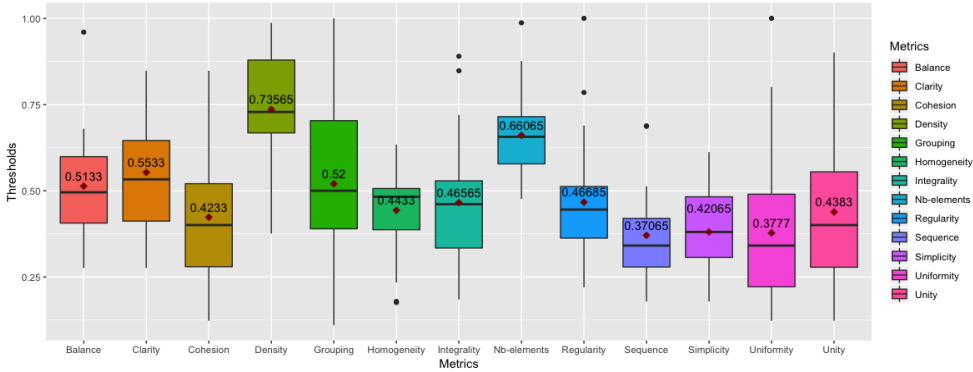


Fig. 6. Metrics values calibration using box plot technique

4.4 STEP4: Defects detection

This method consists of creating rules for MUI evaluation detection. Due to a large number of potential combinations, Ines et al. used the technique of mono-objective genetic programming (GP) for the generation of rules combining different concepts: metrics, demographics of user-profiles and experience information, and usability defects [16]. In our context, we omitted the adaptation of the user profile for two reasons. 1) *Standard design problems*: we seek to define design problems independently of the users' perception of the design. 2) *Restructuring*: as a potential direction, we plan to restructure the structural aesthetic defects of the MUI. It is not an easy process if we aim to fix every defect based on specific user profiles. We used the same genetic programming methodology to produce assessment rules for our module of identification of defects. In our research, the added values are: 1) the use of multi-objective genetic programming (NSGA-II), taking into account two opposing objectives; 2) assigning a weight to each metric to demonstrate its significance in the

identification of a defect. It is necessary to apply the genetic algorithm to know which set of metrics is associated with a specific defect. The process has three steps: 1) the development of a base of examples for the NSGA-II training; 2) the adaptation of the NSGA-II to our context; 3) the metrics weight assignment.

4.4.1 Creation of the base of examples.

This process consists of building a base of examples to train the genetic algorithm to improve the precision of its defect extraction. This training gives us the correct rules for the assessment. The basis of examples is conceived as a pair of < MUIs, structural aesthetic defects >. The first segment consists of the name of the evaluated user interfaces, and the second segment consists of the user opinion on potential structural aesthetic defects of the evaluated MUI. This step is achieved through a Google form questionnaire, in which 20 participants obtained five well-explained structural aesthetic defects and are asked to assess a collection of 80 MUIs. We have divided the MUIs into five questionnaires, each of which has 10, 10, 20, 20, and 20 MUIs because the number of user interfaces is very high. The user-profile diversity allows the genetic algorithm to train and to return the best rules. The base of examples contains user interfaces, which were tested manually to identify potential structural defects. The MUIs are tested in an iterative manner during the training stage of the algorithm to pick the most frequently identified aesthetic defects and the set of metrics. The user profile and the metrics are independent variables since these are automatically determined by the tool, so there is no intervention from the user side in the measurement of these metrics (see subsection 4.4.2 for more information on combinations of defects so metrics).

Selection of Applications: We have filtered 510 Android open-source software hosted at GitHub. Our application selection process is designed to ensure the app has a high user rating in the Google Play store. In Table 3, we provide the number of user interfaces analyzed and the characteristics of our applications.

Table 3. Android applications involved in the study.

Name	Category	# MUI	User rating
GnuCash	Finance	8	4.4
2048-app	Game	2	4.7
Tusky	Internet	8	3.9
Shuttle Music	Music	10	4.3
C-Geo	Navigation	14	4.4
SimpleGallery	Photos&Videos	11	4.5
Telecine	Photos&Videos	2	3.7
SoundRecorder	Utilities	2	4.2
AmazeFileManger	Utilities	9	4.3
Orgzly	Writing	8	4.7
NewPipe	Social Networking	6	4.2

Subjects: In this study, we invited twenty students of different ages, experience levels with software quality, and educational level (70% females, 30% males). We had 30% Ph.D. students (3 students in the field of software quality, one student in multimedia design and two students in the field of machine learning), 20% students in computer science with a bachelor's degree, and 50% students in the field of master's degree. The data collected about the participants indicate that only 55% of respondents have software quality assessment experience. The subjects were asked to assess the MUI screen-shot's structural quality. Therefore, no familiarity is needed with the application. We give the characteristics of the participants in Table 4.

Table 4. Participants characteristics

Educational level			Age		Sex	
Ph.D	Master	Bachelor	21-25	26-30	M	F
6	10	4	14	6	6	14

4.4.2 NSGA-II Adaptation.

Using a multi-objective genetic algorithm (NSGA-II), we generate evaluation rules as an optimisation problem. The GA method starts by describing a problem space as a set of attributes (inputs) that define potential solutions to a problem. The genetic algorithm carries out random combinations of these attributes within this model space, and tries to find the best combination to produce the highest fitness measure. Growing population solutions are tested via a fitness function to determine its efficiency in detecting the defects. These solutions subsequently fall under the control of the genetic programming reproduction operators, which are *mutation* and *crossover*. These operators strive to fine-tune the collection of solutions to keep those with the best fitness interest creating a new generation of rules. This cycle is repeated for a fixed number of generations until the most suitable individual is identified. In the search area, the quality of the rules generated is calculated by the use of two opposing objective functions (fitness functions).

Fitness 1: Maximize the number of detected defects in the rules. It compares from the examples base the list of observed defects in the rules and the expected ones. Equation (3) represents our fitness function.

$$\text{Fitness1}(s_i) = \frac{\sum_{j=1}^{\text{size}} \left(\frac{\text{NbR}_j}{\text{NbOcc}_{j \text{ in } D}} \right)}{\text{NbT}} \quad (3)$$

With

NbR: Number of rules detecting defect j, NbOcc: Occurrence of defect j, D= dataset, NbT: Number of defects type.

Fitness 2: Minimize the number of rules. i.e, return the solution with the minimum number of rules. Equation (4) represents our second fitness function.

$$\text{Fitness2}(s_i) = \min \quad \text{number-of-rules} \quad (4)$$

Input: In our settings, the genetic algorithm takes as its inputs a base of examples containing some manually inspected projects, 13 structural metrics (we exclude the complexity metric as it is a descriptive metric, as well as the clarity metric as conceived for a specific defect) and four structural aesthetic issues excluding the DN defect (see section ??). As well, a threshold between [0, 1] and an operator (" \leq ", " \geq ") are randomly assigned to the metrics.

Rules: Our goal is to combine the collection of metrics pertinent to a specific defect automatically. According to the definition of metrics, we presume that the maximum number of metrics that can detect one defect is about 6. So we created six rule classes that contained 1 to 6 metric combinations, respectively. Then, we ask the NSGA-II to randomly pick a rule of class during the process of rule generation. The logical statements ("AND, OR") are assigned to all rules at random.

Output: The process output is a rule-tree structure, where we consider rules of one to six metrics. We choose the combination that is true for each defect, based on the metrics values of each user interface.

The NSGA-II executes several combinations among the inputs to produce the best rules. The best of them are the ones which adhere to the base of examples. Our aim is to increase the number of detected defects in the assessment rules and reduce the number of rules with maximum defects (see

section 4.4.2 for more details). In particular, to generate a set of rules which have a good ability to detect the maximum number of defects in the base of examples. To this end, a heuristic search for the best set of combinations is performed on the base of examples. Our proposed algorithm has four steps : (1) initial population generation, (2) selection of individuals, (3) operators of reproduction; and (4) new individuals' generation.

4.4.3 NSGA-II reproduction operators.

This step is based on two fundamental genetic operators: crossover and mutation.

- **Crossover:** During crossover, we create new individuals (offspring) by the selection operator combining portions from the selected individuals (parents). The algorithm selects the two-rule cut-point, recombining the four parts into two new rules. Subsequently, both old rules (parents) and new rules (offspring) pass to the population's next generation.
- **Mutation:** It consists of making some modifications over the newly created rules from the crossover to maintain the diversity within the population. In our setting, the mutation operator changes the metrics, the value of the metrics, or the arithmetic operators.

The high-level pseudo code of the NSGA-II is given in Figure 7.

Input: M: metrics; D: defects; N: solutions size/population; R: max rules/solution; R,R1,R2,R3,R4,R5,R6: rules shape; B:base of examples; MV: matrix of metrics values

```

1. while (i<N) do
2.   while (i1<R) do
3.     rule <- ChooseRandomRuleClass(R,R1,R2,R3,R4,R5,R6)
4.     GenerateRandomMetricsValues()
5.     GenerateRandomOperators()
6.     S<- CreateRandomRule(M,D,R)
7.     P<- S
8.   until i1 = R
9.   until i = N
10. while (i<N) do
11.   while (i1<R) do
12.     detectedDefect<- defectInRule(rule)
13.     expectedDefect <- ExistInBaseOfExample(B)
14.     metricValue <- ExistInMatrix(MV)
15.     fitness1<- EvaluateFitness1 (S, detectedDefect, expectedDefect)
16.     fitness2<- EvaluateFitness2 (R, minRulesSize)
17.     Sort (P,fitness1,fitness2)
18.     individual<-BinaryTournament(P)
19.     intermediateRules<- SinglePointCrossove(individual)
20.     newRules<- BitFlipMutation(intermediateRules)
21.     P<- Merge (intermediateRules,newRules)
22.   until i1 = R
23.   until i = N
24. bestSolution <- ChooseBestSolution(fitness1,fitness2)
25. return bestSolution

```

Output: bestSolution

Fig. 7. Pseudo code of adapted NSGA-II algorithm

While the number of solutions per population and the number of rules per solution are not yet reached, NSGA-II randomly selects a set of rules from among the six classes given (line 3). The algorithm produces random values for the metrics in lines 4 and 5, and associates them with random operators. The algorithm allows an initial population (a set of solutions) until all of the

rule variables are ready. Each of these solutions includes a set of assessment rules (line 6) and incorporates the solution generated within the population (line 7). The algorithm checks, from line 12 to 14, if the present defects in the rules exist in the base of examples. It also checks whether the given random metric values are in the matrix (a matrix is declared in the algorithm containing the metrics values for the 80 evaluated MUIs). For any solution, the algorithm tests the fitness functions in lines 15 and 16. NSGA-II then sort out every solution based on fitness values (line 17). The algorithm begins its optimization from lines 18-20 by applying its crossover and mutation operators to produce new rules. The previous stage for the operators of reproduction is the exclusive selection that gives priority to qualified individuals and discards the corrupted individuals. This step helps the individuals to pass on to the population of the next generation (line 18). For the next generation a new population (P) with the new merged rules (line 21) is created, and the best solution is saved (line 24). The algorithm ceases when it completes the maximum iteration number and returns the best set of evaluation rules (line 25). The parameter setting for all executions of the NSGA-II is specified in Table 5.

Table 5. NSGA-II parameters calibration

Parameter	Score
Population size	50, 100, 200
Mutation probability	0.1, 0.2, 0.5
Crossover probability	0.9, 0.8, 0.6
Max generations: stopping criterion	100, 200, 500
Numbers of rules in populations	20-30

After several algorithm executions, we select the generation with the best values for fitness functions (the minimum set of rules which detect most defects). We present the obtained results from the NSGA-II in Table 6.

Table 6. Metrics combination for each defect

Defects	Density	Economy	Integrity	Nb-elements	Sequence	Regularity	Simplicity	Homogeneity	Uniformity	Balance	Unity	Cohesion	Grouping
OM	X	X	X	X									
ILW					X	X	X	X	X				
ICM					X						X	X	X
IM								X		X			

4.4.4 Weighting Metrics Importance.

We are interested in assigning a weight to each metric to illustrate its importance when detecting a structural aesthetic defect. To this end, we use the WEKA framework to manipulate such weighting algorithms. WEKA [15] is a software that gathers a set of different machine learning algorithms. It offers a collection of pre and post-processing tools as well as some methods for evaluating the result of learning techniques applied to any given data set. Besides the classification and association algorithms, WEKA implements some features selection and weighting algorithms.

Wrapper: It considers the subset of features with which the classification algorithm performs the best. These wrappers focus on revealing the essential elements by assigning a rigid weight of 0 and 1. The wrappers are classified as feature selection algorithms.

Filter: It uses an attribute evaluator (correlation, infoGain, etc.) in association with a ranked

algorithm to rank some/all features in the data set (Best first, Ranker, etc.). The filters differ from the wrappers in terms of weight value. Some of the filters would rank the features in an ascendant order (high to low), and some others allow a finer differentiation between the features by assigning each a valued weight ranged between 0 and 1. Then, the feature degree of importance is mirrored in the magnitude of its weight. Among the available ranking algorithms, we chose the *InfoGainAttributeEval* (IGAE) to assign weights to our metrics and the *CorrelationAttributeEval* (CAE) to compute the correlation between the metric and the defect using the Pearson correlation. Metrics with weights equal to 1 are separated with an OR operator to ensure a defect can be triggered by one measure. Operator AND is assigned to metrics with weights below 1. It allows us to ensure satisfaction of the metrics threshold before a defect is triggered. For ILW and ICM defects, we declared two conditions to prioritize the metrics with middleweights. Table 7 summarizes the structural aesthetic defects detection rules as proposed in this work.

Table 7. Detection rules for Android MUI structural aesthetic defects

Defect	Metrics combinations
OM	1-Nb-Elements \geq 0.660 OR Density \geq 0.735 OR (Economy \leq 0.365 AND Integrality \leq 0.465)
ILW	1-Regularity \leq 0.345 OR (Uniformity \leq 0.377 AND Homogeneity \leq 0.478) 2-Regularity \leq 0.345 OR (Simplicity \leq 380 AND Sequence \leq 370)
ICM	1-Cohesion \leq 0.423 OR (Unity \leq 0.438 AND Sequence \leq 0.370) 2-Cohesion \leq 0.423 AND Grouping \leq 0.505
IM	1-Balance \leq 0.513 OR Homogeneity \leq 0.478
DN	1-Clarity \leq 0.553

5 VALIDATION

5.1 Research Questions

To evaluate the reliability of the proposed tool ADDET, we formulate the following research questions:

- (1) **RQ1:** To what extent does ADDET behave as/better than QUESTIM and PLAIN?
 $H1_0$: There is no difference between ADDET and QUESTIM's shared metrics values.
 $H2_0$: There is no difference between ADDET and PLAIN's shared metrics values.
- (2) **RQ2:** To what extent is ADDET accurate in detecting most defects?

To answer research question 1, we conducted a comparative study with two existing tools QUESTIM [57], and PLAIN [47] (see sub-section 2.2 for more information about the tools). The analysis includes evaluating a common set of metrics used to measure the structural quality of MUIs.

To answer research question 2, we conveyed an experiment with five experts in order to determine the tool's accuracy in detecting most of the defects.

5.2 RQ1: To what extent does ADDET behave as/better than QUESTIM and PLAIN?

5.2.1 Study design.

The study's goal is to verify whether the proposed tool could use a set of metrics to reliably describe different structural aspects of a MUI. The metrics chosen are diversified and discuss various MUI representational angles. Therefore, the metrics are intended to help developers refine a MUI design. We compare the results of a specific sub-set of metrics against QUESTIM and PLAIN. The distinction is based on an evaluation of 80 MUIs. We present the technical features of PLAIN, QUESTIM, and

Table 8. Tools characteristics

Criteria	PLAIN	QUESTIM	ADDET
Computes visual metrics	Yes	Yes	Yes
Requires source code	No	No	No
For mobile apps	Yes	Yes	Yes
Requires regions drawing	No	Yes (Manual)	No
Number of metrics	8	11	15
Structural defects detection	No	No	Yes

ADDET in Table 8.

Our tools comparison is conducted as follows:

- Balance and Simplicity metrics comparison between ADDET and QUESTIM .
- Homogeneity, Sequence, and Integrality metrics comparison between ADDET and PLAIN.
- Density metric comparison between ADDET, QUESTIM, and PLAIN.

All shared metrics between ADDET, QUESTIM, and PLAIN use the same formulae proposed by Ngo et al. [33]. The calculated scores vary mainly because of the various ways of obtaining geometric data for the components. We remember that the invisible ViewGroups and different layouts are considered by Soui et al. [47] as graphical components. Zen and Vanderdonkt [56, 57] extract geometric data of the components based on the regions manually defined. i.e, any difference in a region shape would return different X, Y, Width, and Height.

5.2.2 Statistical results.

To determine the degree to which the proposed tool produces the same or better results as QUESTIM and PLAIN, we perform the paired parametric t-test as the data collected always display a normal distribution, with $df=95$ for a level of significance of .05. We are aggregating data from the 80 MUIs, taking into account metrics averages for the sake of clarity. We reject the null hypothesis if the p-value is higher than .05. That is to say, ADDET did not get the same results. Thus, we will review the metrics score manually to know if the method has conducted a correct or incorrect evaluation. The results of the three comparisons are respectively presented in Tables 9, 10, and 11 with M refers to the mean, Std =standard deviation, and P =p-value.

For clarification, in Figure 8, we present three evaluated MUIs in the QUESTIM evaluation tool with their selection of corresponding regions.

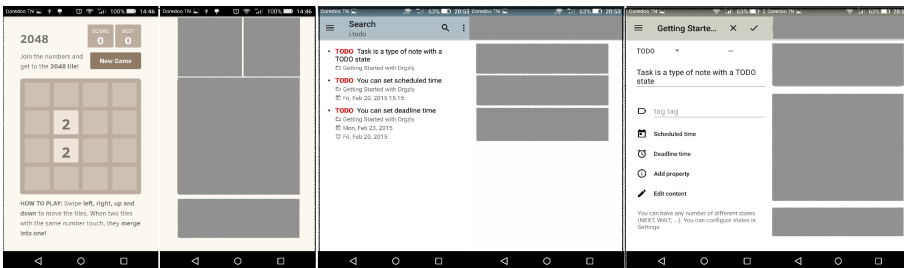


Fig. 8. 2048-app (left), Search (middle), and to do (right) MUIs with their framed regions in QUESTIM

Comparison1: Results & Discussion

Balance: The results obtained from the t-test ($M=.831$, $Std=.126$), as shown in Table 9, suggest that

there was no substantial difference between the tools ADDET and QUESTIM in the calculation of metrics ($t=.588$, $P=.573$). QUESTIM is rating the MUIs as highly balanced, given the regions represented in a balanced way, is not a surprising result.

Simplicity: The results of the t-tests ($M=.256$, $Std=.187$), as shown in Table 9, resulted in $t=-1.631$ and $P=.14$ of both tool simplicity metric values. These results point to the fulfillment of the H_{10} null hypothesis. That is, in determining the simplicity of the MUIs, ADDET obtained the same results as QUESTIM. We need to evaluate the returned scores cautiously, however. QUESTIM considers the MUIs are easier than ADDET. In precise words, depending on the regions that the user draws manually, he/she determines the user interface structure where other components can be skipped accidentally. On the contrary, ADDET calculates a higher number of components, resulting in a reduced level of simplicity. As we are looking for similar metrics evaluation between ADDET and QUESTIM, the correct condition is considered to be H_{10} . There is no significant difference in the p-values $>.05$, which shows no evidence of the alternative hypothesis's validity. However, it suggests that ADDET obtained the same outcomes as QUESTIM to some degree and therefore conducted a thorough evaluation.

Table 9. Comparison 1: ADDET vs. QUESTIM

Metrics	ADDET		QUESTIM	
	M	t-val	Std.	P
Balance	.831	.588	.126	.573
Simplicity	.256	-1.631	.187	.146

Comparison2: Results & Discussion

As shown in Table 10, the same findings are noted with PLAIN, where no evidence supports the validity of the alternative hypothesis; and thus, we do not hesitate to accept H_{20} for the three metrics with ($M=.606$, $Std=.109$), ($M=.689$, $Std=.091$), and ($M=.414$, $Std=.161$) respectively. The findings for the metrics of homogeneity, sequence, and integrality ($t=.348$, $P=.726$), ($t=1.456$, $P=.214$), and ($t=-1.665$, $P=.151$), respectively, suggest that both ADDET and PLAIN performed the same MUI assessment.

Table 10. Comparison 2: ADDET vs. PLAIN

Metrics	ADDET		PLAIN	
	M	t-val	Std.	P
Homogeneity	.606	.348	.109	.726
Sequence	.689	1.456	.091	.214
Integrality	.414	-1.665	.161	.151

Comparison3: Results & Discussion

In both comparisons, as shown in Table 11, the t-test results between ADDET/QUESTIM and ADDET/PLAIN ($M=.731$, $Std=.059$) and ($M=.339$, $Std=.101$) suggest the presence of a difference in density metric measurement ($t=-3.080$, $P=.023$) and ($t=3.192$, $P<.01$), respectively. Therefore the null hypotheses H_{10} and H_{20} are not applicable. We must note that each component area is calculated by the density metric and compared with the frame area. QUESTIM calculates the metrics using the X, Y, Width, and Height zones. PLAIN treats the layouts and ViewGroups as elements, while ADDET considers only the visual components on the screen. The different number of calculated graphical elements yielded a different assessment of density. The accuracy of the ADDET density metric calculation is confirmed by the fact that the tool collects only the geometrical data of the

visual components based on the inspection carried out by the *UiAutomatorViewer*, which has no margin of error.

Table 11. Comparison 3: Density metric comparison between ADDET, QUESTIM and PLAIN

Tools	Density metric			
	M	t-val	Std.	P
ADDET/QUESTIM	.731	-3.080	.059	.023
ADDET/PLAIN	.339	-3.192	.101	<.01

As a validation for the left six metrics, and clarity's sake, we qualitatively analyze the metrics results for a set of the three MUIs (Figure 8) as presented in Table 12.

2048-app MUI: Visual perception: The components are located in the middle of the user interface as shown in Figure 8. However, the left/right margins of the white space are more prominent than the top/down margins. The MUI has over four widget styles, with six different sizes. *ADDET:* As presented in Table 12, the tool measured scores are considered representative of the structural aesthetics aspects of the MUI. For instance, the economy is estimated too low, and the existence of different components resolutions justifies it. The Unity metric is around 0.2, as there are different types of components and different margins spaces.

Search MUI: Visual perception: The MUI presented in Figure 8 groups all components in the upper center side. However, all components have the same size and type, along with the same space between the elements. *ADDET:* As presented in Table 12, we notice a change in the metrics values with a low regularity level. The regularity metric measures 1) the extent to which the alignment points are minimized, and 2) the extent to which the alignment points are consistently spaced. The regularity score will decrease as we observe a large white space from the components group to the bottom margin. The grouping is considerably high thanks to the consistent space, size between the components. While the cohesion and uniformity metrics are estimated low (around 0.3) due to inconsistency of white space between the layout margins.

To do MUI: Visual perception: As presented in Figure 8, this MUI is considered to be structurally pleasing. It is cohesive (same white space between the components, components are positioned in the center of the frame), uniform (more or fewer components have the same size), regular and grouped. *ADDET:* As presented in Table 12, the metrics computation of cohesion, uniformity, regularity, and grouping entail a correct evaluation of the MUI with a score of more than 0.6. The economy level is considered medium with 0.4 and unity of 0.2. We recall that unity is a delicate metric that computes the number of different components sizes and types and evaluates the relative measure of the white space between components groups and that of the margins.

Table 12. Metrics computation for the 2048, Search, and To do MUIs

Metrics	Mobile User Interfaces		
	To do MUI	2048 MUI	Search MUI
Cohesion	0.835	0.862	0.615
Uniformity	0.831	0.849	0.577
Regularity	0.611	1.0	0.208
Economy	0.403	0.05	0.143
Unity	0.233	0.293	0.188
Grouping	1.0	1.0	1.0

5.3 RQ2: To what extent is ADDET accurate in detecting most defects?

5.3.1 Study design.

The goal is to determine to what degree the proposed tool can detect most defects in a MUI. We recall that our detection rules are restrictive as triggering one defect requires assessment and satisfaction of multiple thresholds for structural properties. A set of 24 out of 80 measured MUIs are rated as having three or more structural defects. To improve the tool's applicability, we tested 500 additional MUIs from the [12] RICO data collection, where 282 MUIs were holders with three or more structural defects. To validate the accuracy of the tool in detecting the most number of defects, five experts (1 Ph.D. student in the domain of software quality, and four engineers in 3D/2D graphics design from the PALM 3D studio in Tunisia) evaluated the MUIs. The experts had no previous idea about the types of the 306 selected MUIs in the study and were asked to 1) manually analyze the quality of a MUI and explain the triggering of a defect and 2) compare the results with the results detected by ADDET. It took the experiment about 20 days to collect the findings of all the experts with an average of 17 MUIs per day. We asked the experts to give +1 for each correct detected defect (true positive), -1 for incorrectly detected defects (false positive), -2 for incorrect detected defects (false negative), and -3 for correct non detected defects (true negative). Therefore, ADDET was evaluated on the basis of a series of evaluation criteria widely used to test the efficiency of machine learning models.

- True positive (TP): if a truly existent defect is detected it is called TP;
- False positive (FP): however, if a non-existent defect is detected; it is called FP;
- True negative (TN): if a truly existent defect is not detected it is called TN;
- False negative (FN): it is the \bar{TP} , where a non-existent defect is not detected.
- Recall: it is the proportion of the truly existent defects that are correctly detected. It describes the coverage of the tool in detecting the truly existent defects. The recall is computed as follows:
Recall= True positive/(True positive+ False negative);
- Precision: it is the fraction of correctly detected defects from the set of detected defects (true positive + false positive). The precision is computed as follows:
Precision= True positive/ (True positive+ False positive);
- F1 score: it is the average of Precision and Recall. It takes both false positives and false negatives into account. The F1 score is computed as follows:
F1 score= 2*(Recall * Precision) / (Recall + Precision).

As shown in Table 13, and out of 4992 detected defects, ADDET has scored an almost 71% accuracy in detecting genuinely existing defects. A 63% F1 score is considered suitable as an average of correctly detected defects resulting from actually existing defects and correctly detected defects resulting from all detected defects. Based on the expert's assessment and informative descriptions, we have found that: 1) the experts consider the ICM defect the most easily identified. 2) Just 11.18% of DN defect instances were accepted by the experts, who clarified that most of the icons used are common and thus understandable. 3) The defects of ICM and ILW are often co-approved by experts (co-exist in MUIs). 4) IM and ILW defects are not found together often. An imbalanced MUI does not necessarily mean the components are not located in incompatible cells of the layout. The left/right margins, the component size, and the padding may be different. Consideration by the expert promotes focusing on the co-occurrence of defects.

6 DISCUSSION

Regarding **RQ1: To what extent does ADDET behave as/better than QUESTIM and PLAIN?** We evaluated ADDET's metrics module against two existing tools, QUESTIM and PLAIN. Because

Table 13. ADDET's performance in detecting most defects

	TN	FN	TP	FP	Precision	Recall	F1 score
#	113	2545	3525	1467	0.7061	0.5807	0.6343
%	2.26%	51%	71%	29.3%	71%	58%	63%

of the diversity of the used metrics, we only assessed the set of common measures. The results suggest that the tools computed the same metric values approximately, except for the density metric where ADDET performed better than QUESTIM and PLAIN. We have a common number of components for most MUIs-X, Y, Length, and Height. Therefore, the density metric measures a higher value of the number of occupied area components than the container area. Consequently, a higher value of the density is returned. With respect to the complete validation, specific points have to be cleared. The selected six metrics are considered delicate, as they assess some critical aspects such as "different spaces between components, different types, and sizes of widgets, different occupied areas." The remaining six metrics (excluding complexity, clarity, and Nb of elements) such as Cohesion, Unity, Grouping, Uniformity, Regularity, and Economy are tackling the aspects of: "position of each widget type, number of different heights and widths, number of rows and columns, etc." The results of the metrics have provided an appropriate assessment of the structures of the MUIs. And we can conclude that the findings obtained gave us an indication of ADDET's efficiency in computing the metrics.

Regarding **RQ2: To what extent is ADDET accurate in detecting most defects?**. To address this question, we have asked five graphic design quality experts to analyze the found 306 MUI defects. Such MUIs are classified by ADDET as holders with three or more structural defects. The experts assigned a weight to each TN/TP/FN and FP and accepted about 71% of the detected defects. The true negatives represent only 2.26%, which indicates that many true positives have not been missed by the tool. Two potential explanations for the false positives are: 1) metrics thresholds were not met in the OR statements, or 2) one of the metrics were not met in the AND statements. These combinations of metrics in the rules have sharpened the tool's detection accuracy by satisfying the basic structural metric properties to trigger a specific defect and then fulfilling the secondary metrics. For example, if the homogeneity of the MUI is low and the uniformity is high, there would be no ILW defect triggered. Our goal was to ensure that all the structural properties of a single defect are satisfied. Nevertheless, there were reported 29.3% of the true negatives. Often the experts may be rough in their identification and see certain structural aspects that can not be formulated with equations as an example of the wrong positioning of graphical elements. We know that each expert sees the design from his/her perspective, based on his/her design assessment experience, however, we wanted to ensure our tool was able to identify the most noticeable defect(s). We conclude that with a precision of 71% and a recall of 58%, our proposed tool will correctly detect defects.

7 THREATS TO VALIDITY

In this section, we report threats to the validity of our study.

The internal threats of validity present the factors that do not support our tool's applicability. We have selected a set of Android apps on Google Play Store that is well rated by end-users. Having a high user rating means the app reflects a high-quality product that can please a wide variety of end-users. To build our base of examples, we relied on evaluating the selected MUIs from twenty users with diversified profiles. In addition, we used 14 metrics that are used in several studies to

evaluate the structural aesthetics quality of a MUI and proposed our metric (Clarity) to evaluate the clarity in terms of the comprehensibility of the labels of components.

The construct validity concerns the tool's error rate in computing metrics values and detecting defects. We have built a tool which calculates a set of 15 metrics based on the geometric data taken from the *UiAutomatorViewer*. This latter inspects every single item in the MUI layout with its positioning and dimensions. The proposed tool is based on a collection of restrictive rules which aim to satisfy several structural variables before triggering a defect. Besides, the box-plot was used to produce thresholds. Those values can be manually tuned by an expert to give superior results, we aimed to automate this phase in this research, however, and our findings have been statistically noteworthy. These thresholds may not be the best ones, so we need to test a broader range of MUIs to improve threshold generalization. The tool is limited to evaluating MUIs designed entirely through XML formatting language. That is, evaluating maps or applications for animated games using Java may not produce accurate results.

The external threat centers around evaluating the scalability of our findings. To the best of our knowledge, ADDET is considered the pioneer in detecting structural defects compared to existing studies which only provide metrics calculation. It is true that a metric actually informs about a structural problem. Nonetheless, marking the structural issues is helpful, so developers/designers do not need to re-understand the purpose of a metric. This tool will enable us to build a large user interface base that will be the first step towards automated MUI design restructuring. Our measurements and defect identification are independent from operating system architectures. However, the tool's parser uses patterns which are unique to the format of generated dump files.

8 CONCLUSION

In this paper, we proposed the ADDET tool for automated evaluation and identification of structural aesthetic defects of the Android mobile user interface. The first phase involves collecting a well-known set of metrics for assessing aspects of structural aesthetics. Also, we carefully selected a set of five defects that represent specific structural problems with MUIs. Using a multi-objective genetic algorithm (NSGA-II), we put our effort into automatically assigning each defect to the corresponding metrics. To increase the detection's precision, we thought about calculating how necessary a metric is to detect a specific defect. The tool's quantitative and qualitative analysis shows its usefulness in the evaluation and detection of structural design issues. As for future research, we plan to extend the collection of assessed MUIs. Further research into the weight of metrics regarding their contribution to an assessment of MUI's structural aesthetics will be of interest. This research would require an advanced empirical study, in which we evaluate the different designers' visual techniques and analyze the critical aspects that designers see as problematic. We can also trigger the need for a survey, where a large scale of end-users rank the property they see more critical. Most significantly, we would like to explore how the MUI evolves in function of the number of defects? That is to say, are developers fixing the defects that occur? We plan to research the survivability of a MUI's structural aesthetic defects alongside an application's evolution. This empirical analysis will help in figuring out which defect can be eliminated easily.

REFERENCES

- [1] Khalid Alemerien and Kenneth Magel. 2014. GUIEvaluator: A Metric-tool for Evaluating the Complexity of Graphical User Interfaces.. In *SEKE*. 13–18.
- [2] Khalid Alemerien and Kenneth Magel. 2015. SLC: a visual cohesion metric to predict the usability of graphical user interfaces. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 1526–1533.
- [3] Hannah Alvarez. 2015. Making Your Icons User-Friendly: A Guide to Usability in UI Design. <https://www.useresting.com/blog/user-friendly-ui-icons/>. [Online; accessed 20-June-2019].

- [4] Simon Attfeld, Gabriella Kazai, Mounia Lalmas, and Benjamin Piwowarski. 2011. Towards a science of user engagement (position paper). In *WSDM workshop on user modelling for Web applications*. 9–12.
- [5] Emna Ben Ayed, Christophe Kolski, Rim Magdich, and Houcine Ezzedine. 2016. Towards a context based Evaluation Support System for Quality in Use assessment of mobile systems. In *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. IEEE, 004350–004355.
- [6] Rosnita Baharuddin, Dalbir Singh, and Rozilawati Razali. 2013. Usability dimensions for mobile applications—a review. *Res. J. Appl. Sci. Eng. Technol* 5, 6 (2013), 2225–2231.
- [7] Tengfei Bao, Huanhuan Cao, Enhong Chen, Jilei Tian, and Hui Xiong. 2012. An unsupervised approach to modeling personalized contexts of mobile users. *Knowledge and Information Systems* 31, 2 (2012), 345–370.
- [8] Narjess Bessghaier and Makram Souii. 2017. Towards Usability Evaluation of Hybrid Mobile User Interfaces. In *Computer Systems and Applications (AICCSA), 2017 IEEE/ACS 14th International Conference on*. IEEE, 895–900.
- [9] Nigel Bevan and Miles Macleod. 1994. Usability measurement in context. *Behaviour & information technology* 13, 1-2 (1994), 132–145.
- [10] Patrick Mbenza Buanga. 2011. Automated evaluation of graphical user interface metrics. *PhD thesis. Catholic university of Louvain, Belgium* (2011).
- [11] John W Cresswell. 1998. Qualitative inquiry and research design: Choosing among five traditions. *Thousand Oaks, Calif.: Sage Publications*. (1998).
- [12] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschan, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 845–854.
- [13] Claudia Ehmke and Stephanie Wilson. 2007. Identifying web usability problems from eye-tracking data. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 1*. British Computer Society, 119–128.
- [14] Martin Fowler. 2018. *Refactoring: improving the design of existing code*. Addison-Wesley Professional.
- [15] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. 1994. Weka: A machine learning workbench. In *Proceedings of ANZIS'94-Australian New Zealand Intelligent Information Systems Conference*. IEEE, 357–361.
- [16] Gasmı Ines, Soui Makram, Chouchane Mabrouka, and Abed Mourad. 2017. Evaluation of Mobile Interfaces as an Optimization Problem. *Procedia Computer Science* 112 (2017), 235–248.
- [17] Melody Y Ivory, Rashmi R Sinha, and Marti A Hearst. 2001. Empirically validated web page design metrics. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 53–60.
- [18] Marouane Kessentini and Ali Ouni. 2017. Detecting Android smells using multi-objective genetic programming. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*. IEEE Press, 122–132.
- [19] Jesper Kjeldskov and Mikael B Skov. 2003. Creating realistic laboratory settings: comparative studies of three think-aloud usability evaluations of a mobile system. In *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction*. 663–670.
- [20] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.
- [21] Liisa Kuparinen, Johanna Silvennoinen, and Hannakaisa Isomäki. 2013. Introducing usability heuristics for mobile map applications. In *Proceedings of the 26th International Cartographic Conference, August 25 30, 2013, Dresden, Germany, ISBN 978-1-907075-06-3*. International Cartographic Association.
- [22] Michele Lanza and Radu Marinescu. 2007. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.
- [23] Talia Lavie and Noam Tractinsky. 2004. Assessing dimensions of perceived visual aesthetics of web sites. *International journal of human-computer studies* 60, 3 (2004), 269–298.
- [24] Hosub Lee, Young Sang Choi, and Yeo-Jin Kim. 2011. An adaptive user interface based on spatiotemporal structure learning. *IEEE Communications Magazine* 49, 6 (2011), 118–124.
- [25] Bo-Fu Liu, Shih-Chun Chou, Yu-Ting Lin, and Jih-Yiing Lin. 2011. Toward easy delivery of device-oriented adaptive user interface on mobile devices. In *Information Science and Service Science (NISS), 2011 5th International Conference on New Trends in*, Vol. 1. IEEE, 80–85.
- [26] Rohit Mahajan and Ben Shneiderman. 1997. Visual and textual consistency checking tools for graphical user interfaces. *IEEE Transactions on Software Engineering* 23, 11 (1997), 722–735.
- [27] Michael Minge and Manfred Thüning. 2018. Hedonic and pragmatic halo effects at early stages of user experience. *International Journal of Human-Computer Studies* 109 (2018), 13–25.
- [28] Aliaksei Miniukovich and Antonella De Angeli. 2014. Visual impressions of mobile app interfaces. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*. ACM, 31–40.
- [29] Aliaksei Miniukovich and Antonella De Angeli. 2015. Computation of interface aesthetics. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1163–1172.

- [30] Francisco Montero and V López-Jaquero. 2010. GUILayout++: Supporting prototype creation and quality evaluation for abstract user interface generation. In *Workshop of ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Berlin, Germany*.
- [31] Kevin Mullet and Darrell Sano. 1995. *Designing visual interfaces: Communication oriented techniques*. Vol. 2550. SunSoft Press Englewood Cliffs (NJ).
- [32] Fatih Nayebi, Jean-Marc Desharnais, and Alain Abran. 2012. The state of the art of mobile application usability evaluation. In *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*. IEEE, 1–4.
- [33] DCL Ngo, LS Teo, and JG Byrne. 2000. Formalising guidelines for the design of screen layouts. *Displays* 21, 1 (2000), 3–15.
- [34] David Chek Ling Ngo, Azman Samsudin, and Rosni Abdullah. 2000. Aesthetic measures for assessing graphic screens. *J. Inf. Sci. Eng* 16, 1 (2000), 97–116.
- [35] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 249–256.
- [36] James Noble and Larry L Constantine. 1996. Interactive design metric visualization: Visual metric support for user interface design. In *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*. IEEE, 213–220.
- [37] Heather L O'Brien and Elaine G Toms. 2008. What is user engagement? A conceptual framework for defining user engagement with technology. *Journal of the American society for Information Science and Technology* 59, 6 (2008), 938–955.
- [38] Heather L O'Brien and Elaine G Toms. 2010. The development and evaluation of a survey to measure user engagement. *Journal of the American Society for Information Science and Technology* 61, 1 (2010), 50–69.
- [39] Annarita Paiano, Giovanni Lagioia, and Andrea Cataldo. 2013. A critical analysis of the sustainability of mobile phone use. *Resources, Conservation and Recycling* 73 (2013), 162–171.
- [40] Katharina Reinecke, Tom Yeh, Luke Miratrix, Rahmatri Mardiko, Yuechen Zhao, Jenny Liu, and Krzysztof Z Gajos. 2013. Predicting users' first impressions of website aesthetics with a quantification of perceived visual complexity and colorfulness. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2049–2058.
- [41] Andreas Riegler and Clemens Holzmann. 2018. Measuring Visual User Interface Complexity of Mobile Applications With Metrics. *Interacting with Computers* 30, 3 (2018), 207–223.
- [42] Brian Robinson and Penelope Brooks. 2009. An initial study of customer-reported GUI defects. In *ICSTW'09. International Conference*. IEEE, 267–274.
- [43] Young Sam Ryu. 2005. *Development of usability questionnaires for electronic mobile products and decision making methods*. Ph.D. Dissertation. Virginia Tech.
- [44] Maria Shitkova, Justus Holler, Tobias Heide, Nico Clever, and Jörg Becker. 2015. Towards Usability Guidelines for Mobile Websites and Applications.. In *Wirtschaftsinformatik*. 1603–1617.
- [45] Ian Shoenberger, Mohamed Wiem Mkaouer, and Marouane Kessentini. 2017. On the Use of Smelly Examples to Detect Code Smells in JavaScript. In *European Conference on the Applications of Evolutionary Computation*. Springer, 20–34.
- [46] Andreas Sonderegger and Juergen Sauer. 2010. The influence of design aesthetics in usability testing: Effects on user performance and perceived usability. *Applied ergonomics* 41, 3 (2010), 403–410.
- [47] Makram Soui, Mabrouka Chouchane, Ines Gasmi, and Mohamed Wiem Mkaouer. 2017. PLAIN: PLugin for predicting the usability of Mobile User Interface.. In *VISIGRAPP (1: GRAPP)*. 127–136.
- [48] I Standard. 1998. Ergonomic requirements for office work with visual display terminals (vdt)s—part 11: Guidance on usability. ISO Standard 9241-11: 1998. *International Organization for Standardization* (1998).
- [49] Xu Sun and Andrew May. 2013. A comparison of field-based and lab-based experiments to evaluate user experience of personalised mobile devices. *Advances in Human-Computer Interaction* 2013 (2013), 2.
- [50] Alistair Sutcliffe. 2009. Designing for user engagement: Aesthetic and attractive user interfaces. *Synthesis lectures on human-centered informatics* 2, 1 (2009), 1–55.
- [51] Gustavo F Tondello, Rina R Wehbe, Rita Orji, Giovanni Ribeiro, and Lennart E Nacke. 2017. A Framework and Taxonomy of Videogame Playing Preferences. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. ACM, 329–340.
- [52] Ali Türkyilmaz, Simge Kantar, M Enis Bulak, Ozgur Uysal, et al. 2015. User Experience Design: Aesthetics or Functionality?. In *Managing Intellectual Capital and Innovation for Sustainable and Inclusive Society: Managing Intellectual Capital and Innovation; Proceedings of the MakeLearn and TIIM Joint International Conference 2015*. ToKnowPress, 559–565.
- [53] Eric N Wiebe, Allison Lamb, Megan Hardy, and David Sharek. 2014. Measuring engagement in video game-based environments: Investigation of the User Engagement Scale. *Computers in Human Behavior* 32 (2014), 123–132.
- [54] Aiko Yamashita and Leon Moonen. 2013. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 682–691.

- [55] Rosa Yáñez Gómez, Daniel Cascado Caballero, and José-Luis Sevillano. 2014. Heuristic evaluation on mobile interfaces: A new checklist. *The Scientific World Journal* 2014 (2014).
- [56] Mathieu Zen. 2013. Metric-based evaluation of graphical user interfaces: model, method, and software support. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 183–186.
- [57] Mathieu Zen and Jean Vanderdonckt. 2014. Towards an evaluation of graphical user interfaces aesthetics based on metrics. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*. IEEE, 1–12.

Mathematical formulae of the 14 selected metrics from literature

1- Density metric

$$\mathbf{Dst} = 1 - 2 \left| 0.5 - \frac{\sum_i^n a_i}{a_{frame}} \right| \in [0, 1] \quad (5)$$

n = Number of objects

a_{frame} = Area of the frame

a_i = Area of interactive object i .

2- Sequence metric

$$\mathbf{SQM} = 1 - \frac{\sum_{j=UL,UR,LL,LR} |q_j - v_j|}{8} \in [0, 1] \quad (6)$$

$$q_j = \{q_{UL}, q_{UR}, q_{LL}, q_{LR}\} = \{4, 3, 2, 1\}$$

$$v_j = \begin{cases} 4, & \text{if } w_j \text{ is the largest in } w \\ 3, & \text{if } w_j \text{ is the 2nd largest in } w \\ 2, & \text{if } w_j \text{ is the 3rd largest in } w \\ 1, & \text{if } w_j \text{ is the smallest in } w \end{cases}$$

with:

j = UL, UR, LL, LR

$$w_j = q_j \sum_i^{n_j} a_{ij}$$

$$w = \{w_{UL}, w_{UR}, w_{LL}, w_{LR}\}$$

where:

UL = Upper-left, UR = Upper-right, LL = Lower-left, LR = Lower-right

a_{ij} = the area of object i on quadrant j .

Each quadrant is given a weighting in q .

3- Regularity metric

$$\mathbf{Rgl} = \frac{|RM_{alignment}| + |RM_{spacing}|}{2} \in [0, 1] \quad (7)$$

$RM_{alignment}$ = is the extent to which the alignment points are minimised. with

$$RM_{alignment} = \begin{cases} 1, & \text{if } n = 1 \\ 1 - \frac{n_{vap} + n_{nap}}{2n}, & \text{Otherwise} \end{cases}$$

$RM_{spacing}$ = is the extent to which the alignment points are consistently spaced with

$$RM_{spacing} = \begin{cases} 1, & \text{if } n = 1 \\ 1 - \frac{n_{spacing} - 1}{2(n-1)}, & \text{Otherwise} \end{cases}$$

n_{vap} = Number of vertical alignments points

n_{nap} = Number of horizontal alignments points

$n_{spacing}$ = the number of distinct distances between column and row starting points

n = the number of objects on the frame.

4- Homogeneity metric

$$\mathbf{HM} = \frac{W}{W_{max}} \in [0, 1] \quad (8)$$

W = is the number of different ways a group of n objects can be arranged for the four quadrants
 n_j = Number of objects in a quadrant j .

$$W = \frac{n!}{n_{UL}!n_{UR}!n_{LL}!n_{LR}!}$$

n = Number of objects on the frame

n_{UL} = Number of objects in upper-left

n_{UR} = Number of objects in upper-right

n_{LL} = Number of objects in lower-left

n_{LR} = Number of objects in lower-right

W = is maximum when the n objects are evenly allocated to the various quadrants of the screen.

$$W_{max} = \frac{n!}{(\frac{n}{4})^4}$$

5- Economy metric

$$\mathbf{Ecm} = \frac{1}{n_{size}} \in [0, 1] \quad (9)$$

n_{size} = Number of sizes of different components.

6- Simplicity metric

$$\mathbf{Smpl} = \frac{3}{n_{vap} + n_{hap} + n} \in [0, 1] \quad (10)$$

n_{vap} = Number of vertical alignments points

n_{hap} = Number of horizontal alignments points

n = Number of objects in a UI.

7- Balance metric

$$\mathbf{BM} = 1 - \frac{|BM_{vertical}| + |BM_{horizontal}|}{2} \in [0, 1] \quad (11)$$

$$BM_{vertical} = \frac{w_L - w_R}{\max(|w_L|, |w_R|)}$$

$$BM_{horizontal} = \frac{w_T - w_B}{\max(|w_T|, |w_B|)}$$

$$w_j = \sum_i^{n_j} a_{ij} * d_{ij}$$

a_{ij} = Area occupied by object i on side j

d_{ij} = Distance between object i and the frame

where $j \in \{L, R, T, B\}$

8- Cohesion metric

$$\mathbf{CM} = \frac{|CM_{ft}| + |CM_{lo}|}{2} \in [0, 1] \quad (12)$$

CM_{fl} = is a relative measure of the ratios of the layout and screen .

$$\text{with: } CM_{fl} = \begin{cases} t_{fl}, & \text{if } t_{fl} \leq 1 \\ \frac{1}{t_{fl}}, & \text{Otherwise} \end{cases}$$

with:

$$t_{fl} = \frac{h_{layout}/b_{layout}}{h_{frame}/b_{frame}}$$

h_{layout} = The height of the layout

b_{layout} = the width of the layout

h_{frame} = The height of the frame

b_{frame} = The width of the frame

CM_{lo} = is a relative measure of the ratios of the objects and layout.

with:

$$CM_{lo} = \frac{\sum_i^n f_i}{n}$$

with:

$$f_i = \begin{cases} t_i, & \text{if } t_i \leq 1 \\ \frac{1}{t_i}, & \text{Otherwise} \end{cases}$$

with:

$$t_i = \frac{h_i/b_i}{h_{layout}/b_{layout}} \quad b_i = \text{Width of object } i/$$

h_i = Height of object i.

n = Number of object on the frame.

9- Layout Uniformity metric

$$LU = 100 * \left(1 - \frac{(N_h + N_w + N_t + N_l + N_b + N_r) - M}{6N - M}\right) \in [0, 1] \quad (13)$$

$$M = 2 + 2[2\sqrt{N}]$$

N = Number of objects in a UI.

M = Adjustment number for the minimum number of possible alignments and sizes that make the value of LU ranges from 0 to 100.

N_h =Number of different heights.

N_w =Number of different widths.

N_t =Number of different Top edge alignment.

N_l =Number of different Left edge alignment.

N_b =Number of different Bottom edge alignment.

N_r =Number of different Right edge alignment.

10- Unity metric

$$\mathbf{Unt} = \frac{|UM_{form}| + |UM_{space}|}{2} \in [0, 1] \quad (14)$$

UM_{form} = is the extent to which the objects are related in size and shape. (we have excluded the color variable which is independent of the shape and size, and divided the formula by 2.)

$UM_{form} = 1 - \frac{n_{size} + n_{shape} - 2}{2}$ UM_{space} = is a relative measure of the space between widgets and that of

the margins.

$$UM_{space} = 1 - \frac{a_{layout} - \sum_i^n a_i}{a_{frame} - \sum_i^n a_i}$$

a_i , a_{layout} , and a_{frame} are the areas of object i , the layout, and the frame, respectively; n_{size} and n_{shape} are the numbers of sizes and shapes used, respectively; and n is the number of objects on the frame.

11- Integrality metric

$$Intg = 1 - (0.5 \left[\frac{n_{size} - 1}{n} + \frac{s_{sc} + \sum_i^n a_i}{2a_{MUI}} \right]) \in [0, 1] \quad (15)$$

n_{size} the number of different sizes of objects used by the interface. n : the number of objects. a_{MUI} the area of the mobile interface. a_{sc} the area of the screen. a_i the area of the interactive object i .

12- Grouping metric

$$Grp = (0.85f) * UM_{space} + (0.15f) * AL \in [0, 1] \quad (16)$$

$$UM_{space} = 1 - (a_{Layout} - \text{sumOfAreas} / a_{Frame} - \text{sumOfAreas})$$

$AL = \min(1, (4 * Ni - (nHap * nVap / Ni)) / (4 * Ni - 4))$ a_{Layout} is the area of the layout; sumOf Areas is the sum of the areas of objects;

a_{Frame} is the area of the frame;

Ni is the number of objects of type i ;

$nHap$ and $nVap$ are the number of horizontal alignment points and the number of vertical alignment points, respectively.

13- Nb-Elements metric

$$NB = \sum_i i^{nb} Widgets \in [0, 1] \quad (17)$$

14- Complexity metric

$$Cmpl = \frac{Dst + SQM + Rgl + HM + Ecm + Smpl + BM + CM + LU + Unt + Intg + Grp + Clarity + NB}{14} \in [0, 1] \quad (18)$$