# Don't Do That! Hunting Down Visual Design Smells in Complex UIs against Design Guidelines

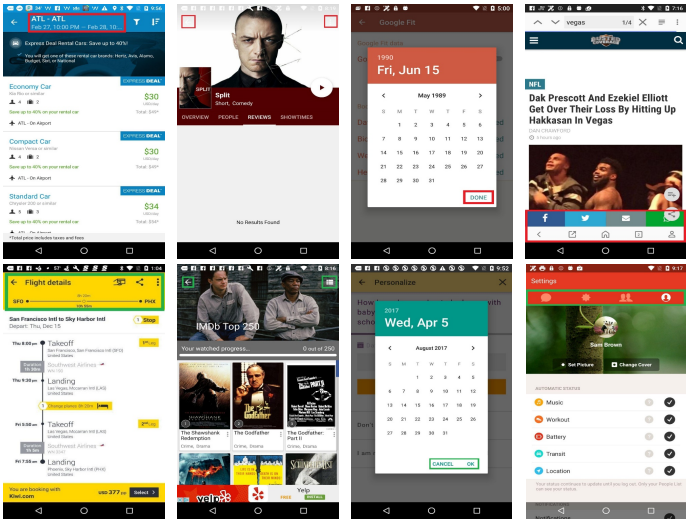
Bo Yang*,Zhenchang Xing†,Xin Xia‡¶,Chunyang Chen†,Deheng Ye§,Shanping Li*
*Zhejiang University, Hangzhou, China
†Australian National University, Canberra, Australia
‡Monash University, Melbourne, Australia
§Tencent AI Lab, Shenzhen, China
imyb@zju.edu.cn,zhenchang.Xing@anu.edu.au,xin.xia@monash.edu,
chunyang.chen@monash.edu,dericye@tencent.com,shan@zju.edu.cn

***Abstract*—Just like code smells in source code, UI design has visual design smells. We study 93 don't-do-that guidelines in the Material Design, a complex design system created by Google. We find that these don't-guidelines go far beyond UI aesthetics, and involve seven general design dimensions (layout, typography, iconography, navigation, communication, color and shape) and four component design aspects (anatomy, placement, behavior and usage). Violating these guidelines results in visual design smells in UIs (or UI design smells). In a study of 60,756 UIs of 9,286 Android apps, we find that 7,497 UIs of 2,587 apps have at least one violation of some Material Design guidelines. This reveals the lack of developer training and tool support to avoid UI design smells. To fill this gap, we design an automated UI design smell detector (UIS-Hunter) which extracts and validates multi-modal UI information (component metadata, typography, iconography, color and edge) for detecting the violation of diverse don't-guidelines in Material Design. The detection accuracy of UIS-Hunter is high (precision=0.81, recall=0.90) on the 60,756 UIs of 9,286 apps. We build a guideline gallery with real-world UI design smells that UIS-Hunter detects for developers to learn best Material Design practices. Our user studies show that UIS-Hunter is more effective than manual detection of UI design smells, and the UI design smells that are detected by UIS-Hunter have severely negative impacts on app users.**

## I. Introduction

Graphical User Interface (GUI) should allow app users to naturally figure out the app's information structure and action flow. Unfortunately, an Android GUI may be poorly designed. According to the Appiterate survey [1], 42% of users would uninstall an app with a poorly designed UI. Fig. 1 presents some examples of poorly designed Android app UIs (issues highlighted in red boxes). In Fig. 1(a), truncating the full title of the top app bar hinders the access of information. The background image in Fig. 1(b) makes the foreground buttons/text illegible. The confirmation dialog in Fig. 1(c) provides only a single action, and cannot be dismissed. In Fig. 1(d), attaching tabs to bottom navigation can cause confusion about what action or tab controls which content. We refer to such poorly designed UIs as *UI design smells*.

Just like code smells that indicate violations of good software design guidelines [2], UI design smells violate good UI design guidelines. UI design guidelines constitute *design systems* for a vast variety of products, platforms and services [3]. In this work, we use Material Design as a case study of design systems. In Material Design documentation, we investigate 93 explicit don't-guidelines for 17 UI components (e.g., app bar, banner, button, tab and dialog) (see Section II-A). These design guidelines go far beyond UI aesthetics. They involve seven general design dimensions (i.e., layout, typography, iconography, navigation, communication, color and shape) and four component design aspects (i.e., anatomy, placement, behavior and usage).




(a) Text truncation (b) Illegible buttons/text (c) Single button in confirmation dialog (d) Tab attached to bottom navigation

Fig. 1: UI design smells (1st row) vs non-smell UIs (2nd row) (issues highlighted in red boxes)

However, there has been little support for detecting UI design smells violating the design guidelines. Many tools have been developed to detect and remedy code smells [2], [4], [5], [6] and other programming or stylistic errors [7], [8], [9]. Some techniques can detect the inconsistencies between UI mockups and implemented UIs [10], or discern abnormal designs from normal ones [11], [12], [13]. But these techniques cannot check the violation of a UI design against the visual guidelines in a design system. First, the design

¶Corresponding author.

examples used to explain the guidelines (e.g., Fig. 2) are very different from the actual UI designs to be checked (e.g., Fig. 1), so they cannot be directly contrasted to determine the guideline violation. Second, checking a UI design against certain design guidelines requires to examine multi-modal information, including a wide range of component information (e.g., type, instance count, size/position, color and text content) and the actual rendering of text and images (see Section II).

In this work, we develop an automated tool (called *UIS-Hunter*) for detecting UI design smells against the don't-guidelines of Material Design. The design of UIS-Hunter is informed by a demographic study of the don't-guidelines in Material Design documentation (see Section II). The study identifies five types of atomic UI information for checking visual guideline violations, including component metadata, typography, iconography, color and edge. Each visual guideline is then defined as a violation condition over the relevant type(s) of atomic UI information. UIS-Hunter currently supports 71 don't-guidelines, covering seven general design dimensions and 11 types of UI components. For each supported visual guideline, we collect a set of conformance and violation UIs from real Android apps. Given a UI design, our tool reports UI design smells if the corresponding guideline violation conditions are satisfied. It produces a violation report (see Fig. 4) that lists the violated visual guidelines, highlights the UI regions violating these guidelines, and provide conformance and violation UI examples to help developers understand the reported UI design smells.

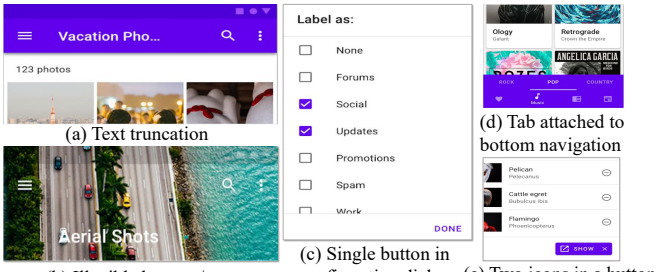
We evaluate UIS-hunter on a dataset of 60,756 unique UI screenshots of 9,286 Android apps. UIS-Hunter achieves a precision of 0.81 and a recall of 0.90. And F1-score is 0.87. We also conduct two user studies to evaluate the utility of UIS-Hunter. Our studies show that UIS-Hunter can detect UI design smells more effectively than human checkers, and most of the UI design smells reported by UIS-Hunter are rated as severe by the majority of 5 app users.

In this paper, we make the following contributions:

- We conduct a demographic study of visual guidelines in Material Design documentation, which informs the design of automated UI design smell detection tools.
- We develop the UIS-Hunter tool that can validate the design of 11 types of UI components against 71 visual guidelines of Material Design.
- We confirm the detection accuracy of UIS-Hunter on a large dataset of real Android app UIs, and the utility of UIS-Hunter for front-end developers and app users.
- Based on the UIS-Hunter's detection results, we build a Material Design guideline gallery to assist the learning of abstract visual guidelines with real-app UI design smells. We also release our manually validated dataset as a benchmark for studying UI design smells.

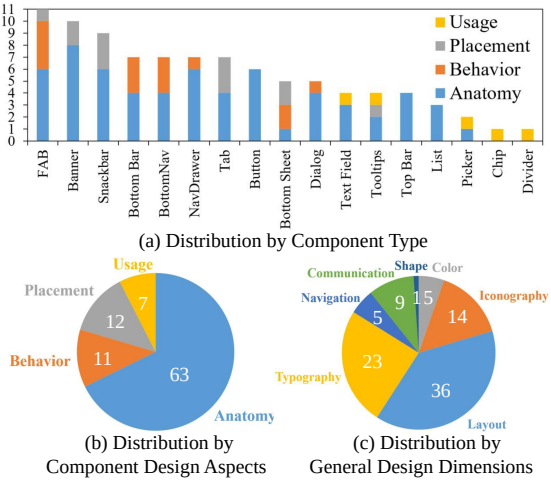
## II. Google Material Design Guidelines: A Demographic Study

Our study examines three perspectives of visual design guidelines in Material Design documentation: *component type and count*, *component design aspect*, and *general design dimension*. In the study, we identify the *types of atomic UI information* required for determining the violation of the visual guidelines, which lays the foundation to develop the automated tool for detecting UI design smells violating these guidelines.




(a) Text truncation (b) Illegible buttons/text (c) Single button in confirmation dialog (d) Tab attached to bottom navigation (e) Two icons in a button

Fig. 2: Examples of don't-guidelines in Material Design




(a) Distribution by Component Type

(b) Distribution by Component Design Aspects

(c) Distribution by General Design Dimensions

Fig. 3: Distribution of don't-guidelines

### A. The Collection of Material Design Guidelines

In this work, we examine the official documentation of Material Design Component (as of May 2020). The documentation contains over 1,000 web pages and over 5,000 UI design examples. Material design webpages (and many other design systems) are well-structured and adopt consistent writing conventions. In this study, we focus on *explicit don't-guidelines* that explicitly marked as "don't" and "caution" with at least one illustrative UI example. Fig. 2 shows five examples of such don't guidelines. The four examples in Fig. 2(a-d) correspond to the design smells in Fig. 1(a-d) respectively. Fig. 2 shows one more guideline "don't use two icons in a button". Although some guidelines involve multiple types of components, they will not be repeated for all involved components. For example, "don't attach tab to bottom navigation" in Fig. 2(d) is described as a guideline for tab but not for bottom navigation. We refer to tab as primary component, and bottom navigation as related component for this design guideline.

We collect 126 explicit don't-guidelines for 23 types of UI components. Three types of components (backdrop, navigation

rail and side sheet) are in beta status. We exclude 14 don't-guidelines of these three beta UI components. As this study focuses on Android app UI design smells, we exclude five don't-guidelines for web or desktop app UI design. We exclude 15 animation don't-guidelines of 10 types of components, such as card flipping and scrolling, bottom sheet sliding up. Analyzing UI animation effects requires sophisticated video embedding techniques [12], which is beyond the scope of this work. Finally, we collect 93 don't-guidelines for 17 types of UI components for the subsequent demographic analysis.

### B. Demographic Analysis of Design Guidelines

*1) Component Type and Count:* Fig. 3(a) shows these 17 types of UI components and the number of don't-guidelines they have. Three types of UI components (floating action button (FAB), banner and snackbar) have 9 or more don't-guidelines. Two types of UI components (chip and divider) have only one don't-guidelines. The remaining 12 types of UI components have 2 to 8 don't guidelines.

We divide the 93 don't-guidelines into single-type and multiple-type. 66 guidelines are single-type as they involve only one type of UI component. Single-type guidelines generally require the count of certain component instance. Some guidelines need to verify the number of destinations in a bottom navigation matches the limitation. Since the right navigation drawer is prohibited, certain guidelines need component coordinates to decide the left and right of a navigation drawer. Also, some single-type guidelines require text rendering results to validate improper text usage (Fig. 1(a)). 27 don't-guidelines are multiple-type as they involve multiple types of UI components, for example, background image and foreground buttons/text (Fig. 1(b)), tab and bottom navigation (Fig. 1(c)), dialog and button (Fig. 1(d)). To validate a multiple-type guideline, we need to examine not only the primary UI component but also related components interacting with or being composed in the primary component.

*2) Component Design Aspects:* Material Design describes the design of a UI component from four aspects: anatomy, placement, behavior and usage. Fig 3(b) shows the distribution of the 93 don't-guidelines across these four component design aspects. Anatomy, placement, behavior and usage guidelines account for about 68%, 13%, 12% and 7%, respectively.

Validating anatomy guidelines requires the information about component composition, text content and actual rendering of text/images/edges of a component's constituent parts. For example, an app bar may be composed of a navigation button, a title, some action items and an overflow menu. The title has guidelines regarding improper text usage, such as text wrapping, truncation and shrinking (Fig. 1(a)). Validating behavior guidelines requires to examine the metadata (e.g., count) of the primary component after certain interaction and the icons of action button (e.g., navigation button). For example, the number of destinations that a float action button can display has a minimum and maximum limitation, or "don't place a navigation button in both top and bottom bar". Usage guidelines describe typical usage scenarios of a UI component. For example, Material Design suggests using bottom navigation when there are three to five destinations. If the destinations are fewer than three, Material Design suggests using tab or navigation drawer respectively, rather than bottom navigation. Usage guidelines also describe the use of different variants of the component. For example, a dialog can be a simple dialog or a confirmation dialog. When using a simple dialog, an action button is not needed. But a confirmation dialog needs two actions to dismiss or confirm the dialog (Fig. 1(c)). To validate usage guidelines, we typically need to examine the component type and its constituent elements. Placement guidelines specify the location of a UI component on the screen or relative to other UI components, such as "don't attach tab to bottom navigation" (Fig. 1(d)). Validating placement guidelines requires the information about component type/size/position, and usually need to examine the relative position of multiple UI components on the screen.

*3) General Design Dimensions:* Material Design describes the principles of 11 design dimensions (so called Material Foundation). We focus on non-animation guidelines and exclude four dimensions: environment, sound, motion and interaction. 93 don't-guidelines can be divided into the remaining seven design dimensions: layout, typography, iconography, navigation, communication, color and shape, as summarized in Fig. 3(c). Layout guidelines account for 39%, typography and iconography for 40%, and rest four dimensions for 21%.

Layout specifies the organization of UI components on the screen. It includes all 11 placement guidelines, and 18 anatomy guidelines, such as maximum number of FAB and banner, minimum number of buttons in a bottom bar, and the usage of divider between destinations in a navigation drawer. 8 layout guidelines are about component behavior, and 3 layout guidelines are about component usage which specifies the improper usage of text field, divider and tooltip. Typography specifies the style and usage of text labels, such as text length, wrapping, truncation and shrinking. Validating typography guidelines requires not only text content (e.g., button label "Cancel"), but also the appearance of the displayed text, such as the presence of "..." in Fig. 1(a), the number of text lines, the height/width of text. Iconography specifies the use of images and icons in UI design, such as avoid imagery that makes button or text illegible in Fig. 1(b), and "don't apply icons to some destinations but not others in a navigation drawer".

Navigation regulates how users should move through an app. They mainly include guidelines for navigation components (top/bottom bar, bottom navigation, tab and FAB), such as "don't place a navigation button in both top and bottom bar". Communication defines guidelines that ask for confirmation before taking action and acknowledge successful actions. For example, confirmation dialog should not provide only a confirmation button (Fig. 1(c)) Color specifies the use of meaningful colors. For example, "don't use multiple colors in bottom navigation", as they make it harder for users to distinguish the status of destinations. As another example, using primary color as the background color of a text field may make it be mistaken as buttons and harm text legibility.

Finally, we have one shape guideline "don't alter the shape of a snackbar", which needs to check the edge of snackbar.

### C. Types of Atomic UI Information

The demographic analysis of Material Design don't-guidelines identifies five types of atomic UI information required for detecting guideline violations. First, component metadata includes component coordinates, type, instance count, composition hierarchy and text content. Text content is used to determine component semantics. For example, the button label "Cancel" or "Discard" indicates a dismissive action. The label "Sorry..", "Warning!", or "Are you sure?" indicates ambiguous dialog title. We collect a common label glossary mentioned in the Material Design documentation for validating the guidelines that need to check component semantics. Second, typography refers to the actual appearance of the displayed text. Although component metadata provides text content, the actual appearance of the displayed text is required to detect improper text usage, such as text wrap, resize and truncation. Third, iconography refers to the presence of images/icons and their visual effects. Fourth, the primary color refers to the color displayed most frequently across an app's screen and components. Fifth, edge information is needed for divider- and shape-related guidelines.

48 of 93 don't-guidelines require one type of atomic UI information to validate. Most of these guidelines are about text usage and limitation of component instances. The remaining 45 guidelines require two or more types of information. For example, the guideline "don't apply icons to some destinations but not others in a navigation drawer" requires component metadata (the component type and coordinate information, the number of destinations in the drawer) and iconography information (the number of icons used in the drawer). When the two numbers do not match, the guideline is violated.

## III. UI Design Smell Detection

As shown in Fig. 4, our approach consists of two main components: a *knowledge base of UI design guidelines* and a *UI design smell detector* including *input UI design parser*, *atomic UI information extractors* and *UI design validator*. In this section, we first give an overview of our approach for UI design smell detection, and then detail the design and construction of its main components.

### A. Approach Overview

The knowledge base of UI design guidelines contains a set of visual design guidelines extracted from a design system (Section III-B). Each guideline is indexed by the type of primary UI component involved and is accompanied by a set of conformance and violation UI examples from the design system and real-world applications. The UI design smell detector takes as input a UI design mockup or an app UI screenshot (Section III-C). It validates the design of each component in the input UI against the guidelines of the corresponding component type. In addition to component metadata obtained from the input UI, the validation examines four types of atomic UI information (Section III-D) (typography, iconography, color and edge) against each guideline's violation condition. If certain guideline violations are detected, the detector produces a UI design smell report (Section III-E).

### B. Constructing Knowledge Base of UI Design Guidelines

A design system, such as Google's Material Design [14] is usually documented in semi-structured documentation. To construct a knowledge base of UI design guidelines, we conduct a demographic study of the design guidelines in the design system documentation, such as the study reported in Section II. The knowledge base indexes each design guideline by the type of primary component involved in the guideline, and annotates the guideline with multi-dimensional information such as component design aspect, general design dimension and guideline severity. This allows the developers to explore the multi-dimensional space of a complex design system in a more structured way, as demonstrated in our Material Design guideline gallery. The guidelines are often abstract. Attaching conformance and violation UI design examples to the design guidelines allows the developers to compare and learn from concrete UI designs. Those examples come from two sources: first, the illustrative UI mockups provided in the design system documentation, and second, the real-world app UI screenshots analyzed by the UI design smell detector. For detecting UI design smells against design guidelines, a critical task is to define atomic UI information required for validating guideline violations. The violation condition can be defined as the first-order logic over the relevant types of atomic UI information. Fig. 4 shows three guideline conditions for navigation drawer. Atomic UI information can be extracted from the input UI design (see Section III-D), and validated against these guideline conditions (see Section III-E).

### C. Parsing Input UI Design Image

The input to our UI design smell detector is a UI design image. Our detector does not need source code for analysis. The UI design image may come from two sources: 1) a UI mockup created in design tools (e.g., Figma [15]) or GUI builders (e.g., Android Studio Layout Editor [16]); 2) an app UI screenshot taken at the runtime by UI testing framework (e.g., Android UI Automator [17] or Selenium [18]).

In this work, we are concerned with five types of component metadata in the UI design, including *component type*, *component bounding box* (top–left coordinate and bottom-right coordinate), *the number of component instances* (of a specific type or within a container component), *component composition hierarchy*, and *text content*. The input UI design image may contain such component metadata. For example, design tools can export UI mockups in vector graphics (e.g., SVG) with metadata. When taking an app UI screenshot, the corresponding UI component metadata can be exported in JSON file. Given an input UI design image with metadata, a file parser can be developed to extract the needed component metadata. When the input UI design is a pixel image, the user can manually annotate UI components and their metadata with

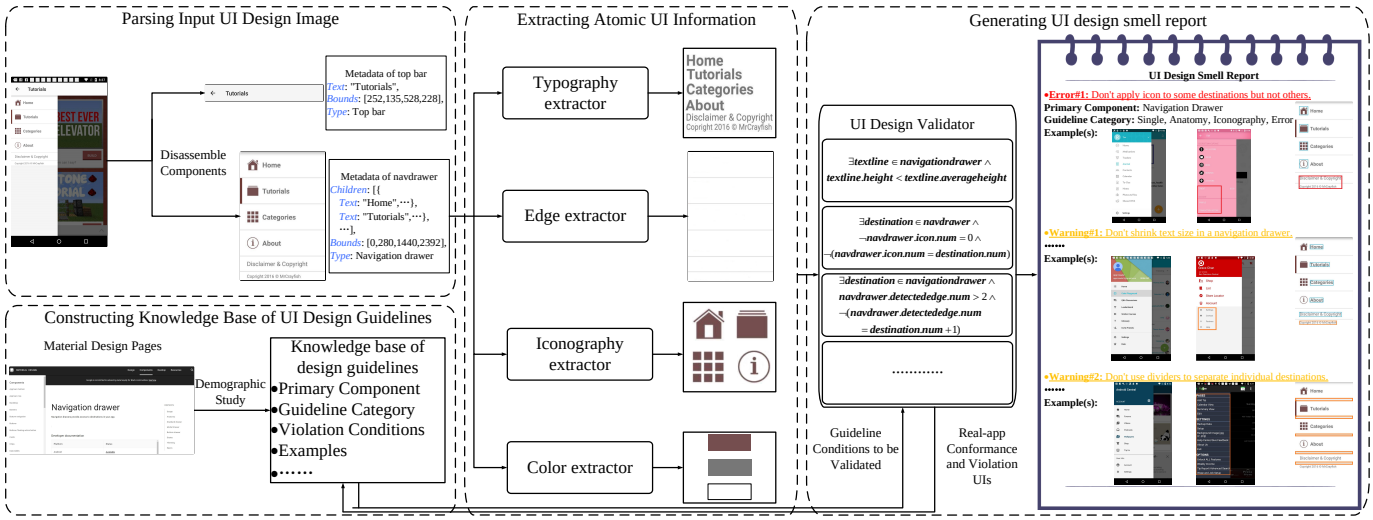



Fig. 4: Approach overview (illustrating the validation of a navigation drawer as an example)

an interactive tool. This annotation tool can support computer-vision based UI component detection techniques [19], which reduces the manual annotation effort. The user then only needs to modify or correct some detected UI components.

### D. Extracting Atomic UI Information

For each UI component in the input UI design, four types of atomic UI information will be extracted, including *typography*, *iconography*, *color*, and *edge*.

*1) Typography Extraction:* The typography extractor aims to recognize how text content (e.g., app bar title, button label, tab label) is actually rendered in the UI. When the input UI comes with the component metadata, the text content can be directly obtained from the component metadata. However, to validate the guidelines regarding text appearance (e.g., wrap, resize, truncation), we must know the number of text lines, the font size, and the presence of special characters (e.g., '...'), when the text is rendered on the UI. Inspired by a recent study on UI widget detection [19], we use EAST [20], [21] (a deep learning based scene text detection tool) to detect text region in the UI, and then use Tesseract OCR [22] to covert the cropped text images into text strings. If the text content is obtained from the component metadata, we match this text content with the OCRed strings to correct OCR errors. Finally, based on the detected text regions and the OCRed texts, we determine the number of text lines, the font height/width, and the presence of special character.

*2) Iconography Extraction:* The iconography extractor has two goals. First, it detects the presence of icons and images in the input UI. Second, it attempts to simulate how app users perceive the text, widget and image in the UI. We use computer-vision techniques to achieve these two goals. In particular, we use the UI widget detection tool [23] to detect non-text UI widget regions (e.g., icon, button), and use EAST [20] to detect text regions. The detected icons and images help to validate the guidelines regarding icon usage, such as "don't use same icons to represent different destinations in navigation drawer". Furthermore, by comparing the detected text, widget and image regions with the component bounding boxes in the metadata, we can determine illegibility-related UI design smells, for example, the two illegible buttons computer vision fails to detect in Fig. 1(b).

*3) Color Extraction:* The color extractor aims to identify primary color of the UI components and their constituent parts. In Fig. 4, we illustrate the primary color of the navigation drawer (white), and that of the icons (brown) and the text (gray) in the navigation drawer. Primary color is important for detecting color-related UI design smells, such as "don't use multiple colors in a bottom navigation bar", or "don't use primary color as the background color of text fields". To determine the primary color of the UI components, we adopt the HSV color space [24]. Unlike the RGB color model, which is hardware-oriented, the HSV model is user-oriented [25], based on the more intuitive appeal of combining hue, saturation, and value elements to create a color.

*4) Edge Extraction:* The edge extractor aims to detect the rendered edges of a UI component or the divider lines within a UI component. The detected edges help to validate divider- and shape-related guidelines. Although the coordinates of a component always form a rectangle, the actual edge of the component may not be a rectangle. We adopt Canny edge detection [26] to detect the edges surrounding or with a UI component. We require that the pixels must have sufficient horizontal and vertical connectivity. The edge extractor outputs the start and end coordinates of the detected edges.

### E. Validating UI Design against Visual Design Guidelines

For each UI component in the input UI design, the validator first retrieves the relevant design guidelines by the component type from the knowledge base. For each design guideline, it enters the relevant atomic UI information of the primary component and other related UI components into the guideline's violation condition. If the condition is satisfied, the design of the primary component violates this design guideline. Fig. 4 illustrates the validation of the three guidelines of navigation drawer, "don't apply icons to some destinations

but not others", "don't shrink text label", and "don't use dividers to separate individual destinations". The validation uses the component metadata, typography, iconography and edge information, and detects one error and two warning violations for the input navigation drawer.

After validating all UI components in the input UI, the validator produces a UI design smell report. Fig. 4 shows the UI design smell report for the input UI. For each component that has at least one design guideline violation, the report summarizes the list of design guidelines being violated. For each guideline being violated, the report highlights the corresponding component part(s) on the input UI in a red or orange box, depending on the violation severity (error or warning) of the design guideline. To assist the developers in understanding each reported guideline violation, the report presents conformance and violation UI examples. The corresponding component parts on the violation UI examples are also boxed to attract the attention.

## IV. Detection Accuracy by UIS-Hunter

We conduct two experiments to investigate the detection accuracy of UIS-Hunter on the real-app UIs from the Rico dataset [27] and the high-quality UI mockups from the Figma design kit. The two experiments also allow us to compare the severity of UI design smells in real-app UIs and UI mockups.

### A. Detection Accuracy for Real-App UIs

*1) Dataset:* We build a dataset of real-app UIs from the Rico dataset [27]. The Rico dataset contains 66,261 app UI screenshots collected from 9,286 Android apps. These app UI screenshots were collected mainly by the automated UI exploration tool [17]. A small portion was collected during manual exploration by human users. A UI screenshot is exported as a pixel image, together with the corresponding component metadata in a JSON file. In this work, we filter out UI screenshots that do not contain any of the 17 types of UI components to be validated, home screens that do not belong to any apps, and landscape UI screenshots. We obtain 60,756 unique app UI screenshots from 9,286 apps.

In these 60,756 app UIs, six types of components (divider, chip, date picker, tooltip, snackbar and bottom sheet) are used much less frequently than the other 11 types. Divider and chip are small decorative components, which have only two usage don't-guidelines. Date picker, tooltip, bottom sheet and snackbar usually need proper user interaction to trigger, which is hard to simulate by automated UI exploration. Date picker has only two guidelines. Tooltip, bottom sheet and snackbar have four, five and nine guidelines respectively, but there are only 264, 213 and 330 instances in the collected UIs. We manually examine the instances of these six types of components against their don't-guidelines, and do not find any guideline violations. Of course, this does not mean these six types of components are smell proof. But considering the data scarcity, we leave out these six types of components as future work.

TABLE I: Performance of detecting UI design smells for the 11 types of components in 60,756 real-app UIs (See Section IV-A2 for the explanation of column abbreviations)

| Component | #INST | #GL | #GLwRV | #RV | Prec. | Rec. | F1 |
|---|---|---|---|---|---|---|---|
| Button | 32,988 | 6 | 5 | 2,872 | 78% | 78% | 0.78 |
| Top Bar | 21,615 | 4 | 4 | 1,704 | 82% | 85% | 0.83 |
| Text Field | 15,297 | 4 | 2 | 1,575 | 88% | 96% | 0.92 |
| NavDrawer | 14,399 | 7 | 6 | 2,829 | 71% | 88% | 0.78 |
| List | 9,455 | 3 | 0 | 0 | - | - | - |
| Tab | 8,426 | 7 | 5 | 586 | 91% | 89% | 0.90 |
| BottomNav | 5,029 | 7 | 5 | 397 | 92% | 84% | 0.88 |
| Banner | 3,721 | 10 | 7 | 382 | 86% | 100% | 0.92 |
| FAB | 3,442 | 11 | 4 | 399 | 91% | 92% | 0.91 |
| Dialog | 2,837 | 5 | 5 | 1,740 | 86% | 99% | 0.92 |
| Bottom Bar | 1,790 | 7 | 2 | 137 | 92% | 89% | 0.90 |
| **Total** | **118,999** | **71** | **45** | **12,621** | **81%** | **90%** | **0.87** |

*2) Evaluation Method:* We apply UIS-Hunter to all instances of the 11 types of components in the 60,756 app UIs. As shown in Table I, these 11 types of components have 118,999 instances (#INST) and 71 guidelines (#GL). #RV shows the number of reported violations. Each reported violation identifies a primary component and the violated guideline. #GLwRV shows the number of guidelines with reported violation instances. We manually check the correctness of all reported violations. A reported violation is correct only if the reported component actually violates the reported guideline. Precision is the percentage of correctly reported violations (#TP) among all reported violations.

We have 50,829 app UIs for which UIS-Hunter does not report any violations. It requires significant manual effort to identify the missing violations in all these 50,829 app UIs. Therefore, we adopt a statistical sampling method [28] to sample and examine the minimum number *MIN* of app UIs that contain a particular type of component. We set $e = 0.05$ at 95% confidence level for determining MIN. That is, the estimated accuracy has 0.05 error margin at 95% confidence level. In total, we sample and examine 3,490 app UIs for the 11 types of components in Table I. Let FN be the violations in the sampled UIs (i.e., false negative ) for a type of component. We estimate the number of false positive violations in all app UIs by #EstFN=#FN/SampleRatio. SampleRatio is the number of sampled UIs divided by the total number of app UIs containing a type of component. Recall is #TP/(#TP+#EstFN). F1-score is the harmonic mean of precision and recall.

*3) Results:* As shown in Table I, UIS-Hunter achieves 0.81, 0.90 and 0.87 for the overall precision, recall and F1. The F1 for 6 types of components are above 0.9. The F1 for top bar and bottom navigation is 0.83 and 0.88 respectively. The F1 is below 0.8 for only button and navigation drawer. For the 3 guidelines of list, UIS-Hunter does not report any violations. Our manual examination of the sampled UIs with list components does not find any violations either.

Button has four variant types - text button, outlined button, contained button and toggle button. We rely on a button's metadata and visual appearance to determine the button type, which may not be always correct. The incorrect button type subsequently leads to the misjudgment of the applicability

TABLE II: Performance of detecting UI design smells for the nine types of components in 493 editing UI mockups

| Component | #INST | #GLwIV | #RV | Prec | Rec | F1 |
|---|---|---|---|---|---|---|
| Button | 272 | 1 | 4 | 100% | 100% | 1 |
| Text Field | 182 | 2 | 7 | 100% | 100% | 1 |
| BottomNav | 104 | 2 | 6 | 100% | 100% | 1 |
| List | 80 | 3 | 9 | 78% | 78% | 0.78 |
| NavDrawer | 70 | 1 | 4 | 100% | 100% | 1 |
| Bottom Bar | 66 | 5 | 16 | 88% | 93% | 0.90 |
| FAB | 59 | 7 | 21 | 95% | 100% | 0.97 |
| Tab | 46 | 2 | 6 | 100% | 100% | 1 |
| Banner | 31 | 3 | 9 | 100% | 100% | 1 |
| **Total** | **910** | **26** | **82** | **94%** | **0.95** | **0.95** |

of some guidelines, for example, "don't use too long text label in a text button" versus "an outlined button's width shouldn't be narrower than the button's text length". We find that UIS-Hunter may still accurately identify the buttons with some design smells (e.g., long text), but it may apply the wrong guideline. In our accuracy evaluation, we consider such reported violations as wrong. In fact, our user study (see Section V-A) shows that button variants often confuse human too when manually validating button designs.

The design of a navigation drawer can be rather complex, for example, with many icons and text labels, and sometimes even with search box or tabs. This poses the challenges of accurate extraction of typography, iconography and edge information. Consequently, UIS-Hunter may misjudge design violations. For example, if the iconography extractor fails to identify some icons in the navigation drawer, UIS-Hunter will misreport the violation of the guideline "don't apply icons to some destinations but other others". In fact, this guideline has the lowest precision (0.63). As another example, it is also challenging to accurately detect thin divider edges used in the navigation drawer. Inaccurate divider edges may lead to the wrong judgment of the guidelines like "don't use divider to separate individual destinations".

We finally obtain 7,497 unique UIs that contain true-positive design violations for the 45 don't-guidelines of 10 types of components. That is, 12.3% of the 60,756 app UIs contains at least one confirmed design violation against material design guidelines. 16% of 7,497 UIs contain more than one violations. Among the 9,286 apps, 2,587 (28%) apps have at least one UI design violation against Material Design guidelines. 24% of these 2,587 apps contain two design violations, and 45% of these 2,587 apps contain 3 or more design violations.

### B. Detection Accuracy for UI Design Mockups

*1) Dataset:* We download the Material Design Kit from the Figma website. Figma is a popular web-based UI design tool. This design kit contains 413 UI mockups that demonstrate the best Material Design practices. These design mockups contain all 17 types of UI components that UIS-Hunter can validate. The design mockups and their constituent components are all editable. We export these design mockups with component metadata in SVG (Scalable Vector Graphics) format.

In our evaluation on real-app UIs, UIS-Hunter does not report any violations for the 26 don't-guidelines of the nine types of UI components (see Table II). Our manual examination of the sampled UIs does not find the violations of these 26 guidelines either. Among these 26 guidelines, 6 guidelines are about improper text styles, 6 are about icon usage, 5 are about button usage inside a component, 4 are about styles when bottom bar and FAB are used together, 3 are about the count of certain component instances, 2 are about space and placement. For 17 of these 26 guidelines, UIS-Hunter actually reports the same-flavor guideline violations for other types of components. For example, although there are no violations of truncating text in bottom navigation, we find the text-truncation violations for top bar.

To confirm the UIS-Hunter's capability for the 26 guidelines without violations in real-app UIs, we take advantage of the editable UI mockups to inject UI design smells that violate these 26 guidelines in the UI mockups. For each guideline, we choose 2-4 UI design mockups that contain the primary component(s) involved in the guideline. We modify primary component(s) to simulate the guideline violation example illustrated in the Material Design documentation. Depending on the guideline, the modification may include changing text content and style, adding relevant components/icons, resizing or moving components, etc. We finally obtain 80 edited UI mockups that violate the 26 don't-guidelines. We combine these 80 edited UI mockups and the original 413 UI mockups for the accuracy evaluation on UI design mockups.

*2) Evaluation Method:* We manually examine all the original 413 UI mockups for the violation of the 71 guidelines of 17 types of components. We do not find any guideline violations. This is not surprising because these 413 UI mockups are created by professional designers for demonstrating the best Material Design practices. Nevertheless, we apply UIS-Hunter to these 413 original UI mockups and the 80 edited UI mockups for detecting the injected violations of the 26 don't-guidelines of the 9 types of components. We include the original UI mockups to increase the test cases, and see if UIS-Hunter may erroneously report the violations that do not actually exist in these high-quality UI mockups.

*3) Results:* Table II presents the results. Overall, UIS-Hunter achieves F1=0.95. It detects 77 of the 80 injected design violations, and reports five erroneous violations. UIS-Hunter achieves F1=1 for 11 guidelines of six types components. It also detects all injected violations for the seven FAB guidelines and the three banner guidelines. Three undetected violations are due to the failure of recognize relevant UI components (e.g., keyboard, icons). Five erroneously reported violations are due to the erroneous recognition of non-existent icons and the unusual distance between components.

## V. Utility of UIS-Hunter

Having confirmed the high precision and recall of our UIS-Hunter tool, we want to further investigate the utility of automated UI design smell detection from two aspects:

- How effective can front-end developers manually detect UI design smells reported by UIS-Hunter?

- How do app users rate the severity of UI design smells reported by UIS-Hunter?

### A. Effectiveness of Manual UI Design Smell Detection

*1) Motivation:* Using our UIS-Hunter, we identify a large number of UI design smells in real-world Android apps. This could be because developers are not aware of relevant Material Design guidelines. In this study, we want to further investigate whether developers can effectively detect UI design smells when they are made aware of relevant don't-guidelines. By contrasting this manual effectiveness with the detection capability of our tool, we will understand the importance and necessity of automated, consistent detection method.

*2) User Study Design:* We select from our Rico dataset app UIs according to the following four criteria. First, we want to cover as diverse primary UI components as possible. The selected app UIs should contain not only UI components with violations but also those without violations. Second, in addition to UIs with violation(s), there should also be UIs without any violations. Third, the violated guidelines should cover as many component design aspects and general design dimensions as possible, roughly with the similar proportion of the guidelines in each category (see Section II). They should also cover both warning and error severity. Finally, validating the selected app UIs against these guidelines should involve all types of atomic UI information. We also need to control the number of UIs to be examined in order to avoid fatigue effect by human annotators.

As a result, we select 40 app UIs, among which 27 UIs with violations (referred to as dirty UIs) and 13 UIs without any violations (clean UIs). These 40 UIs contain 1 to 5 types of UI components (median 3). 24 dirty UIs have one violation each, and 3 dirty UIs have two violations each, i.e., in total 30 instances of violations. We manually confirm that UIS-Hunter correctly identifies all violations, and reports no erroneous violations for the selected 40 app UIs. The 30 violations violate 18 don't-guidelines (see Table III), which in combination cover nine types of primary components, all four component design aspects, all six general design dimensions (except shape), and all five types of atomic UI information. 1 guideline (G7) has 6 violation instances, 2 guidelines (G12 and G13) have 3 violation instances each, 3 guidelines (G10, G16, and G17) have 2 violation instances each, and 12 guidelines have one violation instance each. Each type of involved primary component has both violation and conformance instances, denoted as #IWV and #INV respectively.

We recruited 5 front-end developers from a local company. These developers have at least 1 years web and/or mobile app development experience. None of the developers are involved in the development of UIS-Hunter. Before the study, we ask the developers to review and familiarize themselves with the 18 don't-guidelines and their conformance and violation examples (different from the 40 UIs to be examined). Then, we ask these 5 developers to independently examine the 40 app UIs and identify the violations of the 18 guidelines if they believe there are any. The developers do not know which UIs are dirty or clean, which guidelines have how many violation instances or may not have violations at all. The developers can use as much time as they wish to examine a UI. They complete the examination in 50 to 80 minutes (median 64 minutes).

TABLE III: Don't-guidelines examined in user studies and the results of detection effectiveness and severity rating (#IWV/#INV: the number of instances with/without violations)

| GID | Primary Component (#IWV, #INV) | Detection Effectiveness | | | Severity Rating #(Ratio%) | | |
|---|---|---|---|---|---|---|---|
| | | Prec. | Rec. | F1 | #1-2(%) | #3(%) | #4-5(%) |
| **G1** | Button (3, 17) | 0.57 | **0.8** | 0.67 | **4(80%)** | 1(20%) | 0(0%) |
| **G2** | | 0.2 | 0.2 | 0.2 | 2(40%) | 0(0%) | **3(60%)** |
| **G3** | | 0.2 | 0.4 | 0.3 | 0(0%) | 0(0%) | **5(100%)** |
| **G4** | Top Bar (3, 24) | **1** | **1** | **1** | **2(40%)** | **2(40%)** | 1(20%) |
| **G5** | | **0.8** | **0.8** | **0.8** | 1(20%) | 0(0%) | **4(80%)** |
| **G6** | | 0.2 | 0.2 | 0.2 | 1(20%) | 0(0%) | **4(80%)** |
| **G7** | NavDrawer (8, 3) | **0.93** | **0.9** | **0.91** | 11(37%) | 3(10%) | **16(53%)** |
| **G8** | | **0.83** | **1** | **0.91** | 2(40%) | 0(0%) | **3(60%)** |
| **G9** | | 0.57 | **0.8** | 0.67 | 0(0%) | 1(20%) | **4(80%)** |
| **G10** | Tab (6, 9) | 0.64 | 0.7 | 0.67 | 2(20%) | 3(30%) | **5(50%)** |
| **G11** | | 0.5 | 0.6 | 0.5 | 0(0%) | 0(0%) | **5(100%)** |
| **G12** | | **1** | **1** | **1** | 0(0%) | 2(13%) | **13(87%)** |
| **G13** | Text Field (3, 4) | 0.75 | **0.8** | 0.77 | 1(7%) | 1(7%) | **13(87%)** |
| **G14** | BottomNav (1, 5) | 0.4 | 0.4 | 0.4 | 1(20%) | **2(40%)** | **2(40%)** |
| **G15** | Dialog (1, 4) | **0.8** | **0.8** | **0.8** | 0(0%) | 1(20%) | **4(80%)** |
| **G16** | Banner (2, 6) | 0.05 | 0.1 | 0.07 | 1(10%) | 3(30%) | **6(60%)** |
| **G17** | FAB (3, 4) | **1** | **1** | **1** | 3(30%) | **5(50%)** | 2(20%) |
| **G18** | | **1** | **1** | **1** | **2(40%)** | 1(20%) | **2(40%)** |

* **G1**: don't use too long text label in a text button, **G2**: don't use two icons in a button, **G3**: an outlined button's width shouldn't be narrower than the button's text length, **G4**: don't shrink text in a top bar, **G5**: don't truncate text in a top bar, **G6**: don't wrap text in a regular top bar, **G7**: don't apply icons to some destinations but not others in a navigation drawer, **G8**: don't use the same icon to represent different primary destinations in a navigation drawer, **G9**: don't shrink text size in a navigation drawer, **G10**: don't mix tabs that contain only text with tabs that contain only icons, **G11**: don't truncate labels in a tab, **G12**: don't nest a tab within another tab, **G13**: don't use primary color as the background color of text fields, **G14**: don't use a bottom navigation bar for fewer than three destinations, **G15**: don't use dialog titles that pose an ambiguous question, **G16**: don't use a single prominent button in a banner, **G17**: don't display multiple FABs on a single screen, **G18**: don't include less than two options in a speed dial of FAB.

The developers are asked to report the identified violations in terms of the involved primary components and the violated guidelines. An identified violation is considered correct only if both the involved primary component and the violated guideline are correct. We compute precision and recall to evaluate manual detection effectiveness. Let $V_R$ be the bag of the reported violation components for a don't-guideline by the 5 developers, and $V_T$ be the bag of the ground-truth violation components for this don't guideline (we duplicate each component five times). Precision is $(|V_R \cap V_T|)/|V_R|$, and recall is $(|V_R \cap V_T|)/(|V_T|)$, where $|.|$ is the bag cardinality.

*3) Results:* The Detection Effectiveness column in Table III shows the results. If the 5 developers detect all 30 violation instances, we expect 150 (30*5) reported violations, i.e., $\sum_{i=1}^{30} |V_T|_i$=150. The five developers report in total 154 violations (i.e., $\sum_{i=1}^{30} |V_R|_i$=154), among which only 117 reported violations are true violations (i.e., $\sum_{i=1}^{30} |V_R \cap V_T|_i$ = 112). Therefore, the overall detection precision 75.5%, and the overall recall is 71.9%. Both precision and recall are worse than respective metrics of our UIS-Hunter (see Table I).

Among 18 don't-guidelines, only 5 guidelines (G4, G8, G12, G17 and G18) have all violation instances detected by all five developers (i.e., recall=1). 8 guidelines (G1, G5, G7, G9, G10, G11, G13 and G15) have their violations detected by three or more developers (i.e., recall$\geq$0.6). When primary

components are large UI components (e.g., navigation drawer, tab) or distinct components (e.g., FAB), their design violations are easier to notice by human, for example, a navigation drawer uses the same icons for different destinations (G8), one tab component is nested in another tab (G12), some tabs contain only text while others contain only icons (G10), and multiple FABs in a single screen (G17). Some improper text styles, for example too long text (G1), text shrinking (G4, G9), text truncation (G5, G11), are also relatively easier to notice. Color-relation violations, such as using primary color as the background color of text fields (G13), are also infamous, and easy to spot from illegible components.

For the 5 guidelines (G2, G3, G6, G14 and G16), the majority of the developers fail to detect their violation instances (i.e., recall$\leq$0.4). For G2, G6 and G16, only one developer detects one violation instance. Compared with the guidelines whose violations are easy to spot, these five guidelines demand careful examination of a component's type and its constituent parts. For example, G2 and G14 require counting the number of icons in a button and the number of destinations in a bottom navigation, respectively. G3 needs to check if the button is an outlined button and then compare the button's outline box and the text length. G5 requires to determine if the top bar is a regular or prominent top bar. Wrapping text is fine in prominent top bar, but not in regular top bar. G16 requires to check the presence of dismiss button in a banner. Furthermore, we find that the developers may overlook some components in a complex UI. For example, 4 developers did not notice that a button has two icons, because they were distracted by other components like the banner on the UI.

Two guidelines (G12 and G17) have all reported violations correct (i.e., precision=1). The precision for G4, G5, G7, G8, G13 and G15 is also high ($\geq$0.7). All these guidelines have clear distinction between violation and conformance. For example, G12 is about nested tab and G17 is about multiple FABs on a single screen. It is unlikely to mistake conformance as violation, and vice versa. It is also not difficult to distinguish proper text styles (G4 and G5), icon usage (G7 and G8), color usage (G13) and actions (G15) from style, usage and action violation. But when a component has variant types, developers often misjudge a guideline's applicability, and report erroneous violations. For example, G1 and G3 are similar, but G1 is for text button while G3 is for outlined button. The developers often mistake the two types of buttons and report violations for the wrong button type. Or when a prominent top bar is mistaken as a regular top bar, the developers may report text wrapping in top bar as a violation of G6.

*Manual detection of UI design smells is less effective than automated detection, especially when multiple pieces of information need to be integrated or there are component-variant-sensitive guidelines.*

### B. Severity Ratings of UI Design Smells

*1) Motivation:* Material Design guidelines are a natural response to many poor app UI designs that lead to low-quality user experiences. In this study, we want to investigate how ordinary app users think of the UI design smells reported by our UIS-Hunter. This helps us understand the potential impact on app users by the tools like UIS-Hunter which can flag these UI design smells and inform developers to take actions.

*2) User Study Design:* We use the 27 UIs with violations from the first user study. We highlight the primary components with violations on these UIs and annotate the violation with the guideline explanation. We recruit 3 male and 2 female regular Android users from our school, the age of these users ranges from 21 to 30. We ask the 5 users to independently rate the severity of each violation in the 27 UIs. We use 5-point Likert scale, with 1 being the least severe and 5 being the most severe. For each guideline, we summarize the number of 1-2 ratings as non-severe, 3 ratings as neural, and 4-5 ratings as severe.

*3) Results:* The Severity Ratings column in Table III shows the results. Only three guidelines (G1, G4 and G17) have 1-2 ratio higher than 4-5 ratio. Among these three guidelines, only G1 has a large ratio difference (0.8 versus 0). That is, 80% ratings are non-severe, and there are no severe ratings. Although long text label is not recommended, it does not severely affect the visual effect and the usage of the text button. Therefore, 4 out of 5 users do not consider the G1 violation as severe, 1 user rates it as neural.

In contrast, 14 guidelines have 4-5 ratio higher than 1-2 ratio, among which 9 guidelines (G3, G5, G6, G9, G11, G12, G13, G15 and G16) have large ratio difference (at least 4 times higher). For 8 of these 9 guidelines, less than 20% ratings are non-severe. For 5 guidelines (G3, G9, G11, G12, G15), there are no non-severe ratings at all. We summarize three factors that often lead to severe-rating violations. First, some violations significantly affect the UI look and feel. For example, when G3 is violated, the text label goes outside the box of the outlined button. Second, improper text style (e.g., text truncation (G5, G11), text wrapping (G6)) and color usage (e.g., primary color text fields (G13)) significantly affect the text legibility and the understanding of important information, which are treated as severe issues by most users. Third, some violations significantly affect the interaction with the applications, such as nested tab (G12), ambiguous dialog title (G15), and single prominent button in a banner (G16).

There are seven guidelines (G2, G7, G8, G10, G14, G16, and G18) that we could consider as controversial, because these guidelines have close non-severe versus severe ratings ratios. Indeed, the manual detection efficiency for some controversial guidelines (e.g., G2, G14 and G16) is relatively lower. This was some developers do not even consider them as UI design smells because the violations of these guidelines do not significantly affect UI look and feel or app usage. For example, use two icons in a button (G2), display multiple FABs (G17), have one option in the speed dial of FAB (G18), mix tabs with only text and only icons (G10), use bottom navigation for two destinations (G14). So it is not surprising that the users have mixed opinions. Having said that, we believe Material design puts up these guidelines (albeit seemingly controversial) for

good reasons. Our automatic guideline conformance checking tool can raise the developers' attention to potential UI design smells, contrast these smells against specific design guidelines, and then the developers can make the informed decision whether they are smells to fix or some unique designs to keep.

We also find apps from big companies also violate some of these guidelines. For example, booking.com's Android app does apply icons to some destinations but not others in the navigation drawer (G7). Those destinations without icons are auxiliary features (e.g., help, contact us). Without-icons actually helps to distinguish auxiliary features from main booking-related features.

We find that severity ratings may be context-sensitive. For example, both G4 and G9 are about text shrinking. The users consider the shrinking of app bar title (G4) less severe, because it does not interfere with other components and the text may still be large enough to read. But they consider the shrinking text in a navigation drawer (G9) much more severe, because it leaves the text label inconsistent with other text labels in the drawer. Severity ratings may also be app sensitive. For example, the UI we select for G8 uses the same icon for three different calculators the app supports. As they are all about calculator, some users believe it is fine to use the same icon.

## VI. Related Work

UX Check [29] identifies usability issues against Nielsen's ten usability heuristics. Wu et al. [13] diagnose user engagement issues caused by mobile UI animation.These tools check only general principles. In contrast, UIS-Hunter checks specific guidelines. Checkman [30] also checks specific guidelines but only iOS layout guidelines. Both seenomaly [12] and our UIS-Hunter detect Material Design guideline violations. Seenomaly validates only UI animation guidelines, while UIS-Hunter checks design guidelines across five component design aspects and seven general design dimensions.

Many tools focus on specific UI issues, such as icon tapability [31], image accessibility [32], [33], limited types of display issues (e.g., component occlusion, text overlap, blurred screen) [34], or mismatch between intension and practice [35], [36]. To the best of our knowledge, our tool is the first of its kind. The issues reported by these tools and the guideline coverage are not really comparable to our tool. Consider entering the navigation drawer design in Fig 4 into Google Accessibility Scanner. That scanner reports issues: increase text contrast and enlarge small touch targets. These two issues are similar to our guideline violation "don't shrink text size". But our tool can detect many other guideline violations such as "don't apply icons to some destination but not others", "don't use dividers to separate individual destinations", while the scanner cannot. Also, all these tools rely on deep learning models, which demand large datasets of labeled UI issues for model training. It is impractical to collect sufficient training data for all 71 guidelines of 11 types of components UIS-Hunter validates. Furthermore, some guidelines (e.g., minimum/maximum destinations in a bottom navigation, repeated icons in top and bottom bar, mixing icons/text in tabs) require aggregating specific component information explicitly. Our UIS-Hunter only uses deep learning tools to extract atomic typography and iconography information, and then use explicit logic formulas to aggregate the extracted information.

Many GUI testing tools [37], [38], [39] test system functionalities through automated GUI exploration, some methods [40], [41], [42] compare the rendered DOM trees to detect cross-browser compatibility issues, but UI visual defects receive little attention. Some tools touch text inconsistencies and defects [35], [36], [43], similar to some text style don't-guidelines our tool supports. Moran et al. [10], [44] develops techniques for detecting presentation inconsistencies between UI mockups and implemented UIs, and the UI changes during evolution. Their methods contrast two similar UIs and find their differences. This setting is not applicable to detecting guideline violations where illustrative UIs for the guidelines and the UIs to be checked bear no similarity.

Our UIS-Hunter is remotely related to code linters for detecting program errors and code smells. For example, FindBugs [7] identified common program errors, such as null reference, useless control flow. Stylelint [7] is a linter that detects errors and inconsistencies in Cascading Style Sheets. Some tools have been developed to detect code smells for refactoring, for example, code clones [45], [46] and feature envy [47]. All these code linters detect issues in source code, while UIS-Hunter detects visual design smells in UIs.

## VII. Discussion

This work focuses on explicit don't-guidelines. But we find many implicit design guidelines (e.g., "icons should be placed to the left of text labels in an extended FAB") without illustrative violation examples. Explicit and implicit guidelines differ only in how they are described in material design documentation, but not in component design aspects, general design dimensions, and atomic UI information involved. Therefore, implicit design guidelines, once discovered, can be supported in the same way as those explicit guidelines. Furthermore, we could derive some de-facto design guidelines from real apps based on the principles of official guidelines. For example, Material Design describes 4 guidelines regarding the inconsistent use of icons and text labels in bottom navigation. In our observation of real-app UIs, we find some inconsistent icon/text usage beyond these guidelines. For example, one destination uses a notification badge to indicate an update, while the other destination uses normal text for the same purpose. UIS-Hunter can be extended to support such de-facto guidelines just like those official guidelines.

Material Design components are backed by Android GUI APIs. Although these APIs support well the visual effects and interactions of material design, only few APIs enforce the don't-guidelines of the corresponding components. For example, TabLayout does not support vertical tab, which is a don't-guideline of tab component. Furthermore, Banner API does not support displaying several banners on a single screen, which is a don't-guideline of banner component. In fact, neither UIS-Hunter nor our manual examination identifies the

violation of these two guidelines in our dataset of real-app UIs. This indicates that well designed GUI APIs could prevent developers from making mistakes. We could back-trace UI design smells reported by UIS-Hunter to the source code and summarize visual-design related code smells (a new type of code smells that have never been explored). Studying such code smells could inform the design of GUI APIs that can better enforce UI design guidelines.

Last but not least, in addition to be an after-fact detector, UIS-Hunter could also be integrated with UI design tools and GUI builders to support just-in-time UI design smells analysis, in the similar way as just-in-time code smell analysis and refactoring [48]. For example, at the time the developer designs a bottom navigation with two or five destinations, the tool could raise the issue and suggest to use tab or navigation drawer instead. This could avoid significant redo cost or leave-it-as-is regret after the design has been implemented.

## VIII. Conclusion and Future Work

This paper presents an automated UI design smell detector (UIS-Hunter). The design of UIS-Hunter is informed by a demographic study of Material Design guidelines. The evaluation on real-app UIs and UI mockups confirms the high detection accuracy of UIS-Hunter, and the user studies provide initial evidence of the UIS-Hunter's utility for developers and app users. We release our UIS-Hunter tool, Material Design guideline gallery, manually validated UI design smell dataset for public use and evaluation. In the future, we will extend UIS-Hunter to support implicit and de-facto guidelines in Material Design, as well as other design systems that describe visual do/don't-guidelines for a library of UI components in a similar vein. We will investigate visual-design related code smells and better GUI API design to enforce visual design guidelines in code. We will integrate UIS-Hunter with design tools to support just-in-time UI design smell detection. More user studies will be conducted to evaluate the usefulness of the UIS-Hunter tool and its extensions.

## Acknowledgements

This research was partially supported by the National Key R&D Program of China (No. 2019YFB1600700), Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100021), ARC Discovery grant (DP200100020), and National Science Foundation of China (No. U20A20173).

## References

[1] "Funkyspacemonkey," 2020. [Online]. Available: https://www.funkyspacemonkey.com/7-reasons-people-uninstall-apps

[2] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for software design smells: managing technical debt*. Morgan Kaufmann, 2014.

[3] "Design systems gallery," 2020. [Online]. Available: https://designsystemsrepo.com/design-systems/

[4] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[5] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "Detecting bad smells in source code using change history information," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 268–278.

[6] H. Liu, Q. Liu, Z. Niu, and Y. Liu, "Dynamic and automatic feedback-based threshold adaptation for code smell detection," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 544–558, 2015.

[7] "Findbugs," 2020. [Online]. Available: https://github.com/findbugsproject/findbugs

[8] "Checkstyle," 2020. [Online]. Available: https://checkstyle.sourceforge.io/

[9] "Stylelint," 2020. [Online]. Available: https://stylelint.io/

[10] K. Moran, B. Li, C. Bernal-Cárdenas, D. Jelf, and D. Poshyvanyk, "Automated reporting of gui design violations for mobile apps," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 165–175.

[11] "ebay gui testing," 2020. [Online]. Available: https://tech.ebayinc.com/research/gui-testing-powered-by-deep-learning/

[12] D. Zhao, Z. Xing, C. Chen, X. Xu, L. Zhu, G. Li, and J. Wang, "Seenomaly: Vision-based linting of gui animation effects against design-don't guidelines," in *42nd International Conference on Software Engineering (ICSE'20). ACM, New York, NY*, 2020.

[13] Z. Wu, Y. Jiang, Y. Liu, and X. Ma, "Predicting and diagnosing user engagement with mobile ui animation via a data-driven approach," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[14] "Material design," 2020. [Online]. Available: https://material.io/

[15] "Figma: the collaborative interface design tool," 2020. [Online]. Available: https://www.figma.com//

[16] "Android studio layout editor," 2020. [Online]. Available: https://developer.android.com/studio/write/layout-editor

[17] "Android ui automator," 2020. [Online]. Available: https://developer.android.com/training/testing/ui-automator

[18] "Selenium," 2020. [Online]. Available: https://www.selenium.dev/

[19] M. Xie, S. Feng, , J. Chen, Z. Xing, and C. Chen, "Uied: A hybrid tool for gui element detection," in *Proceedings of the 2020 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[20] "East text detection," 2020. [Online]. Available: https://github.com/argman/EAST/

[21] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: an efficient and accurate scene text detector," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 5551–5560.

[22] "Tesseract ocr," 2020. [Online]. Available: https://tesseract-ocr.github.io/

[23] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, and L. Zhu, "Object detection for graphical user interface: Old fashioned or deep learning or a combination?" in *Proceedings of the 2020 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[24] C. Chen, S. Feng, Z. Xing, L. Liu, S. Zhao, and J. Wang, "Gallery dc: Design search and knowledge discovery through auto-created gui component gallery," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–22, 2019.

[25] S. Douglas and T. Kirkpatrick, "Do color models really make a difference?" in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1996, pp. 399–ff.

[26] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[27] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "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*, 2017, pp. 845–854.

[28] R. Singh and N. S. Mangat, *Elements of survey sampling*. Springer Science & Business Media, 2013, vol. 15.

[29] "Ux check," 2017. [Online]. Available: https://www.uxcheck.co/

[30] "Checkman," 2018. [Online]. Available: https://apps.apple.com/us/app/checkman-the-mobile-app-design-checker/id1247361179?ls=1

[31] A. Swearngin and Y. Li, "Modeling mobile interface tappability using crowdsourcing and deep learning," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–11.

[32] C. S. Jensen, M. R. Prasad, and A. Møller, "Automated testing with targeted event sequence generation," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 67–77.

[33] "Google accessibility scanner," 2020. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=en

[34] Z. Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "Owl eyes: Spotting ui display issues via visual understanding," in *Proceedings of the 35th International Conference on Automated Software Engineering*, 2020.

[35] S. Mahajan and W. G. Halfond, "Detection and localization of html presentation failures using computer vision-based techniques," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–10.

[36] S. Mahajan, B. Li, P. Behnamghader, and W. G. Halfond, "Using visual symptoms for debugging presentation failures in web applications," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2016, pp. 191–201.

[37] A. Swearngin, C. Wang, A. Oleson, J. Fogarty, and A. J. Ko, "Scout: Rapid exploration of interface layout alternatives through high-level design constraints," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[38] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu, and Z. Su, "Large-scale analysis of framework-specific exceptions in android apps," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 408–419.

[39] R. Yandrapally, A. Stocco, and A. Mesbah, "Near-duplicate detection in web app model inference," in *ACM*, 2020, p. 12.

[40] A. Mesbah and M. R. Prasad, "Automated cross-browser compatibility testing," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 561–570.

[41] S. R. Choudhary, M. R. Prasad, and A. Orso, "Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 171–180.

[42] S. R. Choudhary, H. Versee, and A. Orso, "Webdiff: Automated identification of cross-browser issues in web applications," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–10.

[43] R. Mahajan and B. Shneiderman, "Visual and textual consistency checking tools for graphical user interfaces," *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 722–735, 1997.

[44] K. Moran, C. Watson, J. Hoskins, G. Purnell, and D. Poshyvanyk, "Detecting and summarizing gui changes in evolving mobile apps," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 543–553.

[45] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: a multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[46] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1157–1168.

[47] H. Liu, Z. Xu, and Y. Zou, "Deep learning based feature envy detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 385–396.

[48] P. Alves, D. Santana, and E. Figueiredo, "Concernrecs: finding code smells in software aspectization," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 1463–1464.