UIS-Hunter: Detecting UI Design Smells in Android Apps

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*—Similar to code smells in source code, UI design has visual design smells that indicate violations of good UI design guidelines. UI design guidelines constitute *design systems* for a vast variety of products, platforms, and services. Following a design system, developers can avoid common design issues and pitfalls. However, a design system is often complex, involving various design dimensions and numerous UI components. Lack of concerns on GUI visual effect results in little support for detecting UI design smells that violate the design guidelines in a complex design system. In this paper, we propose an automated UI design smell detector named UIS-Hunter (UI design Smell Hunter). The tool is able to (i) automatically process UI screenshots or prototype files to detect UI design smells and generate reports, (ii) highlight the violated UI regions and list the material design guidelines that the found design smells violate, and (iii) present conformance and violation UI design examples to assist understanding. This tool consists of a Material Design guidelines gallery website and a tool website. The gallery website is a back-end knowledge base that attaches conformance and violation examples to abstract design guidelines and allows developers and designers to explore the multi-dimensional space of a complex design system in a more structured way. As a front-end application, the tool website takes a UI design as input, returns a detailed UI design smell report, and marks the violation regions (if any). Moreover, the tool website presents conformance and violation examples based on the gallery website.**

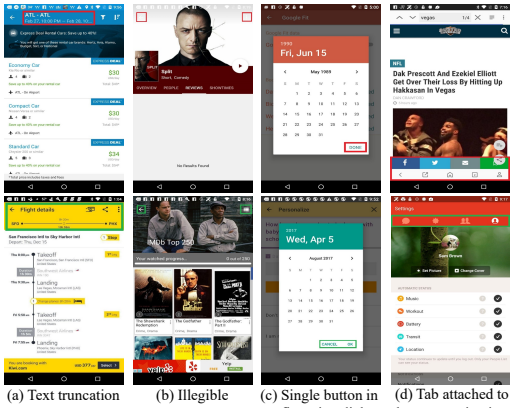
Demo URL: https://uishunter.net.cn/
https://uishuntergallery.net.cn/
Demo Video: https://youtu.be/7UZ0jtD_1gM

***Index Terms*—GUI testing, UI design smell, Violation detection, Material design**

I. Introduction

Graphical User Interface (GUI) supports the interaction between users and software applications. With a poorly designed Android GUI, users would feel frustrated and uninstall the application. We refer to poorly designed UIs as *UI design smells*. For example, truncating the full title of the top app bar in Fig. 1(a) hinders the access of key information (i.e., end time of a trip). In Fig. 1(b), the background image makes illegible foreground buttons/text. In Fig. 1(c), the confirmation dialog provides only a single action, and cannot be dismissed. Attaching tabs to bottom navigation in Fig. 1(d) can cause confusion about what action or tab controls which content.

Similar to code smells that indicate violations of good software design guidelines [1], UI design smells violate good



(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)

UI design guidelines. For a vast variety of products, platforms, and services, *design systems* are organized with UI design guidelines. We take Google Material Design as a case study of design systems. We investigate 130 guidelines for 23 UI components (e.g., app bar, banner, button, tab, and dialog) from Material Design documentation. These guidelines are crucial for creating intuitive and consistent user experience [2], and for improving digital accessibility [3]. The design guidelines go far beyond UI aesthetics.

However, previous research show little concern for detecting UI design smells that violating the design guidelines. Many tools have been developed to detect and remedy code smells [1], [4], [5], [6] and other programming or stylistic errors [7], [8]. Some techniques focus on detecting the inconsistencies between UI mockups and implemented UIs [9], or discern abnormal designs from normal ones [10], [11]. To the best of our knowledge, no technique can check the violation of a UI design against the visual guidelines in a design system. First, the design examples used to explain the guidelines are different from the actual UI designs (e.g., Fig. 1), so examples 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 informa-

tion (e.g., type, instance count, size/position, color, and text content) and the actual rendering of text and images.

In this work, we develop an automated tool (called *UIS-Hunter*) for detecting UI design smells against guidelines of Material Design. The tool is able to (i) automatically process UI screenshots or prototype files to detect UI design smells and generate reports, (ii) highlight the violated UI regions and list the material design guidelines that the design smells violate, and (iii) present conformance and violation UI design examples to assist understanding. The tool consists of Material Design guideline gallery and a tool website. The gallery website is a back-end knowledge base that collects a set of conformance and violation UIs from real Android apps for each supported visual guideline. Through the gallery website, users can explore the multi-dimensional space of material design system in a structured way. The tool website is used as a front-end application. Given a UI design, the tool website reports UI design smells when the corresponding guideline violation conditions are satisfied. The tool website also produces a detailed violation report (see Fig. 2) that lists the violated guidelines, highlights the regions violating these guidelines, and provides conformance and violation examples to assist developers understanding.

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, a recall of 0.90, and F1-score of 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 users.

II. Approach

A. Approach Overview

Fig. 2 show an overview of our approach framework. Our approach composes by 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*.

B. Constructing Guideline Knowledge Base

Material design are well-structured and adopt consistent writing conventions. To construct a knowledge base of UI design guidelines, we collect and examine guidelines explicitly marked as "don't" and "caution" and illustrated with at least one UI design example. Then, we identify the information about primary component, component design aspects, and design dimensions from document headings. Finally, we read the description and the example(s) of each guideline, and iteratively summarize the types of atomic UI information involved. The purpose of these steps is to index each guideline and organize them in multi-dimensional perspective.

C. Parsing Input UI

Our approach takes a UI design image without source code as input. The input UI shall be parsed to extract five types of component metadata, 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*. For an app UI screenshot, the corresponding UI component metadata can be exported in JSON file. For mockups, design tools can export SVG file. When the input UI design is a pixel image, we provide an interactive tool for users that can manually annotate UI components. This annotation tool can support computer-vision based UI component detection techniques [12]. This tool aims to reduce the manual annotation effort. The user then only needs to modify or correct some detected UI components.

D. Extracting Atomic UI Information

The tool extracts four types of atomic UI information that are *iconography*, *typography*, *color*, and *edge*.

- **Iconography.** The iconography extractor detects the presence of icons and images in the input UI. To simulate how app users perceive the text, widget and image in the UI, we use the UI widget detection tool [13] to detect non-text UI widget regions (e.g., icon, button), and use EAST to detect text regions.
- **Typography.** The typography extractor aims to recognize how text content (e.g., app bar title, button label, tab label) is actually rendered in the UI. We use EAST [14] (a deep learning based scene text detection tool) to detect text region in the UI, and then use Tesseract OCR to covert the cropped text images into text strings.
- **Color.** The color extractor aims to identify primary color of the UI components and their constituent parts. Primary color is important for detecting color-related UI design smells. In detail, we adopt the HSV color space [15] to determine the primary color of the UI components.
- **Edge.** The edge extractor aims to detect the rendered edges of a UI component or the divider lines within a UI component. Divider- and shape-related guidelines often need edge information to validate. We adopt Canny edge detection [16] to detect the edges surrounding or with a UI component. We require that the pixels must have sufficient horizontal and vertical connectivity.

E. Validating UI Design against Guidelines

The validator would match the retrieved information of the input UI design with the guidelines' violation condition from the knowledge base. For each design guideline, the validator enters the relevant atomic UI information of components into the guideline's violation condition. When the condition is satisfied, the component of the input UI would be reported as violation. Fig. 2 illustrates the validation of the three guidelines of navigation drawer.

After validating all UI components in the input UI with related guidelines , the validator produces a UI design smell report. The report shall 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 (i.e., error) or orange box (i.e., warning). Conformance and violation UI examples are also attached to

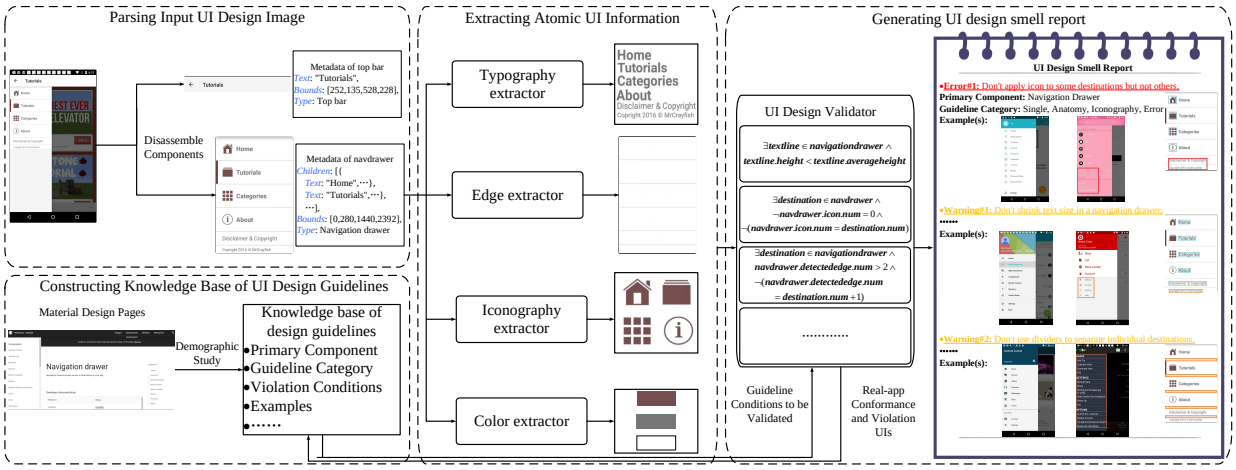


Fig. 2: Approach overview (illustrating the validation of a navigation drawer as an example)

each guideline to assist the developers in understanding each reported guideline violation. The corresponding component parts on the violation UI examples are also boxed to attract the attention.

III. Tool Implementation and Usage

A. Material Design Guideline Gallery and Knowledge Base

Material Design guidelines gallery is a back-end knowledge base for UIS-Hunter. Each design guideline is indexed by the type of primary component involved in the guideline, and annotates the guideline with multi-dimensional information such as component design aspect, design dimension, guideline severity, and needed atomic UI information. The gallery attaches conformance and violation examples to abstract design guidelines and records the violation condition can be defined as the first-order logic over the relevant types of atomic UI information. The gallery collects 130 guidelines for 23 types of UI components and accesses users to explore the multi-dimensional space of the material design in a structured way.

The website are implemented on Vue.js framework. Users can filter interested guidelines with primary component, general design dimension, component design aspect, and severity or send a textual keyword query like Fig. 3 shows.

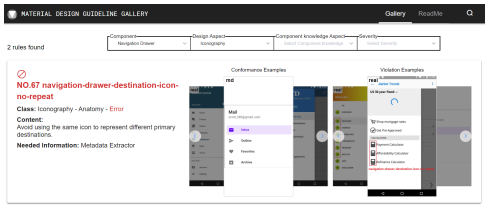


Fig. 3: The homepage of Material Design Guidelines Gallery

B. UIS-Hunter Implementation

UIS-hunter is implemented as a web application. Users can upload their UI data in two types: 1) a UI mockup created in design tools (e.g., Figma) or GUI builders (e.g., Android Studio Layout Editor); 2) an app UI screenshot taken at the runtime by UI testing framework (e.g., Android UI Automator). The tool shall process the input UI data in the background and return a UI design smell report, including a list of violated guidelines, violated regions, conformance and violation examples, and timing. A UI design smell report lists each violated guideline with detailed information. First, the color of violated design guidelines indicates the severity of the UI design smell. The primary component and category information illustrate the result of our demographic study. Then, UIS-Hunter snapshots a violated region on the origin UI and highlights the key elements for validating. Finally, UIS-Hunter queries the knowledge base and exhibits corresponding conformance and violation examples from real apps. When UI examples are more than two, examples shall be displayed in a carousel.

C. Usage Scenarios

We present several examples to illustrate how UIS-Hunter would help developers and designers. For example, someone like A is a newcomer to Android development and A wants to develop his/her own interface consisting of a top bar, a navigation drawer, and several buttons. First, A doesn't know how to design this interface and whether need to follow some design conventions. In this case, A can refer to Material Design guideline gallery, filter guidelines by primary component, and get inspiration from conformance examples in guideline of navigation drawer. Then, A has an initial idea and chooses Figma to prototype this idea. After prototyping, A is not sure whether there are design violations in the interface, but A could upload their mockup data to validate the initial design. In another situation, someone like B is confident in his/her work and implement the interface without checking the mockup. B can also upload the UI screen and hierarchy file from source code to get improvement suggestions. Following the preceding procedure, A and B can finally design a user-friendly interface without an experienced UI designer's check.

IV. Evaluation

A. Quantitative Evaluation

We conduct two experiments to investigate the efficiency of UIS-Hunter on the real-app UIs from the Rico dataset [17]. We build a dataset of real-app UIs by filtering out 60,756 unique UI screenshots of 9,286 apps. We remove UI screenshots like home screens that do not contain any components, and

landscape UI screenshots. We manually examine the reported 12,621 violations instances and adopt a statistical sampling technique [18] to examine the minimum number *MIN* of the rest of 50,829 app UIs without reported violations. The estimated accuracy has 0.05 error margin at 95% confidence level and we sample and examine 3,490 app UIs in total. UIS-Hunter achieves 0.81, 0.90, and 0.87 for the precision, recall and F1, respectively. Based on the manually validated detection results, UIS-Hunter detects 7,497 unique UIs that contain true-positive design violations for the 45 don't-guidelines of 10 types of UI components. UIS-Hunter detects all violation examples in Fig. 1. The results indicate 12.3% of the 60,756 app UIs contains more than one confirmed design violation. In addition, 16% of 7,497 contain more than one violations. Among the 9,286 apps, 6,699 (72%) apps have more than two design violations.

B. User Study

To further investigate whether developers can effectively detect UI design smells and how ordinary app users consider the reported UI design smells, we recruit 5 front-end developers to distinguish 27 identified violation UIs from total of 40 UIs, and ask 3 male and 2 female app users to independently rate the severity of each violation in the 27 UIs. The results can be concluded that manual detection of UI design smells has lower overall precision and recall than automated detection and 15 out of 18 guidelines are considered affecting the experience seriously by majority of users. Five guidelines have recall less than 0.4. The result indicates developers fail to detect these five guidelines. For example, "an outlined button's width shouldn't be narrower than the button's text length" needs to check whether the button is outlined and then compare lengths. 14 guidelines are considered severe like "don't mix tabs that contain only text with tabs that contain only icons". We also find some controversial guidelines (i.e., close non-severe versus severe ratings ratios). For example, booking.com's Android app does apply icons to some destinations but not others in the navigation drawer. Some users think without icons actually helps to distinguish auxiliary features (e.g., help, contact us) from main booking-related features.

V. Conclusion and Future Work

In this paper, we present *UIS-Hunter*, a tool for automatically detecting UI design smells against a wide range of design guidelines in Material Design, along with a *Material Design guidelines gallery*, a gallery presenting a demographic study results of Material Design guidelines and instantiating each guideline with conformance and violation UIs examples. Through the UIS-Hunter tool, developers can upload their UI designs in two formats (SVG from design tools and JSON from UI testing framework) and receive a detailed UI design smell report. Material Design guidelines gallery allows designers and developers to search and filter the guidelines and examples users are interested in and reduce the time cost on browsing the complex Material Design website. In the future, we plan to extend UIS-Hunter to support implicit guidelines in Material Design and de-facto guidelines emerging from real-world apps, as well as other design systems that describe visual do/don't-guidelines for a library of UI components in a similar vein. We shall also integrate UIS-Hunter with UI design tools to support just-in-time UI design smell detection.

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

[2] "How google material design affects mobile app design," 2018. [Online]. Available: https://www.businessofapps.com/news/how-google-material-design-affects-mobile-app-design/

[3] "The designer's guide to accessibility research," 2018. [Online]. Available: https://design.google/library/designers-guide-accessibility-research/

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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